TECNICATURA SUPERIOR EN TELECOMUNICACIONES

ARQUITECTURA Y CONECTIVIDAD

Módulo II: Familia de Protocolos IoT – II

TRABAJO PRÁCTICO N°: 4

Profesor: Ing. Jorge Elías Morales.

Alumnos:

- Huk, Romina vanesa - GitHub: RoHu17
- Roldán, Patricio Leandro - GitHub: pleroldan
- Pantoja, Paola Natalia Alejandra - GitHub: PaolaaPantoja
- Paez, Tiziano Adrian - GitHub: tpaez
- Gutiérrez, Emma - GitHub: Emygut

Actividad

1) ¿Qué es un protocolo COAP?, ¿Para qué se usa? Ejemplifique.

2) ¿Qué es un protocolo AMQP?, ¿Para qué se usa? Ejemplifique

3) ¿Qué es un protocolo MODBUS?, ¿Para qué se usa? Ejemplifique

4) ¿Qué es un protocolo HART?, ¿Para qué se usa? Ejemplifique

5) ¿Qué es un protocolo PROFINET?, ¿Para qué se usa? Ejemplifique

6) ¿Qué es un protocolo CANopen?, ¿Para qué se usa? Ejemplifique

7) ¿Qué es un protocolo PROFIBUS-DP/PA?, ¿Para qué se usa? Ejemplifique

1) **¿Qué es un protocolo COAP?, ¿Para qué se usa? Ejemplifique**

CoAP, o "Protocolo de Aplicación Constringente", constituye un pilar en el crecimiento y proliferación de Internet de las Cosas (IoT) en el panorama tecnológico actual. Destaca por su diseño austero y funcional, permitiendo una interrelación fluida y efectiva entre aparatos, aun con limitaciones de recursos o potencia.

Facetas distintivas y relevantes de CoAP

El atractivo principal de CoAP reside en la posibilidad de operar con UDP en lugar de TCP. Esta peculiaridad conlleva una nimiedad en la conexión inicial para intercambiar datos. A pesar de que CoAP posee un diseño más simple, logra cumplir con servicios que comparte con HTTP, como métodos GET, POST, PUT y DELETE.

Necedad y aplicabilidad de CoAP

En el universo de IoT, es frecuente toparse con dispositivos lidiando con limitaciones de elementos y optimización energética. En tales escenarios, un código de enlace ligero y efectivo como CoAP es crucial, particularmente cuando se somete a comparación con códigos convencionales como HTTP que pueden ser excesivamente enrevesados.

Comparativa de CoAP con otros protocolos

Examinando CoAP y MQTT, ambos desarrollados exclusivamente para mejorar el rendimiento de dispositivos IoT con limitaciones, es viable distinguir variaciones significativas en su

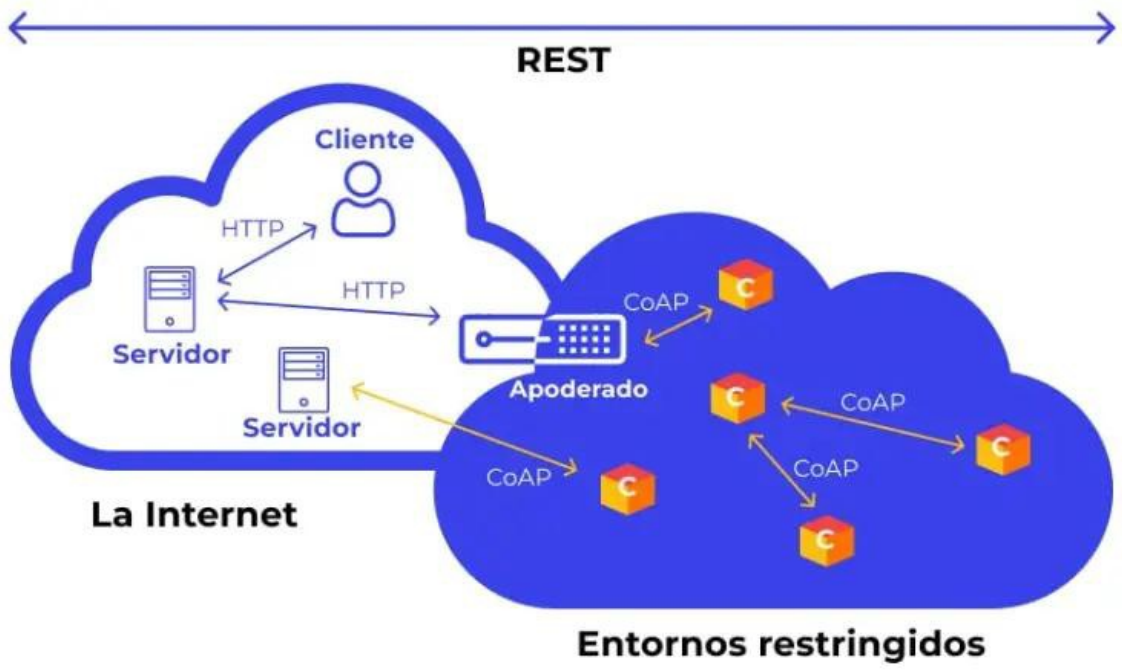
metodología de funcionamiento. A diferencia de MQTT que se rige por un modelo de publicación/suscripción, CoAP sigue un patrón de petición/respuesta parecido a HTTP.

CoAP	MQTT
Opera basándose en petición/respuesta	Funciona con un patrón de publicación/suscripción
Se apoya en UDP	Dependiente de TCP
Facilita la realización de comandos como GET, POST, PUT y DELETE	No incluye estas funciones en su estructura

Consideración relevante

Con su estructura simplista, CoAP se ha venido a posicionar como un protocolo indispensable. Fue ideado con el propósito de apoyar a los artefactos de IoT que enfrentan limitaciones de recursos. Al proveer servicios análogos a los de HTTP, CoAP se postula como una opción notable en la garantía de una comunicación íntegra en dispositivos IoT.

Arquitectura CoAP



El protocolo CoAP, diseñado para propósitos de máxima eficiencia en ambientes de computación de capacidades acotadas, opera de manera simplificada. Su funcionamiento depende de una interacción bidireccional entre dos dispositivos: uno que origina el mensaje (Cliente) y otro que lo recibe y reacciona a él (Servidor).

El esqueleto de CoAP: Clientes y Servidores

La estructura de CoAP descansa en dos elementos clave: Los Clientes y los Servidores CoAP.

- **Cliente CoAP:** Es el nodo que inicia la comunicación. Envía un mensaje al servidor y espera su respuesta. Ejemplos de estos nodos podrían ser smartphones, computadoras o un módulo IoT, siempre y cuando estos tengan la capacidad de intercambiar mensajes a través de una red.
- **Servidor CoAP:** Es el nodo receptor del mensaje original del cliente, y el que actúa en consecuencia. Pueden desempeñar este rol una diversidad de dispositivos que puedan procesar mensajes de la red y responder a ellos, como por ejemplo, un servidor web, un módulo IoT o una base de datos.

Dinámica de la Información en CoAP

La relación Cliente-Servidor en CoAP, opera a través de un intercambio solicitado por el cliente mediante uno de cuatro métodos: GET, POST, PUT y DELETE.

- **GET**: Permite al cliente solicitar cierta información de un recurso específico del servidor.
- **POST**: Se utiliza para enviar información a un recurso del servidor.
- **PUT**: Actualiza la información de un recurso específico del servidor.
- **DELETE**: Permite eliminar el recurso especificado en el servidor.

Protocolos de Comunicación de CoAP

CoAP opta por ejecutar sus operaciones a través del Protocolo de Datagramas de Usuario (UDP) en lugar de TCP (Protocolo de Control de Transmisión). Aunque UDP no garantiza la entrega ordenada de los paquetes de datos, su estructura es más ligera y rápida, lo que resulta óptimo para dispositivos con limitaciones de recursos.

¿Para qué se usa?

CoAP se emplea principalmente en redes IoT donde los dispositivos tienen restricciones de energía, procesamiento y conectividad. Algunos usos incluyen:

•**Automatización del hogar:** Control de luces, termostatos y electrodomésticos inteligentes.

•**Monitoreo industrial:** Sensores que envían datos sobre temperatura, presión o calidad del aire.

•**Sistemas de salud**: Dispositivos médicos que transmiten información a servidores remotos.

•**Ciudades inteligentes:** Gestión de alumbrado público y monitoreo ambiental.

Ejemplo práctico

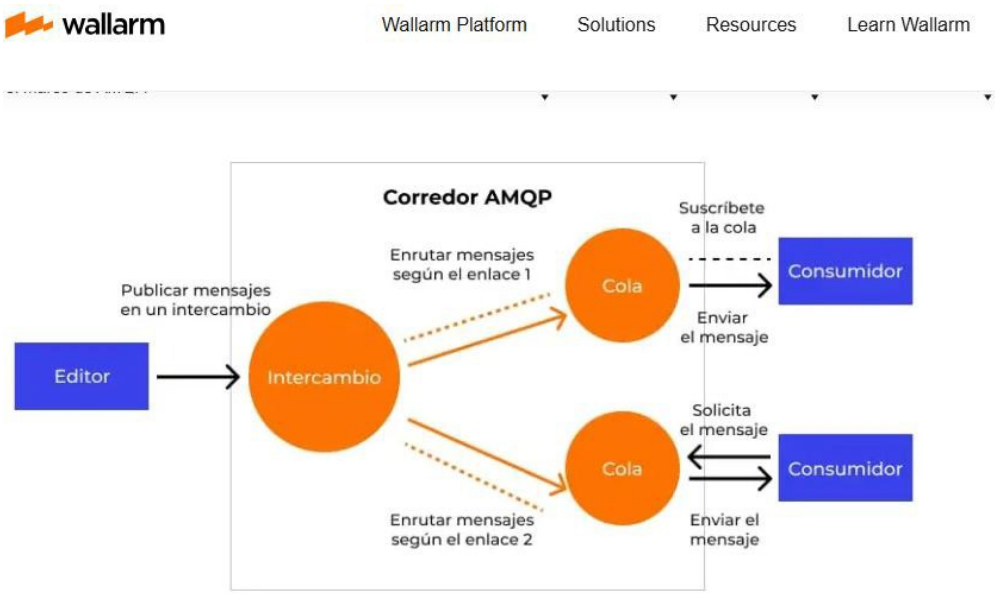
Imagina una red de **sensores inalámbricos** en una casa inteligente. Cada sensor de temperatura usa CoAP para enviar datos a un servidor central. Cuando la temperatura baja demasiado, el servidor activa la calefacción automáticamente. Este tipo de comunicación eficiente permite que los dispositivos IoT funcionen de manera autónoma y optimizada

2) ¿Qué es un protocolo AMQP?, ¿Para qué se usa? Ejemplifique

Protocolo AMQP

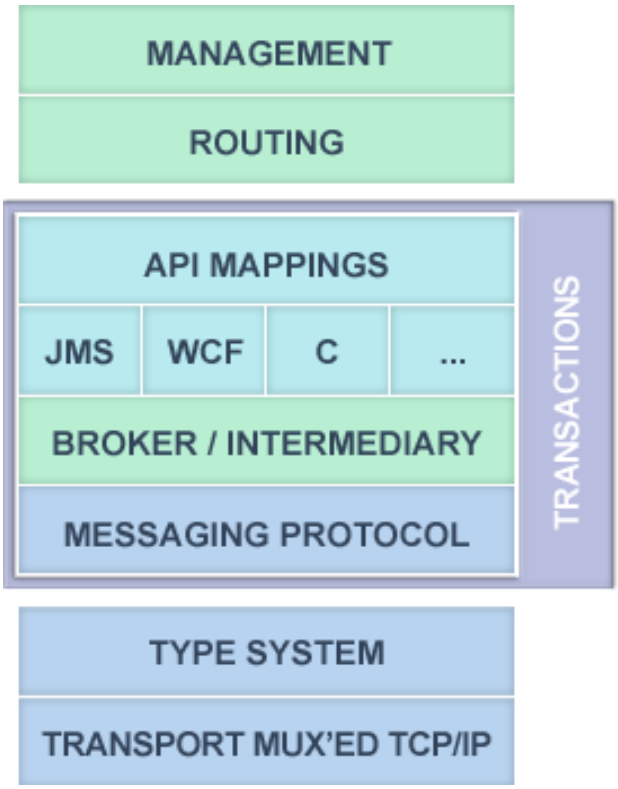
Arquitectura

La versión 1.0 de AMQP proporciona las piezas necesarias para permitir que los clientes y servidores de mensajería interoperen sin problemas, independientemente de su tecnología.



Consiste en un protocolo de cable eficiente que separa el transporte de red de las arquitecturas de intermediario y su gestión. **AMQP versión 1.0** admite diversas arquitecturas de intermediario que pueden utilizarse para recibir, poner en cola, enrutar y entregar mensajes, o para su uso peer**-to-peer.** El alcance de **AMQP 1.0** se especifica en tres

partes principales: el **protocolo de red**, la **representación de los datos del sobre del mensaje** y la semántica básica **de los servicios de intermediario**.



Transporte (TCP/IP)

- Transmisión de bytes ordenada y confiable, y control del flujo de red
- Subdivisión de TCP/IP en *canales*

Protocolo de mensajería y sistema de tipos

- Representación portátil de las propiedades del mensaje
- Transporte de mensajes entre pares de transporte
- Transferencia de responsabilidad por los mensajes entre pares
- Control de flujo de mensajes

Message Broker – la capa MOM

- Intermediación, separación temporal de aplicaciones de mensajería
- Almacenamiento seguro de mensajes
- Recursos transaccionales
- "Direcciones" locales
- Funciones de distribución de mensajes; colas compartidas, temas, etc.
- Colas de respuesta

Mapeos de API

- Representación canónica de conceptos entre API

- Enviar JMS <> WCF <> C <> Python …

Enrutamiento entre corredores: en una próxima incorporación

- Separación política de las aplicaciones de mensajería
- Reenvío de mensajes
- Traducción de dirección hacia adelante/hacia atrás
- Propagación de suscripciones

Gestión –en una próxima incorporación

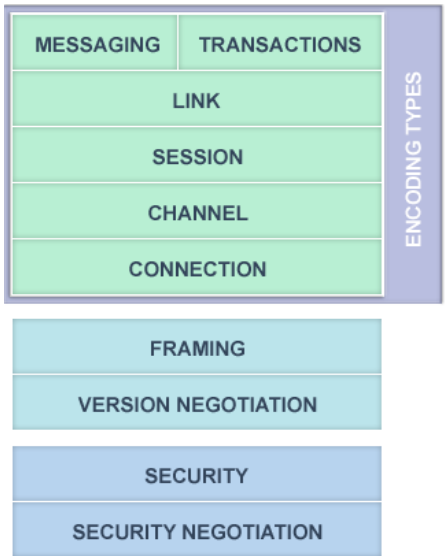
- Gestión estandarizada de cualquier bróker AMQP
- Envía mensajes de gestión de AMQP al bróker para efectuar cambios
- Aproveche todas las funciones de enrutamiento y seguridad de Broker and Routing

El protocolo de red AMQP

- El protocolo de red AMQP define:
- Un protocolo peer to peer; aunque normalmente en AMQP un peer desempeña el papel de una aplicación cliente y el otro peer desempeña el papel de un servicio de enrutamiento y entrega de mensajes confiable, o intermediario.
- Cómo conectarse a servicios, incluido un método para conmutar por error las conexiones a servicios alternativos
- Un mecanismo para permitir que los pares descubran las capacidades de los demás
- Mecanismos de seguridad integrales, incluidos SSL y Kerberos para una confidencialidad perfecta de extremo a extremo
- Cómo multiplexar una conexión TCP/IP para que múltiples conversaciones puedan tener lugar en una sola conexión. Esto simplifica enormemente la gestión del firewall.
- Cómo direccionar una fuente de mensajes con el par de la red y especificar qué mensajes son de interés
- El ciclo de vida de un mensaje: desde la obtención, el procesamiento y la confirmación. AMQP deja muy claro cuándo se transfiere la responsabilidad de un mensaje de un par a otro, lo que mejora la confiabilidad.
- Cómo mejorar el rendimiento, si se desea, recuperando con antelación mensajes en la red, listos para que el cliente los procese sin demora
- Una forma de procesar lotes de mensajes dentro de una transacción
- Un mecanismo para permitir una transferencia completa de mensajes desde el inicio hasta el cierre de sesión en un paquete de red para aplicaciones livianas
- Control de flujo muy capaz, que permite a los consumidores de mensajes reducir la velocidad de los productores a un nivel manejable, y que permite que distintas cargas de trabajo procedan en paralelo a distintas velocidades en una conexión.

- Mecanismos para reanudar las transferencias de mensajes cuando se pierden y se restablecen las conexiones; por ejemplo, en caso de conmutación por error del servicio o conectividad intermitente.

Este protocolo de red de bajo nivel es completo y eficaz, pero normalmente es invisible para los usuarios de software de mensajería. Al igual que el correo electrónico, los usuarios enviarán mensajes a target@example.com sin conocer los detalles de cómo los intermediarios procesan esa solicitud. Dicho esto, es posible que los especialistas en redes utilicen este protocolo para otros fines, pero se anticipa que el mayor uso del protocolo será conectar clientes de mensajería con intermediarios de mensajería; los intermediarios que alojan colas y temas y se encargan del almacenamiento seguro, el enrutamiento y la entrega.



Representación del mensaje

AMQP proporciona portabilidad entre sistemas. Por lo tanto, necesita una forma de representar los datos empresariales necesarios para enrutar y entregar el mensaje de forma portátil, accesible desde diversos lenguajes de programación. El sistema de tipos AMQP 1.0 y las funciones de codificación de mensajes proporcionan una codificación portátil para satisfacer esta necesidad. Normalmente, esta codificación solo se utiliza para añadir propiedades de enrutamiento al "envoltorio" del mensaje; el contenido dentro del envoltorio se transporta intacto. Es probable que las aplicaciones utilicen codificaciones XML, JSON o similares en el contenido de sus mensajes. Opcionalmente, una aplicación también podría optar por usar la codificación AMQP para el contenido del mensaje, pero esto es totalmente opcional.

Servicios de corretaje

La ventaja de usar intermediarios de mensajes reside en que un intermediario confiable, diseñado específicamente para este fin, gestiona las complejidades de la cola, el enrutamiento y la entrega de mensajes. Ese intermediario es el intermediario de mensajes.

AMQP define el conjunto mínimo de requisitos que se esperan de un agente de mensajes y, donde se utilizan frecuentemente facilidades más avanzadas, especifica cómo se deben

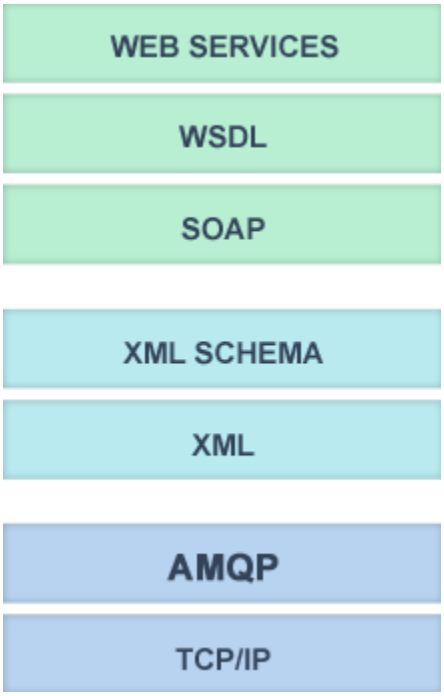
exponer esas facilidades a los clientes. El objetivo de AMQP es permitir que las aplicaciones envíen mensajes a través de los servicios de intermediario; estos conceptos básicos son necesarios, pero no constituyen el objetivo en sí mismos. Al enviar un mensaje por mensajería, ¡no conduces el camión de reparto!

AMQP Diferenciado

AMQP es diferente de otros estándares de middleware porque es:

- **INTEROPERABLE**. Todos los clientes AMQP interoperan con todos los servidores AMQP. Diversos lenguajes de programación se comunican fácilmente. Los agentes de mensajes heredados pueden modernizarse para eliminar protocolos propietarios de su red. La mensajería puede habilitarse como servicio en la nube. Una solución sencilla para la mensajería empresarial y la integración de middleware, diseñada para satisfacer necesidades reales.
- **CONFIABLE** . Capaz de eliminar las brechas de comunicación y las ralentizaciones entre diferentes plataformas, sistemas críticos y componentes de aplicaciones, tanto dentro de la empresa como con sistemas y organizaciones externas.
- **UNIFICADO** . Proporciona un conjunto básico de patrones de mensajería a través de un único protocolo manejable.
- **COMPLETO**. JMS proporciona una API para protocolos de aplicación y se basa en Java. AMQP proporciona transporte a nivel de cable para las aplicaciones que utilizan dicha API. AMQP es ampliamente aplicable y se puede aprovechar en cualquier lenguaje, e identifica las semánticas de almacenamiento y reenvío y publicación y suscripción en una sola especificación.
- **ABIERTO**. Independiente del proveedor y la plataforma, y creado por usuarios y proveedores de tecnología que trabajan en colaboración.
- **SEGURO** . Una solución segura al problema de transportar mensajes valiosos entre organizaciones, plataformas tecnológicas y un entorno virtual de computación en la nube.

Utilice AMQP para implementar una arquitectura orientada a servicios



AMQP completa la imagen de una pila SOA abierta y pragmática al utilizar una infraestructura TCP/IP para el transporte que proporciona conectividad ubicua. El protocolo actúa como intermediario confiable para la entrega asincrónica de mensajes con características de servicio específicas. XML proporciona una codificación de datos uniforme, mientras que el esquema XML proporciona la especificación de metadatos para una representación adecuada. SOAP proporciona los patrones esperados de intercambio de mensajes, mientras que los estándares de servicios web permiten la representación de contratos comerciales.

Protocolo avanzado de cola de mensajes: AMQP

DICIEMBRE 13, 2024 **13 MINS READ**

AMQP significado

El Método Innovador de Distribución de Mensajes (AMQP, por sus siglas en inglés) proporciona un marco para la transmisión segura y fluida de datos entre un universo de sistemas mediante la utilización de una estructura de red. Este enfoque crea un flujo

controlado de datos en una red, respaldando la interacción asíncrona entre los sistemas, eliminando la demanda de operatividad simultánea de cada sistema.

Descripción del funcionamiento de AMQP

AMQP adopta un enfoque comprensivo para la diseminación de datos. Los datos son transportados mediante un 'nexus', un componente que dirige los datos a múltiples canales. De esta forma, las entidades que reciben acceden a los datos desde estos canales para su análisis posterior. Este diseño facilita la interacción efectiva entre los sistemas, permitiendo reservar los datos en los canales hasta que estén listos para ser procesados.

Importancia del AMQP

AMQP es vital por diversas razones. En primer lugar, es de acceso libre, lo que indica que cualquier usuario puede implementarlo y optimizarlo, conduciendo a su adopción extensiva en varios ámbitos y circunstancias.

Además, AMQP respalda la interacción fluida entre sistemas. Los datos se pueden difundir y recuperar de forma asíncrona, es decir, los sistemas no tienen que estar operativos de manera concurrente. Esta característica es especialmente útil cuando la disponibilidad del sistema puede ser volátil o inconstante.

Por último, AMQP puede ser personalizado y adaptado a diversas aplicaciones. Su configuración es adaptable a diferentes esquemas de difusión de datos, desde modelos de emisión/adquisición, donde los datos se pueden enviar a varias entidades receptoras, hasta canales de trabajo, donde los datos se direccionan a un solo receptor.

Comparación del AMQP con otros protocolos de distribución de datos

Existen otros protocolos de distribución de datos como lo son MQTT y STOMP. Sin embargo, AMQP tiene ventajas considerables. AMQP ofrece más flexibilidad debido a los diversos esquemas de diseminación de datos que admite. Además, asegura una seguridad estable en la recepción de datos, que pueden ser reservados en los canales hasta que estén listos para su análisis. Por último, siendo de acceso libre, AMQP tiene alta adopción y es respaldado por un extenso grupo de desarrolladores.

Protocolo	Flexibilidad	Seguridad en la Recepción	Acceso Libre
AMQP	Alta	Excelente	Sí
MQTT	Moderada	Aceptable	Sí
STOMP	Limitada	Insuficiente	Sí

En conclusión, AMQP es un protocolo de distribución de datos que proporciona un medio seguro, eficiente y adaptable para la interacción entre sistemas. Su adopción en una amplia gama de entornos y la disponibilidad del código abierto, lo posicionan como una opción insuperable para un sinfín de aplicaciones y contextos.

Historia de AMQP

La historia de AMQP es una historia de colaboración y desarrollo continuo. En 2004, JPMorgan Chase, una de las instituciones financieras más grandes del mundo, se dio cuenta de que necesitaba una solución de mensajería más eficiente y confiable. En lugar de optar por una solución propietaria, decidió colaborar con iMatix Corporation, una empresa de software, para desarrollar un protocolo de mensajería abierto. Así nació AMQP.

El inicio de AMQP

En sus primeras etapas, AMQP fue diseñado para ser un protocolo de mensajería que pudiera manejar altos volúmenes de mensajes de manera eficiente y confiable. El objetivo era crear un protocolo que fuera fácil de implementar y que pudiera ser utilizado por cualquier empresa, independientemente de su tamaño o industria. En 2006, se lanzó la primera versión de AMQP. Esta versión inicial ya mostraba algunas de las características clave de AMQP, como la entrega garantizada de mensajes y la capacidad de manejar diferentes tipos de mensajes.

Desarrollo y evolución de AMQP

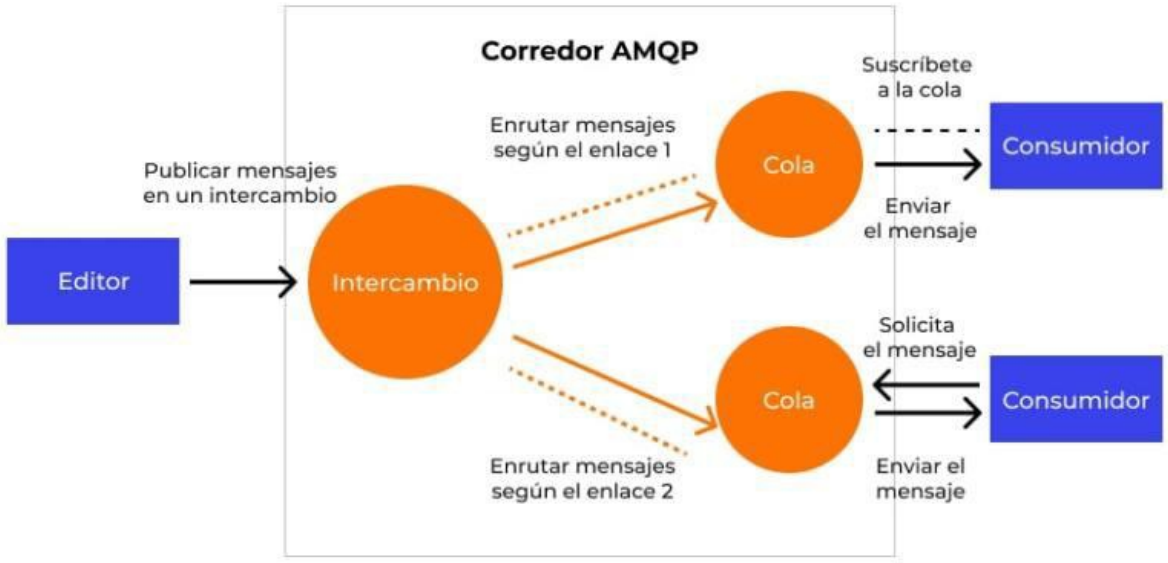
Desde su lanzamiento inicial, AMQP ha evolucionado significativamente. En 2008, se lanzó la versión 0.9.1 de AMQP, que introdujo varias mejoras y nuevas características. Esta versión también marcó el inicio de la adopción más amplia de AMQP, con empresas como Microsoft y Red Hat comenzando a utilizar el protocolo. En 2011, se lanzó la versión 1.0 de AMQP. Esta versión representó un cambio significativo en la arquitectura de AMQP, con un enfoque en la simplicidad y la flexibilidad. La versión 1.0 también introdujo el concepto de "intercambios", que permiten a los mensajes ser enviados a múltiples destinatarios.

AMQP hoy

Hoy en día, AMQP es utilizado por una amplia gama de empresas e instituciones, desde gigantes tecnológicos como Microsoft y Google hasta instituciones financieras y gobiernos. El protocolo ha demostrado ser extremadamente eficaz en la entrega de mensajes de manera eficiente y confiable, y su diseño abierto ha permitido a las empresas adaptarlo a sus necesidades específicas. En resumen, la historia de AMQP es una historia de innovación y colaboración. Desde su creación por JPMorgan Chase e iMatix Corporation, AMQP ha evolucionado para convertirse en uno de los protocolos de mensajería más utilizados en el mundo. Y con su enfoque en la eficiencia, la confiabilidad y la flexibilidad, es probable que AMQP continúe siendo una opción popular para la mensajería en el futuro.

Terminología AMQP

Antes de hundirnos en las profundidades técnicas de AMQP, es innegablemente importante entender una serie de términos clave que se emplean en este protocolo. Adelante, haremos un recorrido por algunos de las terminologías más usuales y su relevancia en el marco de AMQP.



Comunicado

En el ámbito de AMQP, un comunicado es la porción de información transportada de un punto a otro. Los comunicados se componen de dos segmentos esenciales: el preámbulo y el contenido. El preámbulo incorpora metadatos del comunicado, tales como su clase y precedencia, mientras que el contenido engloba los datos genuinos que están siendo transportados.

Generador

Un generador en este escenario es una aplicación que origina y despacha comunicados a una fila. Dicho de otro modo, es el origen de los comunicados en una infraestructura de AMQP.

Receptor

Un receptor en este sentido es una aplicación que recoge y trata los comunicados de una fila. En lenguaje sencillo, es el destino de los comunicados en una arquitectura AMQP.

Fila

La fila es una estructura de datos en la que se almacenan los comunicados a la espera de ser tratados. En el marco de AMQP, las filas son el lugar donde los generadores depositan los comunicados y donde los receptores los recogen.

Cambio

Un cambio en este orden es un distribuidor de comunicados en AMQP. Los generadores remiten comunicados a los cambios, y estos resuelven a qué filas deben ser despachados los comunicados de acuerdo a normas de distribución.

Conexión

Una conexión representa el vínculo entre una fila y un cambio. Los comunicados son transmitidos desde el cambio a la fila a través de la conexión.

Vía

La vía es un enlace virtual que se crea encima de una conexión de red. Las vías se emplean para suministrar y recibir comunicados entre generadores y receptores.

Operador

Un operador es un servidor que aplica el protocolo AMQP y ofrece la funcionalidad para suministrar y recibir comunicados.

Camino de distribución

Un camino de distribución es una llave que se utiliza para decidir a qué filas deberían ser enviados los comunicados. Los cambios usan el camino de distribución para concluir la manera de dispersar los comunicados a las filas.

Verificación

La verificación es un mecanismo que faculta a los receptores para notificar al operador de que han adquirido y procesado un comunicado con éxito. Estas terminologías forman la base de la operativa de AMQP. En los próximos capítulos, indagaremos en cómo estos componentes interactúan entre sí para ofrecer una solución de mensajería sólida y fiable.

Componentes de AMQP

El Protocolo de Alto Rendimiento para la Gestión Avanzada de Mensajes en Cola (AMQP) se estratifica en diversas secciones fundamentales operando juntas para brindar un sistema de correspondencia confortable y versátil. Estas constituciones contemplan el eje primordial de la manipulación de mensajes, los emisores y suscriptores integrados en la plataforma, las vías de trasiego, las cajas de mensajes, las estaciones de tránsito y las pautas de vinculación.

Eje de Coordinación de Mensajes

Este eje esencial del AMQP funge como la estructura elemental del sistema. Encargado de la acogida, alojamiento y distribución de los mensajes a diferentes emisores y suscriptores integrados en la plataforma, también se inviste de la dirigencia y gestión de las cajas de mensajes y las estaciones de tránsito.

Emisores y Suscriptores Integrados en la Plataforma

Emisores y suscriptores están integrados en la plataforma y de hecho, son la misma plataforma, cuya responsabilidad yace en la creación y utilización de los mensajes. Los emisores brindan los mensajes al eje, mientras los suscriptores (usuarios) los captan. Tanto emisores como suscriptores se conectan con el eje de coordinación de mensajes a través de un enlace en la red.

Vías de Trasiego

Una vía de trasiego es una reunión virtual activa en un enlace de red entre el usuario y el eje de coordinación de mensajes. Estas vías posibilitan la concurrencia en la entrega y recepción de mensajes, evitando conexiones múltiples en la red. Cada vía tiene una designación única para las cajas y las estaciones de tránsito, permitiendo a los usuarios operar con múltiples cajas y estaciones de tránsito de manera simultánea.

Cajas de Mensajes

Las cajas de mensajes se utilizan para la conservación de los mensajes hasta su empleo. Cada mensaje es posicionado en una caja específica y los usuarios pueden subscribirse a un número variado de cajas para captar mensajes. Las cajas pueden ser permanentes (donde los mensajes persisten incluso si el sistema de mensajes se reinicia) o temporales (donde los mensajes se desvanecen al reiniciar el sistema de mensajes).

Estaciones de Tránsito

Las estaciones de tránsito son los puntos de arribo para los mensajes en el sistema AMQP. Los emisores brindan los mensajes a las estaciones de tránsito y éstas a su vez los dirigen a uno o más cajas basándose en las pautas de vinculación. Existen múltiples versiones de estaciones de tránsito, como las directas, temáticas, de salida y de encabezados, cada una con su propio conjunto de pautas de vinculación.

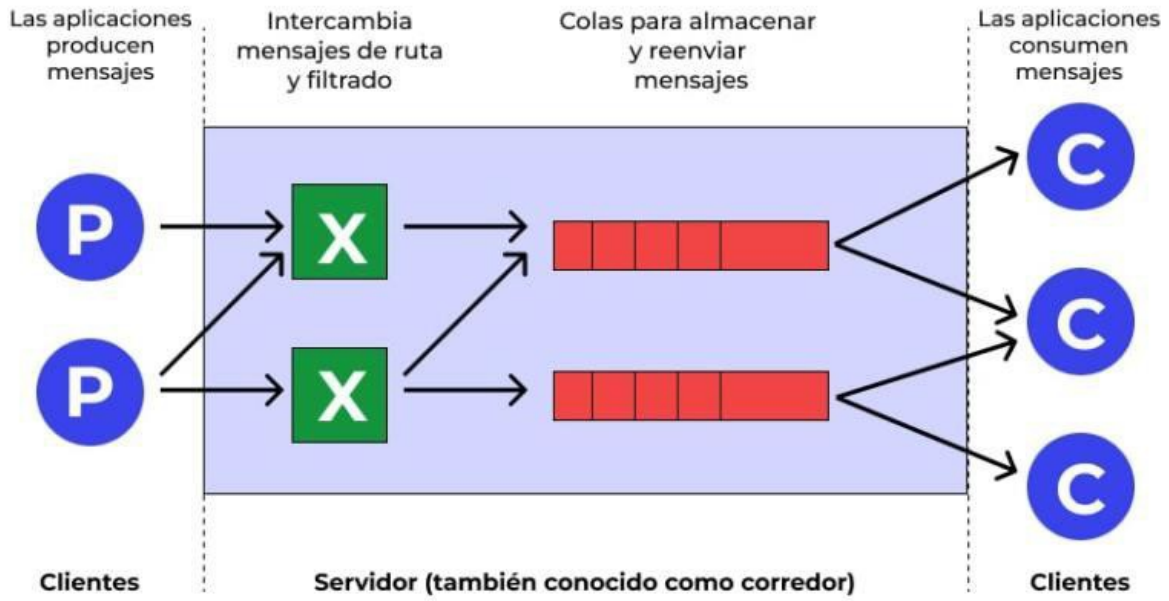
Pautas de Vinculación

Las pautas de vinculación son reglas que esbozan cómo los mensajes son dirigidos desde las estaciones de tránsito hasta las cajas. Una pauta de vinculación puede ser tan simple como una coincidencia precisa de una clave de vinculación o tan compleja como una expresión que contenga comodines y otros patrones. Para concluir, cada segmento del AMQP colabora en conjunto para suministrar un mecanismo de mensajería eficiente y adaptable. Los emisores brindan mensajes a las estaciones de tránsito, las cuales los dirigen a las cajas

adecuadas en base a las pautas de vinculación. Los usuarios después recogen los mensajes de las cajas a través de las vías de trasiego.

¿Cómo funcionan los intercambios AMQP?

Considerar AMQP y su funcionamiento de intercambios puede ser determinante para comprender la manipulación de la comunicación a través de este protocolo. Los intercambios actúan como mediadores entre los emisores de mensajes y las colas apropiadas, utilizando patrones y reglamentaciones específicas.



Variantes de Intercambios en el AMQP

Principalmente, se subdividen en cuatro categorías los intercambios del AMQP, cada uno aplicando su propia serie de normas para el direccionamiento de mensajes:

1) **Directo**: Los mensajes son redirigidos a las colas que corresponden exactamente con la clave del mensaje en este tipo de intercambio.
2) **Tema**: Esto permite la distribución de mensajes basada en patrones de correspondencia en los que las claves pueden ser especificadas con caracteres de reemplazo para permitir variadas correspondencias.
3) **Fanout**: Ignorando las claves de direccionamiento, este tipo de intercambio se encarga de distribuir el mensaje a todas las colas vinculadas a él.
4) **Cabeceras**: Este intercambio se encarga de direccionar los mensajes basándose en los atributos del encabezado del mensaje en vez de la clave de direccionamiento.

Dirección de Mensajes en los Intercambios AMQP En los intercambios de AMQP se procede la dirección de mensajes de la siguiente manera:

- **Generación del mensaje**: El remitente crea un mensaje específico y lo envía a un intercambio, el mensaje contiene una clave de dirección y posiblemente algunos encabezados.
- **El intercambio recibe el mensaje**: El intercambio incorpora los mensajes y los categoriza en base a su clave de direccionamiento y/o encabezados.
- **Distribución del Mensaje**: De acuerdo con las normas de direccionamiento y el tipo de intercambio, el mensaje es enviado a una o varias colas.
- **Consumo del mensaje**: Los receptores ligados a las colas adquieren y procesan el mensaje.

Código de Ejemplo para trabajar con el intercambio de AMQP A continuación, se muestra un ejemplo práctico del envío de un mensaje a través de un intercambio y la recepción de este en una cola por parte del receptor.

```
# Remitente
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='journal_direct', exchange_type='direct')

importance = 'error'
message = 'This is an error message.'

channel.basic_publish(exchange='journal_direct', routing_key=importance,
body=message)
print(" [x] Sent %r:%r" % (importance, message))

connection.close()

# Receptor
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='journal_direct', exchange_type='direct')

result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue

importance = 'error'
channel.queue_bind(exchange='journal_direct', queue=queue_name,
routing_key=importance)

print(' [*] Waiting for logs. To exit press CTRL+C')

def callback(ch, method, properties, body):
    print(" [x] %r:%r" % (method.routing_key, body))

channel.basic_consume(queue=queue_name, on_message_callback=callback,
auto_ack=True)

channel.start_consuming()
```

En este código, se observa que el remitente envía un mensaje con una clave de direccionamiento 'error' a un intercambio de tipo 'directo'. El receptor sintoniza la misma cola y recibe todos los mensajes que tengan la misma clave de direccionamiento 'error'. Los intercambios en AMQP juegan un papel esencial para la distribución adaptable de mensajes entre remitentes y receptores. Dependiendo del tipo de intercambio y sus normas de
direccionamiento, los mensajes se pueden entregar a una o más colas, lo que facilita una excelente flexibilidad y eficiencia en la gestión de la comunicación.

Desarrollo de API con AMQP

El mérito de estructurar APIs utilizando el Advanced Message Queuing Protocol (AMQP) consiste en la habilidad de configurar canales de comunicación entre diferentes sistemas

de software. Mediante AMQP, se establece un roadmap consistente para intercambiar data entre distintos programas, optimizando la eficiencia y protección de dichos intercambios de información.

Beneficios de utilizar AMQP para construir APIs

Implementar AMQP en la creación de APIs desbloquea diversos beneficios. Primordialmente, AMQP es un protocolo estandarizado, facilitando la integración con múltiples plataformas y lenguajes de programación, lo que capacita para diseñar APIs que son compatibles con diversas aplicaciones, sin importar la plataforma o lenguaje de programación en uso. Además, AMQP aporta funcionalidades que simplifican la estructuración de APIs confiables y seguras. Tal como, AMQP respalda la entrega confirmada de mensajes, asegurando que la data llegará a su destino, aun en circunstancias de fallos de red o del sistema.

Cómo crear una API empleando AMQP

Para construir una API con AMQP se requieren algunos pasos. En primer lugar, debes estructurar los mensajes que la API va a transferir. La complejidad y el tipo de data que decidas transmitir puede ser adaptable a tus necesidades. Posteriormente, es requerido configurar las colas e intercambios que la API empleará. Las colas son los depósitos donde los mensajes esperan ser procesados, mientras que los intercambios actúan como los portales de acceso para los mensajes. Finalmente, deberás discurrir el código que maneja el envío y recepción de mensajes mediante la API. Dicho código puede ser redactado en cualquier idioma de programación que soporte AMQP, y deberá aprovechar las herramientas proporcionadas por AMQP para el intercambio de mensajes.

Ejemplo de código

Aquí se muestra un exemplum de cómo uno puede desarrollar una API empleando AMQP con Python y la librería pika:

```
# Iniciar conexión con el servidor AMQP
conexion = pika.BlockingConnection(pika.ConnectionParameters('localhost'))

# Configuración del canal comunicativo
canal = conexion.channel()

# Declaración de la cola
canal.queue_declare(queue='hola')

# Transmisión de un mensaje
canal.basic_publish(exchange='',
                    routing_key='hola',
                    body='¡Hola Mundo!')
print(" [x] Enviado '¡Hola Mundo!'")

# Finalización de la conexión
conexion.close()
```

Este programa inicia una conexión con un servidor AMQP, instancia una cola, transfiere un mensaje a la cola, y luego termina la conexión.

Conclusión

La creación de APIs empleando AMQP es un método que puede asistirte en la creación de APIs de alta confiabilidad y seguridad. Con su amplio soporte para múltiples lenguajes de programación y plataformas, y su capacidad de entregar mensajes de forma segura, AMQP se consolida como una excelente elección para cualquier desarrollador de APIs.

La última palabra

El innovador sistema de comunicación que se apoya en el Proceso Superior de Manejo de Colas de Mensajes (AMQP) se posiciona como un recurso imprescindible para organizaciones contemporáneas. Su naturaleza de código abierto y estandarizada, facilita la fusión de variados sistemas y plataformas, dando pie a una colaboración más fluida.

Razones para decantarse por AMQP

La verdadera esencia de AMQP yace en su naturaleza de código abierto y estandarizada, facilitando la fusión sin interrupciones de variados sistemas y programas, dejando atrás cualquier preocupación por discrepancias. Su capacidad para adaptarse a una multitud de soluciones de transmisión de mensajes de diversos proveedores es su notable atributo.

Además, AMQP ofrece una gama de funciones extra que potencia la creación de sistemas de transmisión de mensajes duraderos y escalables. Gracias a su capacidad para ser compatible con varios modelos de intercambio de mensajes, como la transmisión directa, la transmisión temática y la difusión fanout, otorga a las organizaciones la posibilidad de personalizar sus soluciones de comunicación acorde a sus requerimientos particulares.

Uno de los puntos clave de AMQP es la garantía de entrega de mensajes. Esta certificación de envío y persistencia de mensajes facilita la recuperación en episodios inesperados de fallas de sistema o red, convirtiéndose en un elemento crucial para aplicaciones comerciales de relevancia.

Visiones Futuras para AMQP

En el actual entorno digital, la necesidad de implementar soluciones de transmisión de mensajes resilientes y expansibles es indiscutible. AMQP posee un papel principal en este desafío.

AMQP, respaldado por su diseño universal y normalizado, está preparado para ayudar en la consolidación de un marco efectivo para la integración y colaboración entre sistemas y plataformas dispares. Sus funciones avanzadas y la garantía en la entrega de mensajes le otorgan un lugar prominente como opción ideal para robustas y en crecimiento soluciones de transmisión de mensajes.

En resumen, AMQP es una extraordinaria y flexible herramienta de transmisión de mensajes que ayuda a las organizaciones a superar las barreras de integración y colaboración en el intrincado entorno empresarial actual. Su creciente adopción reafirma que seguirá siendo un elemento indispensable en el futuro de la transmisión de mensajes en los negocios.

Conclusión

El Proceso Superior de Manejo de Colas de Mensajes (AMQP) es un sistema de transmisión de mensajes resistente y versátil, fundamental para las corporaciones actuales. Gracias a su diseño universal y estandarizado, AMQP facilita la colaboración eficaz entre sistemas y plataformas variadas. Su singularidad y la garantía de la entrega de mensajes lo colocan como primera opción para la implementación de fuertes soluciones de transmisión de mensajes en expansión. En resumidas cuentas, si una empresa busca superar los obstáculos de integración y colaboración, AMQP sin duda debe ser considerado.

FAQ

Voy a abordar distintas cuestiones relativas al Protocolo Mejorado para la Administración de Filas de Mensajes, mejor conocido por sus siglas en inglés, AMQP.

¿Por qué es crucial AMQP y cómo se le emplea?

El AMQP tiene un rol fundamental como protocolo en la capa de aplicación, ya que facilita la transmisión y recepción de mensajes entre aplicaciones a través de una red. Su importancia es particularmente evidente en la mensajería corporativa, en la cual garantiza una transmisión efectiva de mensajes entre diferentes programas y sistemas.

¿De qué manera opera AMQP?

La operación de AMQP se basa en la gestión de intercambios y filas. Los mensajes se envían a un intercambio que, en base a las reglas de enrutamiento predefinidas, categoriza y dirige los mensajes hacia una o varias filas. Posteriormente, los destinatarios extraen los mensajes de las filas para su procesamiento.

¿Cuáles son los elementos esenciales de AMQP?

Los principales componentes de un sistema AMQP son el emisor, el intercambio, la fila y el receptor. El emisor remite los mensajes al intercambio, que los clasifica y reparte en las respectivas filas. Desde ahí, los receptores acceden y procesan los mensajes.

¿Cómo AMQP garantiza una entrega de mensajes segura?

AMQP utiliza varias estrategias para asegurar la entrega segura de los mensajes. Entre estas se incluyen las confirmaciones de envío, las confirmaciones de recepción y las transacciones.

¿Cómo se desarrolla una API con AMQP?

Cuando se desarrolla una API empleando AMQP, se utiliza este protocolo para facilitar la transmisión y recepción de mensajes entre aplicaciones a través de dicha interfaz de programación. Producto de esto es una interacción eficiente y segura entre las aplicaciones.

¿En qué se diferencia AMQP de otros protocolos de mensajería?

AMQP se distingue de otros protocolos de mensajería por diversas características. Como protocolo en la capa de aplicación, ofrece funcionalidades avanzadas inalcanzables en protocolos de capas inferiores. Adicionalmente, AMQP es de código libre, pudiendo ser implementado y usado sin limitaciones. Por último, es preferido en la mensajería de negocios por su habilidad para asegurar una comunicación segura de mensajes.

¿Cuáles son los pros y contras de usar AMQP?

El uso de AMQP trae consigo varias ventajas, como su capacidad para garantizar un intercambio de mensajes seguro, su condición de código libre y su uso extendido en la mensajería corporativa. No obstante, su implementación y uso pueden resultar más complejos en comparación con otros protocolos de mensajería. Además, a pesar de ser de código libre, no todos los sistemas de mensajería son compatibles con este.

¿Dónde puedo obtener más información acerca de AMQP?

Para indagar más acerca del AMQP, puedes visitar su página oficial en http://www.amqp.org. También existen diversos libros y tutoriales en línea que proporcionan una completa comprensión acerca del protocolo AMQP y su aplicación.

3) ¿Qué es un protocolo MODBUS?, ¿Para qué se usa? Ejemplifique

PROTOCOLO MODBUS

Descripción detallada: MODBUS

es un protocolo de comunicación de capa de aplicación que se utiliza principalmente en sistemas de automatización industrial. Fue desarrollado originalmente por Modicon para facilitar la comunicación entre controladores lógicos programables (PLC) y dispositivos periféricos, como sensores y actuadores. Existen varias variantes de **MODBUS: MODBUS RTU** (para comunicación serial), **MODBUS ASCII, y MODBUS TCP** (que usa **TCP/IP** para la comunicación sobre Ethernet). **MODBUS** es ampliamente utilizado en aplicaciones donde se requieren tiempos de respuesta rápidos y comunicaciones de bajo costo. **Modbus** es un protocolo de comunicación utilizado en la automatización industrial para permitir la comunicación entre dispositivos como los controladores lógicos programables (PLC). Opera en una configuración maestro-esclavo, donde el maestro consulta y controla múltiples dispositivos esclavos. El puerto 502 se utiliza habitualmente para Modbus TCP/IP, lo que lo convierte en un objetivo clave para las pruebas de penetración y la detección de vulnerabilidades de seguridad.

Características clave: Simplicidad:

La estructura de los mensajes es muy sencilla, lo que facilita su implementación en diversos dispositivos.

Maestro-esclavo:

El maestro solicita datos a los esclavos y estos responden con la información solicitada.

Versatilidad:

Compatible con diversas formas de comunicación, como serial (RS-232, RS-485) y Ethernet (MODBUS TCP).

Uso en SCADA:

Es utilizado por sistemas SCADA para la supervisión y control de dispositivos en tiempo real.

Flujo de trabajo

Buscar dispositivos: utilizar Nmap para encontrar dispositivos que escuchen en el **puerto 502.** Tomar huellas digitales del sistema: utilizar herramientas como el módulo **Modbusdetect de Metasploi**t para conocer la versión y las capacidades de Modbus. Comprobar características de seguridad: pruebe si el sistema requiere autenticación, ya que muchas implementaciones de Modbus no lo hacen, lo que permite que cualquiera pueda emitir comandos.

Códigos de función de prueba:

explorar los códigos de función **Modbus** (como 0x01 para bobinas de lectura) para ver si pueden leer o escribir datos confidenciales. Buscar exploits: verificar si hay desbordamientos de búfer enviando solicitudes de gran tamaño y pruebe ataques de intermediario, dado que el tráfico Modbus a menudo no está cifrado.

Falta de cifrado

Un punto interesante es que la comunicación Modbus normalmente no está cifrada, lo que significa que un atacante puede interceptar y modificar mensajes fácilmente, aumentando el riesgo en entornos industriales.

Buscando dispositivos

Comenzar usando Nmap para buscar dispositivos Modbus en el puerto 502. Ejecute este comando:

nmap -p 502 --script modbus-discover <RANGO_IP>p 502 --script modbus-discover <RANGO_IP> Esto ayudará a identificar dispositivos y recopilar información inicial.

Reconocimiento inicial

Utilizar Metasploit para un reconocimiento más profundo. Primero, detecte el servicio Modbus: **msf > usar auxiliar/escáner/scada/modbusdetect**

msf auxiliar(modbusdetect) > establecer RHOSTS <DIRECCIÓN IP>establecer RHOSTS <DIRECCIÓN

IP>

msf auxiliar(modbusdetect) > ejecutar

Luego, se enumera los ID de las unidades:

msf > usar auxiliar/escáner/scada/modbus_findunitid

msf auxiliar(modbus_findunitid) > establecer RHOSTS <DIRECCIÓN IP>establecer RHOSTS <DIRECCIÓN_IP>

msf auxiliar(modbus_findunitid) > ejecutar

Interactuando con dispositivos

Instala y usa Smod para una interacción detallada. Clónalo y ejecútalo:

clon git https://github.com/enddo/smod

cd smod

python smod.py

Conectarse al dispositivo:

conectar -ip <DIRECCIÓN IP> -puerto 502< DIRECCIÓN IP > -puerto 502

Enumerar códigos de función y leer/escribir registros, por ejemplo:

enum_ func lectura_registro_de_retención - addr 0 - cuenta 1

escritura_registro_de_retención - addr 0 - valor 100

Explotación de vulnerabilidades

Comprobar si hay desbordamientos de búfer enviando solicitudes de gran tamaño en Smod. Para ataques de intermediario, capture el tráfico con Wireshark:

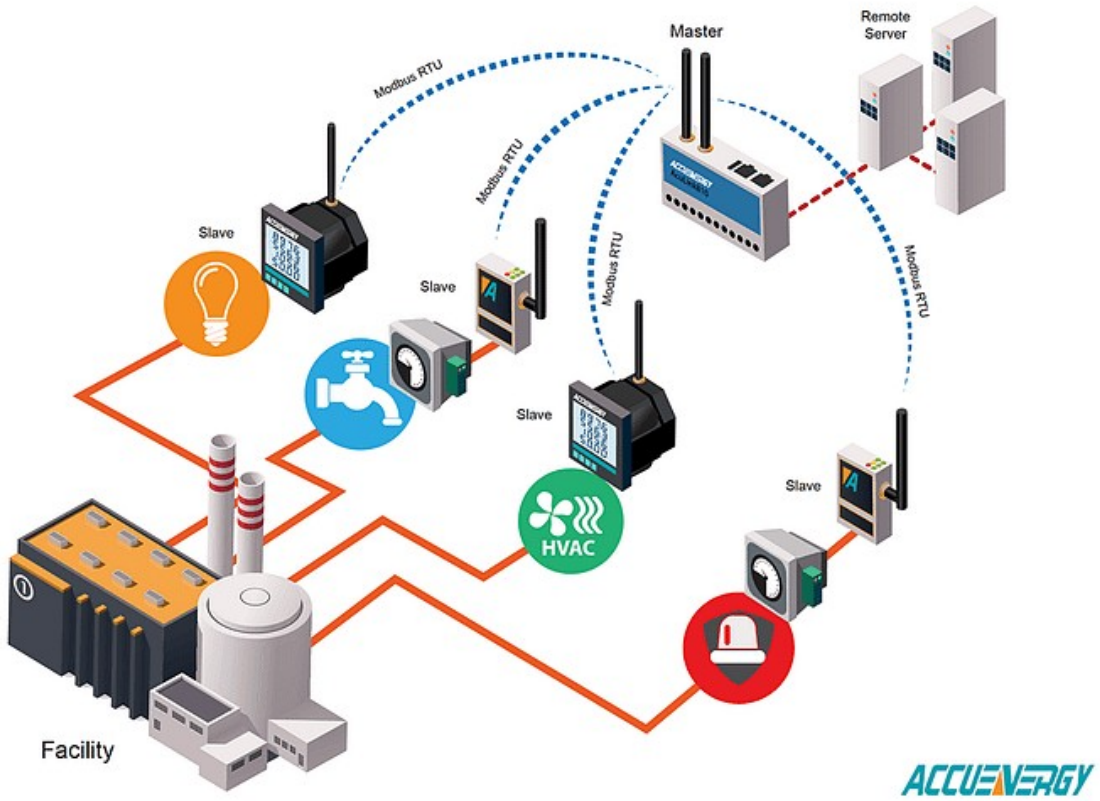
wireshark -i <INTERFAZ>i <INTERFAZ>

Luego, usar Scapy para modificar y reenviar paquetes:

de scapy. todo importa * modbus_packet = Ether() / IP() / TCP() / Raw(carga= '<datos_modificados>') sendp(modbus_packet)

Modbus: una puerta de entrada a la comunicación industrial

El protocolo de comunicación actual proviene del mundo de la comunicación industrial, un recurso indispensable durante décadas: **Modbus** . Este protocolo se implementa ampliamente en fábricas, centrales eléctricas e incluso en hogares inteligentes. En esta guía, presentamos los fundamentos de Modbus, exploramos algunas aplicaciones reales y, a continuación, presentamos una demostración con un ejemplo maestro-esclavo.



1. **Introducción al protocolo Modbus:** qué es, por qué es importante y los conceptos fundamentales.
2. **Casos de uso:** dónde destaca Modbus y por qué se inventó.
3. **Una implementación cliente-servidor Modbus:** examen de la estructura del código y las funciones clave, mostrando la aplicación práctica de los principios Modbus.

1. Comprensión de Modbus: el lenguaje de la automatización

Modbus es un protocolo de comunicación serie publicado originalmente por Modicon (ahora Schneider Electric) en 1979. Debido a las dificultades de los PLC entre los diferentes proveedores, Modbus se convirtió en una forma estandarizada para que estos dispositivos intercambiaran información, independientemente del fabricante. Su atractivo reside en su simplicidad y naturaleza abierta, lo que propició su amplio uso en diversas industrias.

Caracteres clave de Modbus:

- **Arquitectura Maestro-Esclavo (Cliente-Servidor):** Modbus funciona según el principio de solicitud-respuesta. Un dispositivo **maestro** (o **cliente**) envía una solicitud a un dispositivo **esclavo** (o **servidor**), y este responde con los datos solicitados o ejecuta el comando. Solo el maestro puede iniciar la comunicación.
- **Comunicación en serie:** Originalmente diseñado para líneas seriales como RS-232 y RS-485, Modbus ha evolucionado para funcionar en redes TCP/IP (Modbus TCP).
- **Organización de datos:** Modbus define cuatro tablas de datos principales: **1. Bobinas (salidas discretas):** datos de lectura/escritura de un solo bit, que se

utilizan normalmente para controlar los estados de ***encendido/apagado*** *de los dispositivos* (p. ej., encender/apagar un motor).

2. Entradas discretas: datos de solo lectura de un solo bit, que se utilizan para *representar el* ***estado*** *de los sensores* (p. ej., un interruptor de límite está abierto/cerrado).

3. Registros de retención: datos de lectura/escritura de 16 bits, que se utilizan para *almacenar parámetros de configuración o valores numéricos* (p. ej., un punto de ajuste de temperatura).

4. Registros de entrada: datos de solo lectura de 16 bits, que se utilizan para representar valores de entrada analógicos (p. ej., una lectura de temperatura).

5. Códigos de función: los comandos específicos se definen utilizando códigos de función, que indican al dispositivo esclavo qué acción realizar (p. ej., leer registros de retención, escribir una sola bobina).

2. Modbus en acción: dónde y por qué se utiliza

Modbus se utiliza en casi todas partes del mundo. Su simplicidad, baja sobrecarga y facilidad de implementación lo han consolidado en diversas aplicaciones, entre ellas:

- **Automatización industrial:** control de procesos de fabricación, supervisión del estado de los equipos y recopilación de datos de sensores.
- **Automatización de edificios:** gestión de sistemas HVAC, controles de iluminación y sistemas de seguridad.
- **Gestión energética:** monitorización del consumo energético, control de fuentes de energía renovables (solar, eólica) e integración con redes inteligentes.
- **Transporte:** Control de señales de tráfico, monitoreo de sistemas ferroviarios y gestión de operaciones de flotas.
- **Y mucho más:** en cualquier lugar donde necesite una comunicación simple y confiable entre dispositivos, Modbus es un fuerte competidor.

3. Demostración práctica: Modbus maestro-esclavo en Python

Para demostrar el funcionamiento de Modbus, se implementó el proyecto modbussim . Puede clonarlo y probarlo en su entorno de desarrollo. El repositorio del proyecto proporciona una implementación cliente-servidor Modbus escrita en Python, junto con una **API Flask** para interactuar con el servidor Modbus a través de solicitudes HTTP.

main.py:Este es el punto de entrada de la aplicación. Inicia el servidor Modbus y la API de Flask en subprocesos separados, lo que permite su ejecución simultánea. También incluye un bucle principal que realiza llamadas periódicas update_registers()para simular cambios en los datos del sensor.

 server.py: Contiene la implementación del servidor Modbus.
 flask_api.py: Contiene la implementación de la API de Flask.
 client.py: Contiene la implementación del cliente Modbus.

Funciones cruciales y fragmentos de código:

 server.py: start_modbus_server(): Esta función se encarga de configurar e iniciar el servidor Modbus TCP. Utiliza la pymodbusbiblioteca. Observe cómo toma el contextobjeto (que define las tablas de datos Modbus) y lo vincula a un puerto TCP:

```
desde pymodbus.server.async_io importar StartTcpServer
# ...
StartTcpServer(contexto=contexto, dirección=(“localhost”, MODBUS_PORT),
identidad=identificación)
```

Esto refleja directamente el concepto Modbus TCP de servir datos a través de una red.
 server.py: update_registers()Esta función simula los datos del sensor y actualiza los bloques de datos Modbus. Muestra cómo se asignan los datos reales (en este caso, temperaturas simuladas) a los registros Modbus.

```
def update_registers():def update_registers ():
 # ...
temperaturas_actuales, promedios_históricos, estado_de_enfriamiento,
estados_de_alarma = simular_datos_de_temperatura(
 temperaturas_actuales, promedios_históricos
)
# Actualizar los bloques de datos
holding_registers.setValues( 0 , temperaturas_actuales) # Datos de temperatura
coils.setValues( 0 , estado_de_enfriamiento) # Estado de enfriamiento como booleanos
```

La holding_registers.setValues()función es una implementación directa de la escritura de datos en una tabla de datos Modbus específica.

- **client.py: run_modbus_client () Esta función implementa el cliente Modbus. Se conecta al servidor y lee periódicamente datos de varios registros:**

  ```
  de pymodbus.client importar ModbusTcpClient
  cliente = ModbusTcpClient(MODBUS_SERVER_IP,
  puerto=MODBUS_SERVER_PORT)
  respuesta = cliente.read_holding_registers(dirección= 0 , conteo= 9 , esclavo= 1 )
  #Leer datos
  temporales si respuesta.isError():
  logging.info( f"Error al leer los registros de retención: {respuesta} " )
   de lo contrario :
   logging.info( f"Temperaturas actuales: {respuesta.registros} " )
  ```

  La client.read_holding_registers() función demuestra la funcionalidad principal del cliente Modbus de solicitar datos utilizando un código de función específico (en este caso, leer registros de retención).
- **flask_api.py: modbus_register ()** Esta función de la API de Flask permite interactuar con el servidor Modbus mediante solicitudes HTTP. Permite obtener y colocar valores en registros Modbus con diferentes tipos de datos.

```
# API REST para interactuar con el servidor Modbus
@app.route( '/modbus/<string:register>/< int :address>', methods=['GET', 'PUT'] )
def modbus_register ( register, address ):
try :
register = register.lower() # Estandarizar caso
```

```
 store = get_register(register)
 if request.method == 'PUT' :
 data = request.get_json() # Usar get_json para un análisis más seguro
decimal_value = data[ "value" ]
data_type = str (data[ "data_type" ]).upper() # Asegurar el tipo de cadena
 hex_values = convert_to_registers(decimal_value, data_type)
# Actualizar el/los registro(s) Modbus
 set_register_value(register, address, hex_values)
 log.info( f"Valores actualizados en la dirección {address} : {current_values} " )
```

Cómo se conectan la teoría y la implementación:

El código refleja directamente los principios básicos de Modbus:

 Modelo Cliente-Servidor: El **client.py** solicita datos activamente, mientras que el server.pysirve pasivamente.

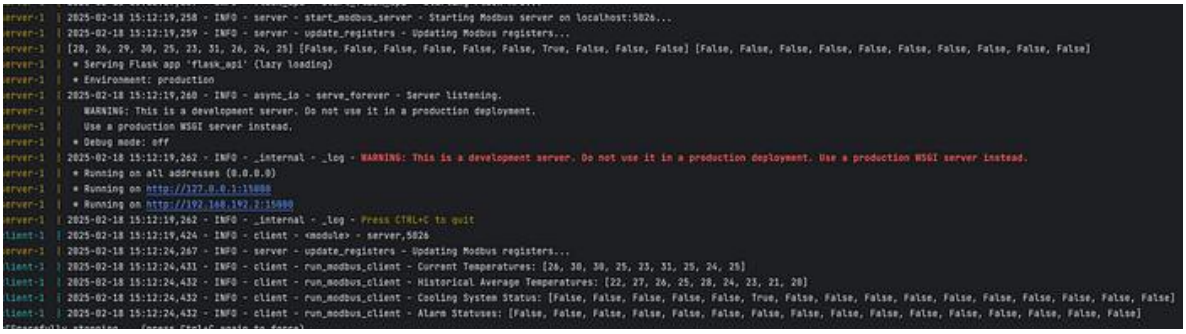
 Organización de datos: Los **ModbusSequentialDataBlock**objetos representan las bobinas, las entradas discretas, los registros de retención y los registros de entrada.

 Códigos de función: aunque no los hemos definido explícitamente como variables, las client**.read_holding_registers (), client.read_coils ()** funciones, etc., son abstracciones de los códigos de función Modbus.

 Espacio de direcciones: el **addressparámetro** en las funciones de lectura y escritura especifica la dirección del registro dentro de la tabla de datos Modbus.

Para ejecutar esta demostración:

1. Clonar el **repositorio:** git clone git@github.com:cemakpolat/iot-simulated-devices.git
2. **cd modbussim**, y luego tienes dos opciones: instalación manual o docker compose
3. Aplique uno de los enfoques definidos en el repositorio de GitHub y los resultados de la consola serán los siguientes:



Para interactuar con la API de Flask y manipular los valores de registro, puede probar los ejemplos a continuación:

- **GET dirección de registro de tenencia 0**

rizo http://localhost:15000/modbus/holding_registers/0

PUT dirección del registro de tenencia 0

curl -X PUT -H "Tipo de contenido: aplicación/json" -d"Tipo de contenido: aplicación/json" -d

'{ "valor" : 1234 , "tipo_de_datos" : "UINT16" }'
http: //localhost:15000/modbus/holding_registers/0
Al examinar el código y ejecutar la demostración, puede obtener una comprensión más profunda de cómo funciona Modbus y cómo se implementan sus principios en el software.
Resumen
Modbus es un pilar de la comunicación industrial, conocido por su simplicidad y robustez. Desde fábricas hasta hogares inteligentes, permite que los dispositivos intercambien datos de forma fiable. Al explorar el repositorio de GitHub, has adquirido una comprensión práctica de cómo los principios de Modbus se traducen en código real. Puedes ampliar tu aprendizaje modificando el código, añadiendo nuevas funciones y experimentando con diferentes configuraciones de Modbus. ¡Que disfrutes programando!
Código fuente : https://github.com/cemakpolat/iot-simulated-devices/modbussim
Referencias:
1. https://www.modbus.org/
2. https://theautomization.com/modbus-ascii-vs-modbus-rtu-vs-modbus-tcpip/

herramientas Modbus
Para pruebas, simulación y programación.
HOGAR PRODUCTOS ORDEN DESCARGAR MODBUS CONTACTO Descripción del protocolo Modbus El protocolo MODBUS© es una estructura de mensajería ampliamente utilizada para establecer la comunicación maestro-esclavo entre dispositivos inteligentes. Un mensaje MODBUS enviado de un maestro a un esclavo contiene la dirección del esclavo, el comando (p. ej., "leer registro" o "escribir registro"), los datos y una suma de comprobación (LRC o CRC). Dado que el protocolo Modbus es simplemente una estructura de mensajería, es independiente de la capa física subyacente. Tradicionalmente, se implementa mediante RS232, RS422 o RS485. La solicitud. El código de función en la solicitud indica al dispositivo esclavo destinatario qué tipo de acción realizar. Los bytes de datos contienen cualquier información adicional que el esclavo necesitará para realizar la función. Por ejemplo, el código de función 03 solicitará al esclavo que lea los registros de retención y responda con su contenido. El campo de datos debe contener la información que indica al esclavo en qué registro comenzar y cuántos registros leer. El campo de comprobación de errores proporciona un método para que el esclavo valide la integridad del contenido del mensaje. La respuesta. Si el esclavo da una respuesta normal, el código de función en la respuesta es un eco del código de función en la solicitud. Los bytes de datos contienen los datos recopilados por el esclavo, como los valores de registro o el estado. Si se produce un error, el código de función se modifica para indicar que se trata de una respuesta de error, y los bytes de datos contienen un código que describe el error. El campo de comprobación de errores permite al maestro confirmar la validez del contenido del mensaje. Los controladores se pueden configurar para comunicarse en redes Modbus estándar utilizando cualquiera de dos modos de transmisión: ASCII o RTU. Modo ASCII.
Cuando los controladores se configuran para comunicarse en una red Modbus mediante el modo ASCII (Código Estándar Americano para el Intercambio de Información), cada byte

de ocho bits de un mensaje se envía como dos caracteres ASCII. La principal ventaja de este modo es que permite intervalos de hasta un segundo entre caracteres sin causar errores.

Sistema de codificación

Caracteres hexadecimales ASCII imprimibles 0 ... 9, A ... F

Bits por byte

1 bit de inicio

7 bits de datos, el bit menos significativo se envía primero

1 bit para paridad par/impar - ningún bit si no hay paridad

1 bit de parada si se utiliza paridad - 2 bits si no hay paridad

Comprobación de errores

Comprobación de redundancia longitudinal (LRC)

Modo RTU:

Cuando los controladores se configuran para comunicarse en una red Modbus mediante el modo RTU (Unidad Terminal Remota), cada byte de ocho bits de un mensaje contiene dos caracteres hexadecimales de cuatro bits. La principal ventaja de este modo es que su mayor densidad de caracteres permite un mejor rendimiento de datos que el ASCII para la misma velocidad en baudios. Cada mensaje debe transmitirse en un flujo continuo.

Sistema de codificación

Binario de ocho bits, hexadecimal 0 ... 9, A ... F

Dos caracteres hexadecimales contenidos en cada campo de ocho bits del mensaje

Bits por byte

1 bit de inicio

8 bits de datos, el bit menos significativo se envía primero

1 bit para paridad par/impar - ningún bit si no hay paridad

1 bit de parada si se utiliza paridad - 2 bits si no hay paridad

Campo de comprobación de errores

Comprobación de redundancia cíclica (CRC)

En modo ASCII, los mensajes comienzan con dos puntos (:) (ASCII 3A hexadecimal) y terminan con un par de retorno de carro y avance de línea (CRLF) (ASCII 0D y 0A hexadecimal).

Los caracteres permitidos para todos los demás campos son hexadecimales 0 ... 9, A ... F. Los dispositivos en red monitorean el bus de red continuamente para detectar el carácter de dos puntos. Al recibir uno, cada dispositivo decodifica el siguiente campo (el campo de dirección) para determinar si corresponde al dispositivo al que se dirige.

Pueden transcurrir intervalos de hasta un segundo entre caracteres dentro del mensaje. Si el intervalo es mayor, el dispositivo receptor asume que se ha producido un error. A continuación se muestra una trama de mensaje típica.

Comenzar	DIRECCIÓN	Función	Datos	LRC	Fin
:	2 caracteres	2 caracteres	N caracteres	2 caracteres	CR LF

Entramado RTU

En el modo RTU, los mensajes comienzan con un intervalo de silencio de al menos 3,5 veces el carácter. Esto se implementa más fácilmente como un múltiplo de veces el carácter a la velocidad en baudios que se utiliza en la red (mostrado como T1-T2-T3-T4 en la figura siguiente). El primer campo que se transmite es la dirección del dispositivo. Los caracteres permitidos transmitidos para todos los campos son hexadecimales 0 ... 9, A ... F. Los dispositivos en red monitorean el bus de red continuamente, incluso durante los intervalos de silencio. Cuando se recibe el primer campo (el campo de dirección), cada dispositivo lo decodifica para averiguar si es el dispositivo direccionado. Después del último carácter transmitido, un intervalo similar de al menos 3,5 veces el carácter marca el final del mensaje. Un nuevo mensaje puede comenzar después de este intervalo. La trama completa del mensaje debe transmitirse como un flujo continuo. Si ocurre un intervalo de silencio de más de 1,5 veces el carácter antes de que se complete la trama, el dispositivo receptor vacía el mensaje incompleto y asume que el siguiente byte será el campo de dirección de un nuevo mensaje.
De igual forma, si un nuevo mensaje comienza más de 3,5 caracteres después de un mensaje anterior, el dispositivo receptor lo considerará una continuación del mensaje anterior. Esto generará un error, ya que el valor del campo CRC final no será válido para los mensajes combinados. A continuación se muestra un marco de mensaje típico.

Comenzar	DIRECCIÓN	Función	Datos	CRC	Fin
3,5 Tiempo de char	8 bits	8 bits	N * 8 bits	16 bits	3,5 Tiempo de char

Campo de dirección:
El campo de dirección de una trama de mensaje contiene dos caracteres (ASCII) u ocho bits (RTU). A cada dispositivo esclavo se le asignan direcciones en el rango de 1 a 247.
Campo de función
El campo Código de Función indica al esclavo direccionado la función que debe realizar. Las siguientes funciones son compatibles con Modbus Poll.
01 (0x01) Leer bobinas
02 (0x02) Leer entradas discretas
03 (0x03) Leer registros de retención
04 (0x04) Leer registros de entrada
05 (0x05) Escribir bobina única
06 (0x06) Escribir registro único
08 (0x08) Diagnóstico (solo línea serie)
11 (0x0B) Obtener contador de eventos de comunicación (solo línea serie)
15 (0x0F) Escribir varias bobinas
16 (0x10) Escribir varios registros
17 (0x11) ID de servidor de informes (solo línea serie)
22 (0x16) Enmascarar registro de escritura
23 (0x17) Leer/escribir varios registros
43 / 14 (0x2B / 0x0E) Leer identificación del dispositivo
El campo de datos contiene los datos solicitados o enviados.

Contenido del campo de comprobación de errores.

Se utilizan dos tipos de métodos de comprobación de errores para las redes Modbus estándar. El contenido del campo de comprobación de errores depende del método utilizado.

ASCII:

Cuando se utiliza el modo ASCII para la estructura de caracteres, el campo de comprobación de errores contiene dos caracteres ASCII. Estos caracteres son el resultado de un cálculo de Comprobación de Redundancia Longitudinal (LRC) realizado sobre el contenido del mensaje, excluyendo los dos puntos iniciales y los caracteres CRLF finales. Los caracteres LRC se añaden al mensaje como el último campo que precede a los caracteres CRLF. Código de ejemplo LRC.

RTU:

Cuando se utiliza el modo RTU para la estructuración de caracteres, el campo de comprobación de errores contiene un valor de 16 bits implementado como dos bytes de ocho bits. El valor de comprobación de errores es el resultado de un cálculo de comprobación de redundancia cíclica (CRC) realizado sobre el contenido del mensaje. El campo CRC se añade al mensaje como último campo. En este caso, se añade primero el byte de orden inferior del campo, seguido del byte de orden superior. El byte de orden superior de CRC es el último byte que se envía en el mensaje.

Código de ejemplo de CRC.

Función 01 (01hex) Leer bobinas

Lee el estado de encendido/apagado de bobinas discretas en el esclavo.

Solicitud

El mensaje de solicitud especifica la bobina inicial y la cantidad de bobinas a leer. Ejemplo de una solicitud para leer 13 direcciones de bobinas 10...22 (bobinas 11 a 23) desde la dirección del dispositivo esclavo 4:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	01	0 1
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	0A	0 A
Cantidad de bobinas Hola	00	0 0
Cantidad de bobinas Lo	0D	0 días
Comprobación de errores Lo	DD	LRC (E 4)
Comprobación de errores Hola	98	
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

El mensaje de respuesta de estado de la bobina se empaqueta como una bobina por bit del campo de datos. El estado se indica como: 1 es el valor ON y 0 es el valor OFF. El LSB del primer byte de datos contiene la bobina a la que se dirige la solicitud. Las demás bobinas se dirigen hacia el extremo de orden superior de este byte y, de orden inferior a orden superior, en los bytes subsiguientes. Si la cantidad de bobinas devuelta no es múltiplo de ocho, los bits restantes del último byte de datos se rellenarán con ceros (hacia el extremo de orden superior del byte). El campo de recuento de bytes especifica la cantidad de bytes de datos completos. Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	01	0 1
Recuento de bytes	02	0 2
Datos (Bobinas 18...11)	0A	0 A
Datos (Bobinas 23...19)	11	1 1
Comprobación de errores Lo	B3	LRC (DE)
Comprobación de errores Hola	50	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	7	15

Función 02(02hex) Leer entradas discretas

Lee el estado ON/OFF de las entradas discretas en el esclavo.

Solicitud

El mensaje de solicitud especifica la entrada inicial y la cantidad de entradas a leer. Ejemplo de una solicitud para leer 13 entradas dirección 10...22 (entradas 10011 a 10023) desde la dirección del dispositivo esclavo 4:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	02	0 2
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	0A	0 A
Cantidad de insumos Hola	00	0 0
Cantidad de insumos Lo	0D	0 días
Comprobación de errores Lo	99	LRC (E 3)
Comprobación de errores Hola	98	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

El mensaje de respuesta de estado de entrada se empaqueta como una entrada por bit del campo de datos. El estado se indica como: 1 es el valor ON y 0 es el valor OFF. El LSB del primer byte de datos contiene la entrada a la que se dirige la solicitud. Las demás entradas se dirigen hacia el extremo de orden superior de este byte y, de orden inferior a orden superior, en los bytes subsiguientes. Si la cantidad de entrada devuelta no es múltiplo de ocho, los bits restantes del último byte de datos se rellenarán con ceros (hacia el extremo de orden superior del byte). El campo de recuento de bytes especifica la cantidad de bytes de datos completos.

Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	02	0 2
Recuento de bytes	02	0 2
Datos (Entradas 18...11)	0A	0 A
Datos (Entradas 23...19)	11	1 1
Comprobación de errores Lo	B3	LRC (DD)
Comprobación de errores Hola	14	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	7	15

Función 03 (03hex) Leer registros de retención

Leer el contenido binario de los registros de retención en el esclavo.

Solicitud

El mensaje de solicitud especifica el registro inicial y la cantidad de registros que se leerán. Ejemplo de una solicitud para leer 0...1 (registros 40001 a 40002) desde el dispositivo esclavo 1:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	03	0 3
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	00	0 0
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Comprobación de errores Lo	C4	LRC (FA)
Comprobación de errores Hola	0B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

Los datos de registro en el mensaje de respuesta se empaquetan en dos bytes por registro, con el contenido binario justificado a la derecha dentro de cada byte. Para cada registro, el primer byte contiene los bits de orden superior y el segundo, los de orden inferior.

Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	03	0 3
Recuento de bytes	04	0 4
Hola datos	00	0 0
Datos Lo	06	0 6
Hola datos	00	0 0
Datos Lo	05	0 5
Comprobación de errores Lo	DA	LRC (ED)
Comprobación de errores Hola	31	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	9	19

Función 04 (04hex) Leer registros de entrada

Leer el contenido binario de los registros de entrada en el esclavo.

Solicitud

El mensaje de solicitud especifica el registro inicial y la cantidad de registros que se leerán. Ejemplo de una solicitud para leer 0...1 (registros 30001 a 30002) desde el dispositivo esclavo 1:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	04	0 4
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	00	0 0
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Comprobación de errores Lo	71	LRC (F 9)
Comprobación de errores Hola	CB	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

Los datos de registro en el mensaje de respuesta se empaquetan en dos bytes por registro, con el contenido binario justificado a la derecha dentro de cada byte. Para cada

registro, el primer byte contiene los bits de orden superior y el segundo, los de orden inferior.

Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	04	0 4
Recuento de bytes	04	0 4
Hola datos	00	0 0
Datos Lo	06	0 6
Hola datos	00	0 0
Datos Lo	05	0 5
Comprobación de errores Lo	Base de datos	LRC (CE)
Comprobación de errores Hola	86	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	9	19

Función 05 (05hex) Escribir bobina simple

Escribe una sola bobina en ON o OFF.

Solicitud:

El mensaje de solicitud especifica la referencia de la bobina que se escribirá. Las bobinas se direccionan desde cero; la bobina 1 se direcciona como 0.

El estado de encendido/apagado solicitado se especifica mediante una constante en el campo de datos de solicitud. Un valor de FF 00 hexadecimal indica que la bobina está encendida. Un valor de 00 00 indica que está apagada. Todos los demás valores son ilegales y no afectan a la bobina.

A continuación se muestra un ejemplo de una solicitud para escribir la bobina 173 ON en el dispositivo esclavo 17:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	05	0 5
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	C.A.	C.A.
Escribir datos Hola	FF	0 0
Escribir datos Lo	00	FF
Comprobación de errores Lo	4E	LRC (3 pisos)
Comprobación de errores Hola	8B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

La respuesta normal es un eco de la solicitud, que se devuelve después de escribir el estado de la bobina.

Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	05	0 5
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	C.A.	C.A.
Escribir datos Hola	FF	0 0
Escribir datos Lo	00	FF
Comprobación de errores Lo	4E	LRC (3 pisos)
Comprobación de errores Hola	8B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Función 06 (06hex) Escribir un solo registro

Escribe un valor en un único registro de retención.

Solicitud:

El mensaje de solicitud especifica la referencia del registro que se escribirá. Los registros se direccionan desde cero; el registro 1 se direcciona como 0.

El valor de escritura solicitado se especifica en el campo de datos de la solicitud. A continuación, se muestra un ejemplo de una solicitud de escritura del registro 40002 al 00 03 hexadecimal en el dispositivo esclavo 17.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	06	0 6
Registrar dirección Hola	00	0 0
Registrar Dirección Lo	01	0 1
Escribir datos Hola	00	0 0
Escribir datos Lo	03	0 3
Comprobación de errores Lo	9A	LRC (E 5)
Comprobación de errores Hola	9B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta

La respuesta normal es un eco de la solicitud, devuelta después de que se hayan escrito los contenidos del registro.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	06	0 6
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	01	0 1
Escribir datos Hola	00	0 0
Escribir datos Lo	03	0 3
Comprobación de errores Lo	9A	LRC (E 5)
Comprobación de errores Hola	9B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Función 15 (0Fhex) Escribir múltiples bobinas Escribe cada bobina en una secuencia de bobinas en ON o OFF.

Solicitud: El mensaje de solicitud especifica las referencias de bobina que se escribirán. Las bobinas se direccionan desde cero; la bobina 1 se direcciona como 0. Los estados de encendido/apagado solicitados se especifican mediante el contenido del campo de datos de solicitud. Un 1 lógico en una posición de bit del campo solicita que las bobinas correspondientes estén encendidas. Un 0 lógico solicita que estén apagadas. A continuación se muestra un ejemplo de una solicitud para escribir una serie de diez bobinas comenzando en la bobina 20 (direccionada como 19, o 13 hexadecimal) en el dispositivo esclavo 17. El contenido de los datos de la solicitud consta de dos bytes: CD 01 hexadecimal (1100 1101 0000 0001 binario). Los bits binarios corresponden a las bobinas de la siguiente manera:

Bit: 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 1 Bobina: 27 26 25 24 23 22 21 20 - - - - - - 29 28

El primer byte transmitido (CD hexadecimal) se dirige a las bobinas 27… 20, y el bit menos significativo se dirige a la bobina más baja (20) de este conjunto. El siguiente byte transmitido (01 hexadecimal) direcciona las bobinas 29 y 28, y el bit menos significativo direcciona la bobina más baja (28) de este conjunto. Los bits no utilizados del último byte de datos deben rellenarse con ceros.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	0F	0 F
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	13	1 3
Cantidad de bobinas Hola	00	0 0
Cantidad de bobinas Lo	0A	0 A
Recuento de bytes	02	0 2
Escribir datos Hola	CD	CD
Escribir datos Lo	01	0 1
Comprobación de errores Lo	BF	LRC (F 3)
Comprobación de errores Hola	0B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	11	23

Respuesta: La respuesta normal devuelve la dirección del esclavo, el código de función, la dirección inicial y el número de bobinas escritas. A continuación, se muestra un ejemplo de respuesta a la solicitud mostrada arriba.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	0F	0 F
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	13	1 3
Cantidad de bobinas Hola	00	0 0
Cantidad de bobinas Lo	0A	0 A
Comprobación de errores Lo	26	LRC (C 3)
Comprobación de errores Hola	99	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Función 16 (10hex) Escribir múltiples registros
Escribe valores en una secuencia de registros de retención

Solicitud: El mensaje de solicitud especifica las referencias de registro que se escribirán. Los registros se direccionan desde cero; el registro 1 se direcciona como 0.

Los valores de escritura solicitados se especifican en el campo de datos de la solicitud. Los datos se empaquetan en dos bytes por registro. A continuación se muestra un ejemplo de una solicitud para escribir dos registros que comienzan en 40002 hasta 00 0A y 01 02 hexadecimal, en el dispositivo esclavo 17:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	10	1 0
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	01	0 1
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Recuento de bytes	04	0 4
Hola datos	00	0 0
Datos Lo	0A	0 A
Hola datos	01	0 1
Datos Lo	02	0 2
Comprobación de errores Lo	C6	LRC (CB)
Comprobación de errores Hola	F0	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	13	23

Respuesta: La respuesta normal devuelve la dirección del esclavo, el código de función, la dirección inicial y la cantidad de registros escritos. A continuación, se muestra un ejemplo de respuesta a la solicitud mostrada anteriormente.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	10	1 0
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	01	0 1
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Comprobación de errores Lo	12	LRC (DC)
Comprobación de errores Hola	98	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Código de ejemplo LRC Esta función es un ejemplo de cómo calcular un BYTE LRC usando el lenguaje C.

```
BYTE LRC (BYTE *nData, WORD wLength)
{
BYTE nLRC = 0 ; // LRC char initialized
for (int i = 0; i < wLength; i++)
nLRC += *nData++;
return (BYTE)(-nLRC);
} // End: LRC
```

Código de ejemplo CRC

Esta función es un ejemplo de cómo calcular una palabra CRC usando el lenguaje C.

Explicador

```
WORD CRC16 (const BYTE *nData, WORD wLength)
{
static const WORD wCRCTable[] = {
   0X0000, 0XC0C1, 0XC181, 0X0140, 0XC301, 0X03C0, 0X0280, 0XC241,
   0XC601, 0X06C0, 0X0780, 0XC741, 0X0500, 0XC5C1, 0XC481, 0X0440,
   0XCC01, 0X0CC0, 0X0D80, 0XCD41, 0X0F00, 0XCFC1, 0XCE81, 0X0E40,
   0X0A00, 0XCAC1, 0XCB81, 0X0B40, 0XC901, 0X09C0, 0X0880, 0XC841,
   0XD801, 0X18C0, 0X1980, 0XD941, 0X1B00, 0XDBC1, 0XDA81, 0X1A40,
   0X1E00, 0XDEC1, 0XDF81, 0X1F40, 0XDD01, 0X1DC0, 0X1C80, 0XDC41,
   0X1400, 0XD4C1, 0XD581, 0X1540, 0XD701, 0X17C0, 0X1680, 0XD641,
   0XD201, 0X12C0, 0X1380, 0XD341, 0X1100, 0XD1C1, 0XD081, 0X1040,
   0XF001, 0X30C0, 0X3180, 0XF141, 0X3300, 0XF3C1, 0XF281, 0X3240,
   0X3600, 0XF6C1, 0XF781, 0X3740, 0XF501, 0X35C0, 0X3480, 0XF441,
   0X3C00, 0XFCC1, 0XFD81, 0X3D40, 0XFF01, 0X3FC0, 0X3E80, 0XFE41,
   0XFA01, 0X3AC0, 0X3B80, 0XFB41, 0X3900, 0XF9C1, 0XF881, 0X3840,
   0X2800, 0XE8C1, 0XE981, 0X2940, 0XEB01, 0X2BC0, 0X2A80, 0XEA41,
   0XEE01, 0X2EC0, 0X2F80, 0XEF41, 0X2D00, 0XEDC1, 0XEC81, 0X2C40,
   0XE401, 0X24C0, 0X2580, 0XE541, 0X2700, 0XE7C1, 0XE681, 0X2640,
   0X2200, 0XE2C1, 0XE381, 0X2340, 0XE101, 0X21C0, 0X2080, 0XE041,
   0XA001, 0X60C0, 0X6180, 0XA141, 0X6300, 0XA3C1, 0XA281, 0X6240,
   0X6600, 0XA6C1, 0XA781, 0X6740, 0XA501, 0X65C0, 0X6480, 0XA441,
   0X6C00, 0XACC1, 0XAD81, 0X6D40, 0XAF01, 0X6FC0, 0X6E80, 0XAE41,
   0XAA01, 0X6AC0, 0X6B80, 0XAB41, 0X6900, 0XA9C1, 0XA881, 0X6840,
   0X7800, 0XB8C1, 0XB981, 0X7940, 0XBB01, 0X7BC0, 0X7A80, 0XBA41,
   0XBE01, 0X7EC0, 0X7F80, 0XBF41, 0X7D00, 0XBDC1, 0XBC81, 0X7C40,
   0XB401, 0X74C0, 0X7580, 0XB541, 0X7700, 0XB7C1, 0XB681, 0X7640,
   0X7200, 0XB2C1, 0XB381, 0X7340, 0XB101, 0X71C0, 0X7080, 0XB041,
   0X5000, 0X90C1, 0X9181, 0X5140, 0X9301, 0X53C0, 0X5280, 0X9241,
   0X9601, 0X56C0, 0X5780, 0X9741, 0X5500, 0X95C1, 0X9481, 0X5440,
   0X9C01, 0X5CC0, 0X5D80, 0X9D41, 0X5F00, 0X9FC1, 0X9E81, 0X5E40,
   0X5A00, 0X9AC1, 0X9B81, 0X5B40, 0X9901, 0X59C0, 0X5880, 0X9841,
   0X8801, 0X48C0, 0X4980, 0X8941, 0X4B00, 0X8BC1, 0X8A81, 0X4A40,
   0X4E00, 0X8EC1, 0X8F81, 0X4F40, 0X8D01, 0X4DC0, 0X4C80, 0X8C41,
   0X4400, 0X84C1, 0X8581, 0X4540, 0X8701, 0X47C0, 0X4680, 0X8641,
   0X8201, 0X42C0, 0X4380, 0X8341, 0X4100, 0X81C1, 0X8081, 0X4040 };
BYTE nTemp;
WORD wCRCWord = 0xFFFF;
   while (wLength--)
   {
      nTemp = *nData++ ^ wCRCWord;
      wCRCWord >>= 8;
      wCRCWord  ^= wCRCTable[nTemp];
   }
   return wCRCWord;
} // Fin: CRC16
```

Comparación de MODBUS con otros protocolos de distribución de datos

Característica	MODBUS	PROFIBUS	PROFINET	CANopen
Año de Creación	1979	1989	2003	1995
Capa Física	RS-232 / RS-485 / TCP/IP	RS-485	Ethernet Industrial	Bus CAN
Topología	Línea / Bus / Estrella	Línea Bus	Estrella, Línea, Árbol	Línea Bus
Velocidad de Comunicación	Hasta 10 Mbps (TCP)	Hasta 12 Mbps	100 Mbps - 1 Gbps	1 Mbps
Tipo de Comunicación	Maestro-Esclavo	Maestro-Esclavo	Tiempo Real, Cíclico	Maestro-Esclavo
Facilidad de Implementación	Muy Alta	Media	Media-Alta	Alta
Determinismo	Bajo (TCP), Medio (RTU)	Alto	Muy Alto (Tiempo Real)	Medio-Alto
Robustez	Media	Alta	Muy Alta	Alta
Coste de Implementación	Bajo	Medio-Alto	Alto	Bajo-Medio
Aplicaciones Típicas	Automatización simple, SCADA	Automatización industrial	Automatización avanzada, Robots	Vehículos, maquinaria médica
Compatibilidad	Muy alta	Alta	Alta	Alta

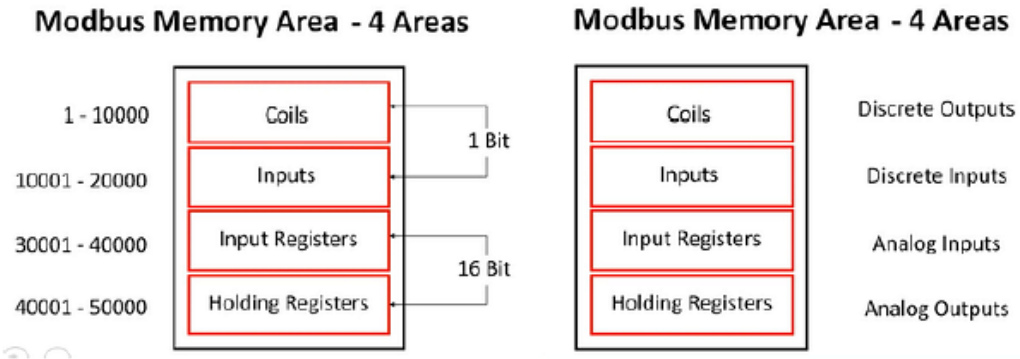
Un breve recorrido por Modbus con diagramas Tablas y diagramas de flujo de Modbus

Modbus es un protocolo de comunicación comúnmente utilizado en sistemas de automatización y control industrial. Aquí, utilizamos algunos diagramas para explicarle rápidamente sus conceptos principales.

Descripción

(1) Bloques del modelo de datos/áreas de memoria
(2) Diagramas de flujo de Modbus maestro y esclavo
(3) Desglose de Modbus en el contexto del modelo de cinco capas TCP/IP
(4) Enmarcado y desenmarcado en Modbus
(5) Códigos de función Modbus
(6) Trama de datos de comandos y respuestas Modbus
(7) CRC
(8) Referencias

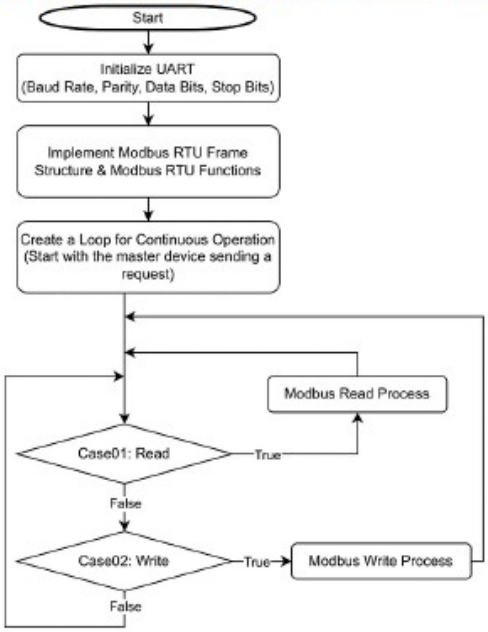
(1) Bloques del modelo de datos/Áreas de memoria



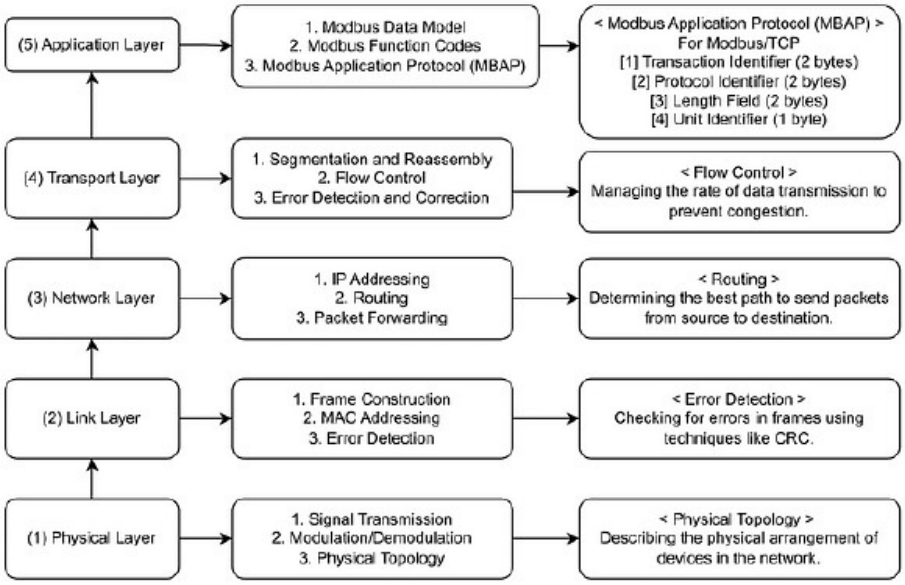
Memory Block	Data Type	Master Access	Slave Access
Coils	Boolean	Read/Write	Read/Write
Discrete Inputs	Boolean	Read-only	Read/Write
Holding Registers	Unsigned Word	Read/Write	Read/Write
Input Registers	Unsigned Word	Read-only	Read/Write

	Coils	Inputs	Input Registers	Holding Registers
Function	Discrete output	Discrete input	Analog input	Analog output
Address	1 ~ 10000	10001 ~ 20000	30001 ~ 40000	40001 ~ 50000
Size	1 bit	1 bit	16 bits	16 bits
R / W	R / W	R	R	R / W

(2) Diagramas de flujo del Modbus Maestro y Esclavo



(3) Falla de Modbus en el contexto del modelo de cinco capas TCP/IP

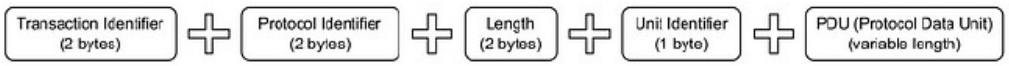


(4) Enmarcado y desenmarcado en Modbus

(1) Physical Layer: UART Framing and Deframing

(2) Transport & Network Layer: TCP & IP Framing and Deframing

(3) Application Layer: MBAP Framing and Deframing



(5) Códigos de función Modbus

Function Codes Description

Modbus Addressing Model

Decimal	Hexadecimal	Description
01	0x01	Read Coil Status
02	0x02	Read Input Status
03	0x03	Read Holding Registers
04	0x04	Read Internal Registers
05	0x05	Force Single Coil
06	0x06	Preset Single Register
15	0x0F	Force Multiple Coils
16	0x10	Preset Multiple Registers
22	0x16	Masked Write Register

Code	1/16-bit	Description	I/O Range
01	1-bit	Read coils	00001 – 10000
02	1-bit	Read contacts	10001 – 20000
05	1-bit	Write a single coil	00001 – 10000
15	1-bit	Write mulitple coils	00001 – 10000
03	16-bit	Read holding registers	40001 – 50000
04	16-bit	Read input registers	30001 – 40000
06	16-bit	Write single register	40001 – 50000
16	16-bit	Write mulitple registers	40001 – 50000
22	16-bit	Mask write register	40001 – 50000
23	16-bit	Read/write mulitple registers	40001 – 50000
24	16-bit	Read FIFO queue	40001 – 50000

Modbus Function Codes

Code	Function
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Register
07	Read Exception Status
08	Diagnostics
09	Program 484
10	Poll 484
11	Fetch Comm Event Counter
12	Fetch Comm Event Log
13	Program Controller
14	Poll Controller
15	Force Multiple Coils
16	Preset Multiple Registers
17	Report Slave ID
18	Program 884/M84
19	Reset Comm. Link
20	Read General Reference
21	Write General Reference
22	Mask Write 4x Registers
23	Read/Write 4x Registers
24	Read FIFO Queue

Function Code (Decimal)	Function Code (Hexadecimal)	1/16-bit	Description	R/W Single or Multiple Values	Memory Block
01	0x01	1-bit	Read Coil Status	Multiple	Coils
02	0x02	1-bit	Read (Discrete) Input Status	Multiple	Discrete Input
03	0x03	16-bit	Read Holding Registers	Multiple	Holding Registers
04	0x04	16-bit	Read Input Registers	Multiple	Input Registers
05	0x05	1-bit	Write Single Coil	Single	Coils
06	0x06	16-bit	Write Single Register	Single	Holding Registers
15	0x0F	1-bit	Write Multiple Coils	Multiple	Coils
16	0x10	16-bit	Write Multiple Registers	Multiple	Holding Registers

(6) Marco de datos de comandos y respuestas Modbus

Ahora, profundicemos en los detalles relacionados con los marcos de datos de los comandos y respuestas de la función Modbus:

Para una solicitud de lectura de múltiples registros (lectura de registros de retención) [0x03] de **Modbus RTU:**

Inicio (siempre 1 bit)
Dirección de esclavo (1 byte)
Código de función 0x10 (1 byte)
Dirección de inicio (2 bytes)
Cantidad de registros (2 bytes)
Comprobación CRC (2 bytes)
Fin (1 o 2 bits)

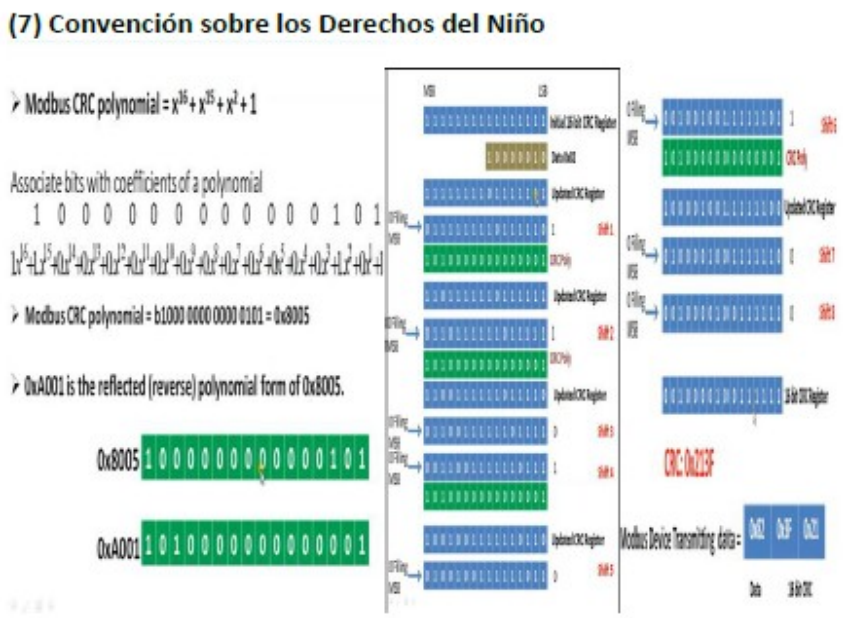
| Inicio (siempre 1 bit) | Dirección de esclavo (1 byte) | Código de función [0x03] (1 byte) | Dirección de inicio (alta) (1 byte) | Dirección de inicio (baja) (1 byte) | Cantidad (alta) (1 byte) | Cantidad (baja) (1 byte) | CRC bajo (1 byte) | CRC alto (1 byte) | Fin (1 o 2 bits) |

Query message (Function code 03)

Start	Slave address	Function code	Data				Error check	End
			Starting address		Number of registers			
	XX	03	Hi	Lo	Hi	Lo	XX	
Start								*Stop*

Figura: Diagrama delicado de formatos de comandos de función Modbus

Los formatos de comandos de funciones Modbus ilustraron de forma excelente todas las tramas de datos de los comandos (del dispositivo maestro) y las respuestas (de los dispositivos esclavos) de todas las funciones Modbus.



4) ¿Qué es un protocolo HART? ¿Para qué? Ejemplifique.

El protocolo HART (del inglés *Highway Addressable Remote Transducer-Transductor remoto direccionable de alta velocidad*) es un protocolo de comunicación híbrido que combina señales analógicas tradicionales de 4-20 mA con comunicación digital.

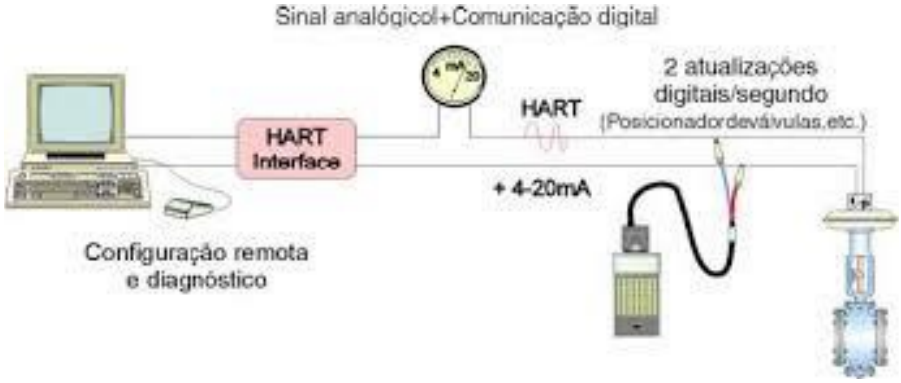
Permite la transmisión simultánea de datos digitales y analógicos sobre el mismo par de cables, facilitando la configuración remota, el monitoreo y el diagnóstico de los instrumentos. Sin interferir con la señal analógica de control.

Es una solución robusta y flexible para la comunicación digital en aplicaciones de medición y control de procesos industriales, que ofrece ventajas en términos de costo, eficiencia y flexibilidad operativa.

¿Para qué se usa?

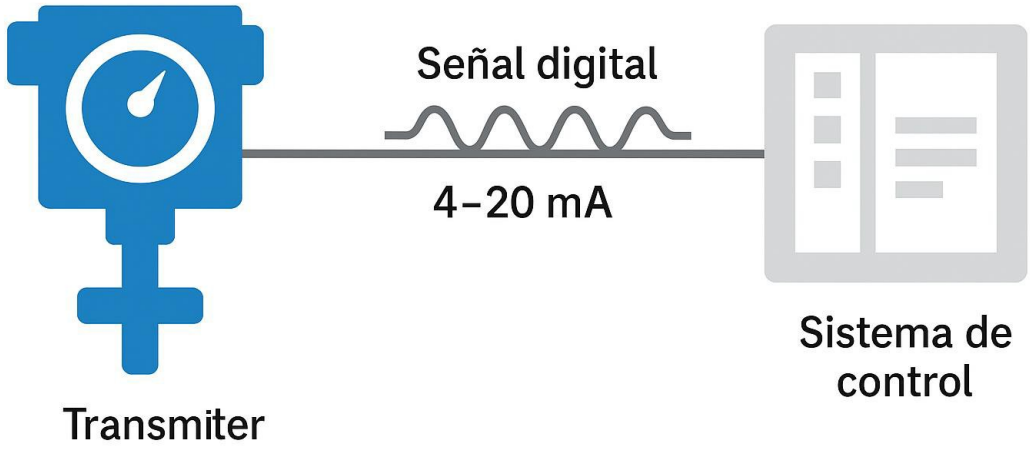
Se utiliza principalmente en la industria en procesos para la comunicación entre dispositivos de campo inteligentes (como transmisores de presión, temperatura, caudalímetros, válvulas de control) y los sistemas de control (como PLC, DCS o sistemas SCADA).

Permite configurar, calibrar, diagnosticar y monitorear dispositivos de campo de forma remota, lo que facilita el mantenimiento predictivo y reduce tiempos de inactividad



Ejemplo:

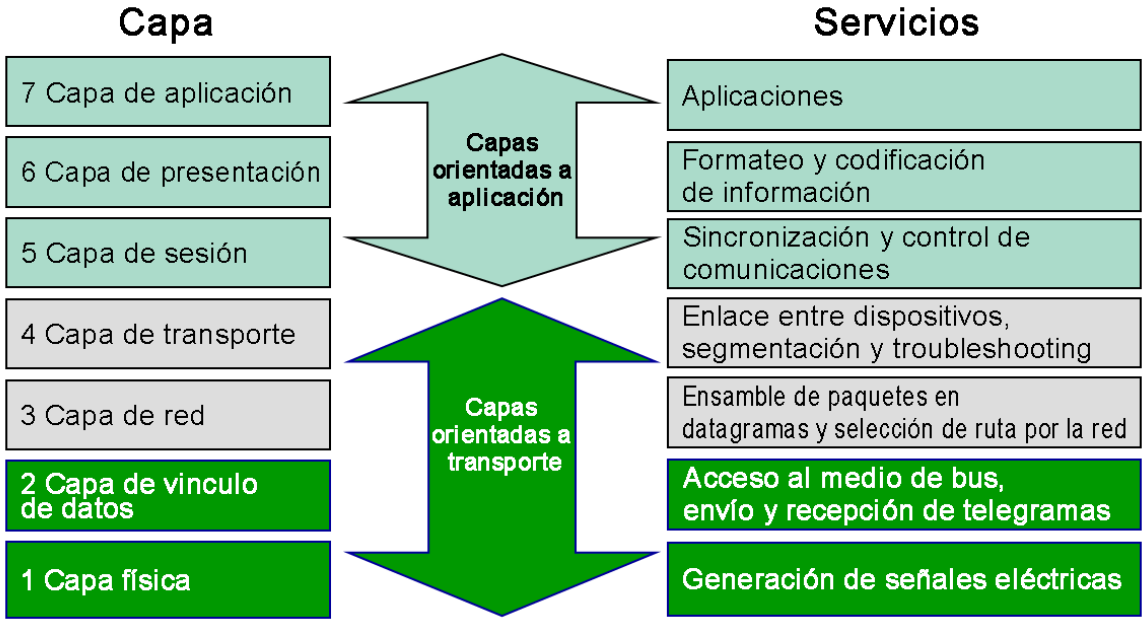
Un transmisor de presión instalado en una planta química envía su valor de presión de proceso mediante una señal de corriente analógica de 4-20 mA al sistema de control, pero además, utilizando el protocolo HART, también transmite datos digitales como la temperatura del proceso, el estado del dispositivo y parámetros de calibración, todo a través del mismo cableado.



5) **¿Qué es un protocolo PROFINET?, ¿Para qué se usa? Ejemplifique**

PROTOCOLO PROFINET

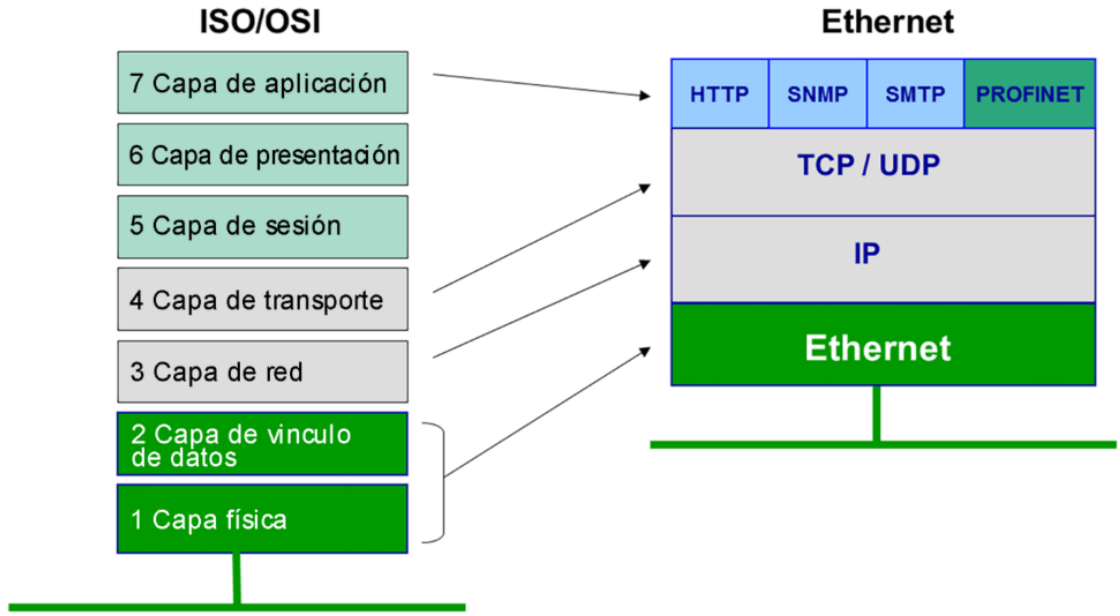
Es un protocolo de comunicación Ethernet industrial para automatización, que permite conectar dispositivos y sistemas en tiempo real. Se utiliza para la interconexión de dispositivos en entornos industriales, como sensores, actuadores, PLCs, y sistemas de control, facilitando la comunicación y el intercambio de datos entre ellos. ¿Para qué se usa? **Comunicación en tiempo real:** PROFINET garantiza la entrega de datos en el momento preciso, lo cual es crucial en aplicaciones de automatización donde el tiempo de respuesta es crítico. **Intercambio de datos entre dispositivos:** P ermite la comunicación entre dispositivos de diferentes fabricantes, facilitando la integración de sistemas de automatización. **Control y supervisión de procesos:** Facilita la supervisión de procesos industriales y la ejecución de acciones de control basadas en datos de sensores y actuadores. **Diagnóstico y mantenimiento:** Ofrece herramientas de diagnóstico para identificar y solucionar problemas en la red y los dispositivos conectados. Ejemplos de uso: **Control de movimiento en una fábrica: PROFINET** puede conectar PLCs con servos, actuadores y encoders, permitiendo un control preciso y sincronizado de los movimientos de máquinas y robótica. **Sistemas de automatización de edificios:** Se utiliza para conectar sistemas de iluminación, climatización, seguridad y otros sistemas automatizados, facilitando la gestión y el control del edificio. **Automatización de procesos de manufactura: PROFINET** interconecta dispositivos de medición, control, y sistemas de transporte en líneas de producción, mejorando la eficiencia y el control de los procesos. **Aplicaciones de seguridad: PROFINET** se utiliza en sistemas de seguridad industrial, como sistemas de acceso, detección de intrusiones y gestión de incendios, garantizando la seguridad de personas y equipos.
Las redes por lo general se representa mediante un modelo de siete capas: el modelo de referencia ISO/OSI. En el mundo de Ethernet, las siete capas se colapsan en cuatro. PROFINET utiliza todas las cuatro, pero no al mismo tiempo. Pero, nos estamos adelantando, comencemos por el principio: el modelo de siete capas. Este es el modelo de siete capas con las capas etiquetadas y sus funciones definidas:



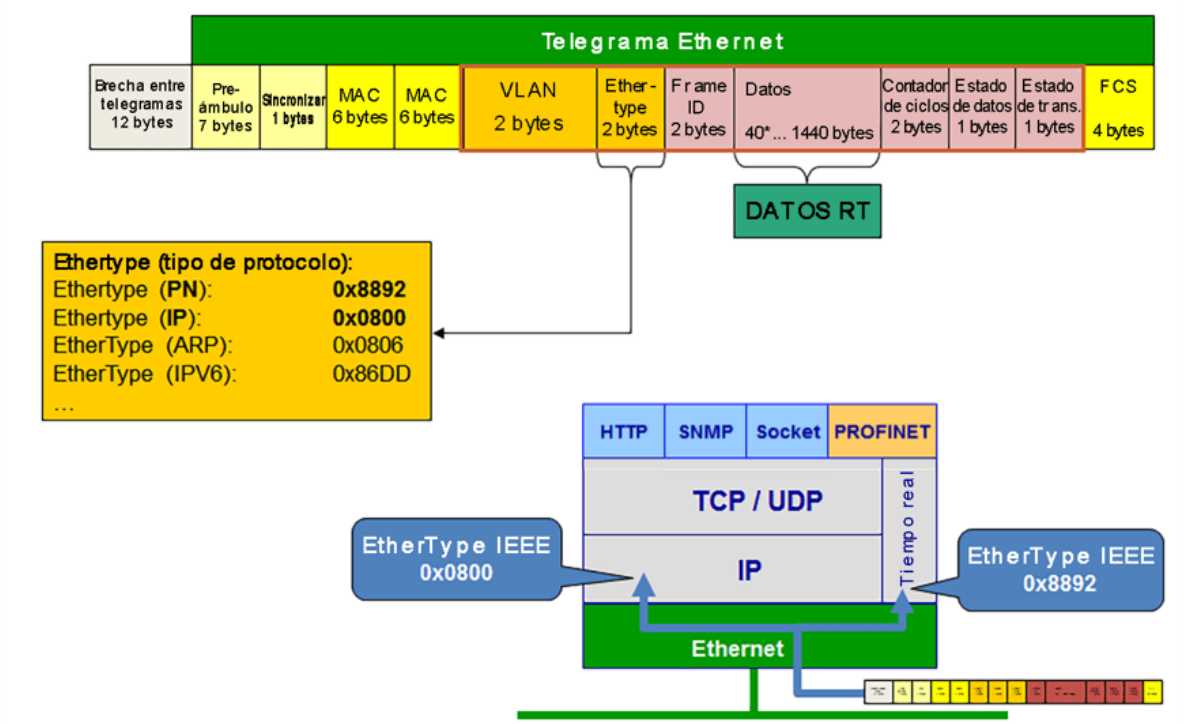
Las capas son: Física, enlace de datos, red, transporte, sesión, presentación y aplicación.

FUNCIONES DE CADA CAPA

Capa 1: las señales en los cables. Capa 2: lo que ocurre cuando el cable se conecta al dispositivo. Cómo se crea la señal en el cable. Capa 3: cuando los mensajes son más largos que un telegrama Ethernet estándar, se desensamblan al salir o reensamblan al entrar. (Piense en los mensajes de correo entrante y saliente que probablemente son más largos de la longitud de carga útil Ethernet de aproximadamente 1500 bytes). Capa 4: crea una conexión entre dos dispositivos. (Piense en una línea telefónica abierta). Capas 5 y 6: hacen lo que está descrito. Capa 7: los programas que efectivamente crean y consumen lo que se transmite. Con PROFIBUS (y muchos otros fieldbuses), se utilizan las capas 1, 2 y 7. En el mundo de Ethernet, el modelo se colapsa en cuatro capas:



Las capas 1 y 2 se combinan y están definidas por IEEE 802.3, Ethernet. La capa 3 es de IP (Protocolo de Internet). La capa 4 es de TCP o UDP (Protocolo de control de transmisión, Protocolo de datagramas de usuario). La capa 7 es la capa de aplicación. Algunas aplicaciones utilizan las cuatro capas, por ejemplo, los navegadores y el correo electrónico. Algunas aplicaciones solo usan Ethernet más la aplicación, por ejemplo, el Protocolo de resolución de direcciones (ARP). El telegrama Ethernet entrante se dirige a la siguiente capa de acuerdo al campo estándar definido en IEEE 802.3, llamado el EtherType. Hay cientos de EtherTypes, uno de ellos, (0x0800) dirige el telegrama a la capa IP. Otro, (0x0806) dirige el telegrama directamente a la aplicación ARP. Los telegramas PROFINET en tiempo real (PROFINET RT) utilizan el EtherType 0x8892, por lo que el telegrama es dirigido a la aplicación PROFINET directamente de la capa dos (Ethernet). Esto evita el tiempo variable necesario para su procesamiento por las capas TCP/IP, por lo tanto, mejora tanto la velocidad como el determinismo.



PROFINET también usa comunicaciones TCP/IP, pero solo para tareas sin tiempo crítico como configuración y datos de diagnóstico. PROFINET también puede utilizar otras técnicas para lograr rendimiento más rápido, pero PROFINET RT maneja más del 90 % de los requisitos de las aplicaciones.

Descripción detallada:

El protocolo **PROFINET** ofrece una instalación flexible para satisfacer los requisitos de conectividad en aplicaciones industriales. Esto se logra gracias a su arquitectura jerárquica, la cual permite una instalación modular y flexible, lo que facilita el mantenimiento y el ensamblaje.

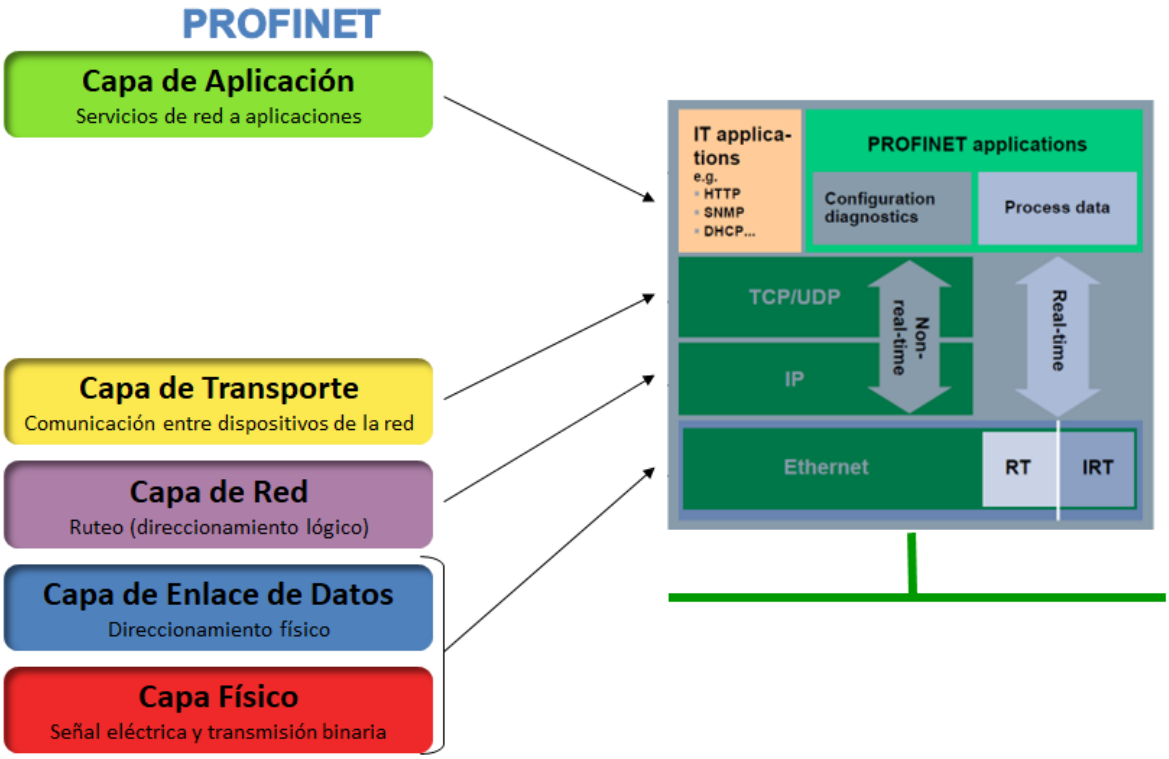
PROFINET: ¿Qué es y cómo funciona?

es un protocolo de comunicación desarrollado por la organización PROFIBUS & PROFINET International (PI). Está diseñado para conectar dispositivos industriales con varios tipos de equipamiento productivo, como motores, sensores y otros dispositivos electrónicos. **El protocolo fue diseñado para ser independiente del fabricante**, lo que significa que los dispositivos de diferentes marcas pueden funcionar juntos sin problemas. Además, el protocolo se ha optimizado para proporcionar un rendimiento óptimo a través de soluciones innovadoras como la detección automática de equipamiento, herramientas de diagnóstico avanzadas y recuperación rápida en caso de fallas. Estas características hacen que las redes basadas en PROFINET sean ideales para los entornos industriales modernos.

Características clave de PROFINET

El estándar PROFINET es una **tecnología de red de campo** para la comunicación entre equipos industriales y controladores. Está basado en Ethernet y se usa para interconectar dispositivos dentro de sistemas automatizados para el ahorro de tiempo, reducción de costes y mejora del rendimiento. El estándar PROFINET ofrece los siguientes beneficios: En primer lugar, **proporciona alta velocidad de comunicación a partir de 100 Mbps**, lo que permite realizar múltiples tareas simultáneamente al mismo tiempo. También **admite protocolos flexibles** para permitir a los dispositivos compartir

información sin conflictos. Además, el estándar PROFINET ofrece una **arquitectura escalable** que permite añadir dispositivos fácilmente según sea necesario. Utiliza estructuras jerárquicas complejas, lo que facilita la gestión del sistema, y también posee capacidades autodescriptivas e interoperables que simplifican el diseño y la configuración de redes. Por otro lado, el estándar PROFINET, al estar basado en Ethernet (IEEE 802.3), es compatible con otros protocolos como TCP-UDP/IP, SNMP, LLDP, DHCP. Esto significa que se puede manejar múltiples protocolos en un único sistema sin complicaciones adicionales. Además, cuenta con seguridad integrada para proteger sus datos confidenciales contra cualquier intruso externo o ataque malicioso. Finalmente, el estándar PROFINET ofrece funciones avanzadas como monitorización en tiempo real, diagnóstico remoto y análisis predictivo de fallas para ayudar a los operadores en su procesamiento industrial diario. Estas características contribuyen enormemente a la eficiencia operacional al permitir que los problemas detectados se aborden inmediatamente antes de que cause un daño significativo a los activos o producción del negocio industrial. Además, el estándar PROFINET cuenta con funciones para garantizar la integridad dentro del sistema industrial al permitir verificar regularmente sus componentes contra versiones anteriores no autorizadas o desactualizadas.



Detección automática

La detección automática permite a los usuarios detectar fácilmente los dispositivos conectados a la red sin necesidad de configurarlos manualmente; esta característica también facilita enormemente el mantenimiento predictivo y correctivo del software asociado al sistema.

Supervisión remota avanzada

La supervisión remota avanzada permite a los usuarios realizar un seguimiento en tiempo real del rendimiento global del sistema mediante la recopilación continua de

datos desde los dispositivos conectados; esta información puede luego ser utilizada para identificar tendencias o puntos débiles en el sistema antes de que tengan lugar fallas mayores.

Interoperabilidad

Finalmente, la **interoperabilidad** entre plataformas tecnológicas de diferentes fabricantes se logra a través de un exhaustivo proceso de certificación de todos los componentes PROFINET, en los laboratorios de prueba certificados, en la cual se garantizan que todos los requerimientos sean cumplidos, facilitando de esta manera la adopción independientemente de la plataforma.

PROFINET es el estándar de comunicación para la automatización de procesos industriales

¿Por qué es el estándar?

PROFINET es el estándar de comunicación para la automatización de procesos industriales porque ofrece ventajas en términos de eficiencia, fiabilidad y seguridad a los usuarios finales. Estas características han contribuido a que PROFINET se convierta en el estándar más extendido para la automatización industrial, con una penetración cada vez mayor en el mercado. Por un lado, PROFINET posee un **alto nivel de eficiencia debido a sus bajos requerimientos de ancho de banda y su baja latencia**. Además, su arquitectura profesional permite a los usuarios implementar rápidamente soluciones efectivas para sus necesidades específicas. Por otro lado, la fiabilidad y seguridad que ofrece PROFINET son fundamentales a la hora de trabajar con equipos críticos en entornos industriales. Su arquitectura basada en red abierta permite al usuario totalmente controlar todos los dispositivos conectados, lo que garantiza la prevención de fallas durante el proceso de trabajo. La seguridad del sistema se refuerza además mediante mecanismos tales como autenticación y cifrado. Si lo comparamos con otras soluciones disponibles actualmente para el campo industrial, podemos decir que PROFINET destaca por su capacidad para cumplir con exigencias tales como alta velocidad, complejidad en los procesos y eficiencia energética sin comprometer la estabilidad del sistema. Esto hace que sea un estándar ideal para actividades críticas donde la mínima falla puede ocasionar daños a personas, medioambiente o instalaciones industriales. Además, PROFINET ha logrado posicionarse como solución dominante gracias al creciente reconocimiento por parte del mercado industrial. Muchas empresas importantes han optado por este estándar en sus sistemas productivos debido a su simplicidad y rapidez a la hora de implementarlos, así como su capacidad para integrar tecnologías innovadoras (como redes inalámbricas).

Fieldbus: Origen

Fieldbus agrupa las tecnologías de comunicación industrial que se emplean para la interconexión entre diferentes dispositivos y sistemas de automatización. Está formado por un bus físico en el que los dispositivos conectados se comunican a través de señales eléctricas, de manera digital. El objetivo de un Fieldbus es maximizar la eficiencia energética y reducir al mínimo los requerimientos de cableado. Por lo tanto, su implementación permite ahorrar significativamente en los costes

relacionados con la instalación eléctrica, así como mejorar el rendimiento del sistema. Uno de los principales motivos para la implementación de Fieldbus es la mayor flexibilidad y disponibilidad para configurar redes industriales sin necesitar realizar modificaciones profundas en sus componentes. Esto reduce considerablemente el tiempo requerido para realizar cambios o actualizaciones en un sistema automatizado.

PROFINET para comunicaciones deterministas

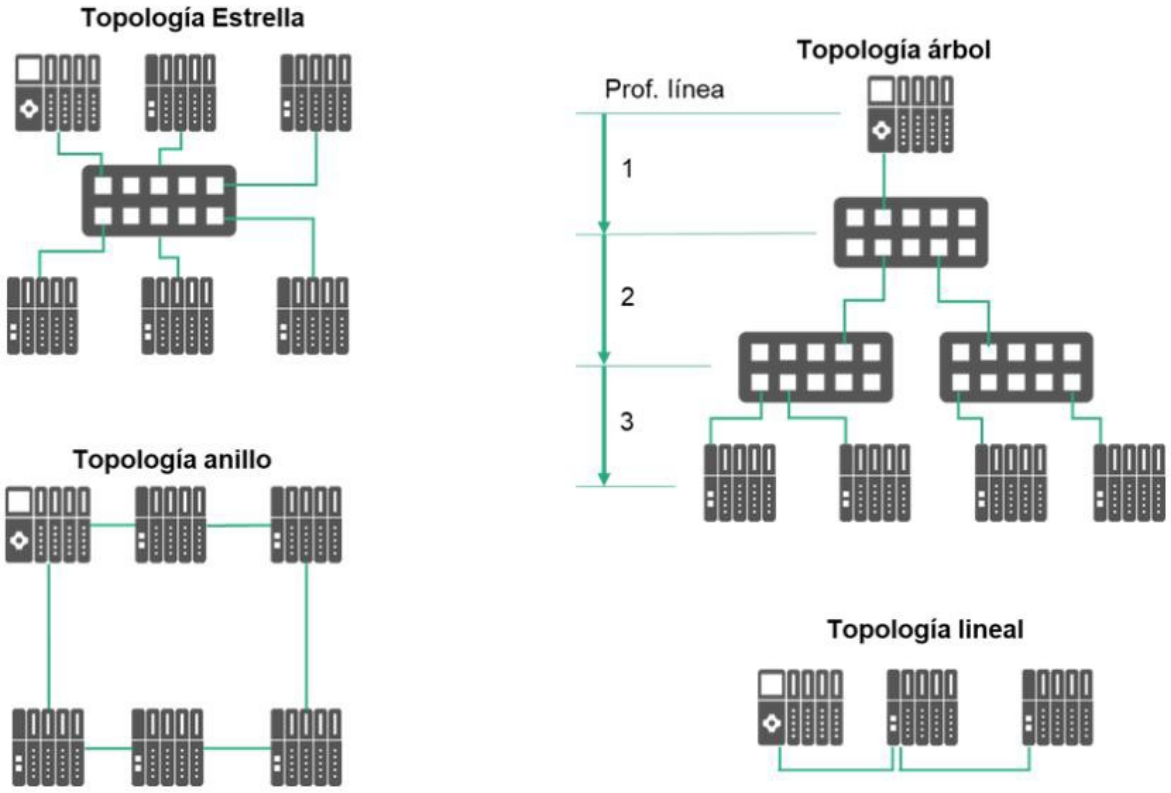
PROFINET es una red industrial desarrollada para garantizar comunicaciones deterministas. PROFINET ofrece un rendimiento excepcional gracias al uso de protocolos basados en Ethernet, con canales de comunicación de alta eficiencia. Esto le permite a PROFINET proporcionar transferencias rápidas con tiempos de reacción cortos para aplicaciones que requieren mucha información entre los nodos. Esto significa que el retardo entre la entrada (por ejemplo, desde el sensor) y la salida (por ejemplo, hacia el actuador), es perfectamente conocido en todo momento, logrando mantener al sistema bajo control. Además, PROFINET ofrece tiempos de ciclo de trabajo inferiores a los experimentados por tecnologías comunes basadas en buses serie, lo que significa un mejor rendimiento general. El bajo consumo energético y la alta fiabilidad también son grandes ventajas para este protocolo de comunicación industrial. PROFINET admite múltiples topologías tales como bus lineal, estrella lineal y anillo; así como también escalonamiento en profundidad (cascada) para permitir conexión con sitios remotos.

Escalable en tiempo real

PROFINET es una red industrial escalable en tiempo real que proporciona a los fabricantes de automatización flexibilidad y fiabilidad para la transmisión de datos. Esto se logra mediante el uso de múltiples protocolos que permiten una comunicación bidireccional entre dispositivos conectados, tales como servidores, computadoras y otros equipos industriales. Esta escalabilidad en tiempo real se traduce en mayor productividad, menor costo y menores riesgos para los fabricantes.

La escalabilidad en tiempo real de PROFINET se debe principalmente a su **arquitectura modular**, lo que significa que cada usuario puede acceder fácilmente a todas las funciones sin interferir con el

desempeño de otros dispositivos conectados. Esta arquitectura modular también permite a los usuarios agregar nuevos dispositivos y módulos cuando sea necesario, así como actualizar dinámicamente la configuración existente para mejorar el rendimiento general del sistema. Además, esta red industrial es compatible con la tecnología Ethernet y proporciona un acceso remoto ininterrumpido. Otra ventaja importante es la **facilidad de su implementación**. PROFINET utiliza protocolos estándar que reducen drásticamente el costo de formación del personal involucrado en su implementación y mantenimiento. Esto reduce significativamente tanto el impacto financiero como el tiempo requerido para instalar y ponerla en marcha. También ofrece compatibilidad con PLCs existentes y otros dispositivos industriales, lo que permite a los usuarios integrar sus sistemas existentes sin necesitar reemplazarlos por sistemas completamente nuevos o modificar radicalmente su configuración existente.



Los switches gestionados y no gestionados de PROFINET

Los switches gestionados y no gestionados de PROFINET son dispositivos de conmutación que se utilizan para la comunicación, el mantenimiento y el control en el sistema. Ambos permiten una conexión directa a la red de una manera sencilla. Los switches gestionados ofrecen capacidades adicionales como diagnósticos, monitoreo en tiempo real de la red y seguridad mejorada y maximizan la eficiencia de la comunicación a través de mecanismos de control de calidad. La diferencia principal entre los dos tipos es que los switches gestionados se pueden configurar para cumplir con diversas necesidades del sistema, mientras que los no gestionados sólo se pueden usar para un nivel básico de funcionalidad. Los switches diseñados

para soportar el protocolo PROFINET, requieren una certificación, al igual que el resto de los componentes, que se extiende en alguno de los laboratorios de prueba certificados por la organización.
Estos dispositivos están disponibles en varios formatos tales como compactos o modulares, rackeables o para montaje en riel DIN. Les permiten a las redes PROFINET ofrecer un funcionamiento óptimo gracias a características avanzadas como programación simplificada y alta disponibilidad. Además, hay una variedad de opciones disponibles cuyas funcionalidades incluyen detección automática de dispositivos, monitorización del ancho de banda, verificación FIFO (First In First Out) basada en prioridades y tecnología dinámica QoS. Los switches PROFINET también ofrecen características avanzadas como filtrado MAC/VLAN para mejorar la confidencialidad de la información e integración con servicios web para facilitar el acceso remoto a través del firewall.



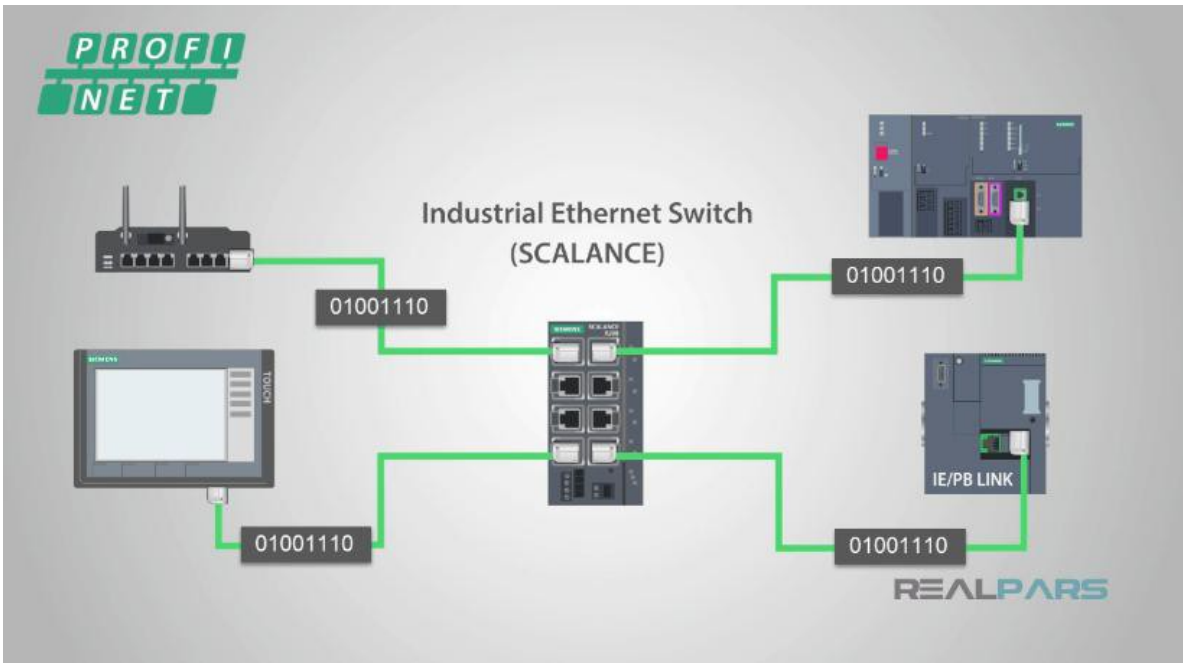



Recomendación: en una red PROFINET utilice siempre switches de tipo gestionados, debido a que sus funciones adicionales le van a permitir una ayuda importante en la gestión y diagnóstico de la red, a la vez que le van a garantizar la mejor performance posible.

¿Cómo se realiza el cableado en la red PROFINET?

El cableado PROFINET se realiza para conectar dispositivos industriales entre sí, como PLCs, sensores y acciones, en una red. Esto proporciona una forma eficiente de compartir información y datos entre dispositivos en toda una planta industrial.

El cableado PROFINET se hace mediante el uso de cables Ethernet, que son robustos y resistentes a los factores ambientales. Estos cables están equipados principalmente con conectores RJ45 (hay otros tipos de conectores disponibles https://www.profibus.com/download/PROFINET-cabling-and-interconnection-technology) que permiten la transferencia de datos a altas velocidades a lo largo del cable. Los cables se pueden utilizar para conectar dispositivos individuales o grupos de dispositivos. Si es necesario, se pueden instalar varios cables para garantizar que los dispositivos estén conectados correctamente a la red.



Los switches PROFINET también juegan un papel importante para lograr la conectividad de los dispositivos y ayudar a gestionar el tráfico de la red. Estos switches permiten la comunicación entre los dispositivos de manera rápida y sencilla, lo que facilita el intercambio de datos entre ellos. Los switches incluyen características avanzadas como protección contra fallas en el sistema, control inteligente del ancho de banda y compatibilidad con protocolos multifabricante para garantizar un rendimiento óptimo de sus redes industriales. Además, hay diversas herramientas disponibles que ayudan a monitorear la salud y estabilidad general del sistema PROFINET así como su optimización general. Estas herramientas son extremadamente útiles para diagnosticar problemas potenciales antes de que lleguen a ser críticos o provoquen interrupciones no deseadas en el procesamiento normal del trabajo dentro del sistema industrial

¿Qué cables de cobre son adecuados para PROFINET?

Los cables adecuados para PROFINET son cables trenzados y apantallados, los cuales ofrecen una mayor protección contra interferencia electromagnética externa (presente de manera permanente en instalaciones industriales). Los cables CAT5 (según EIA/TIA 568) o class D (según ISO/IEC 11801) o superiores son recomendados para la conexión de dispositivos en una red PROFINET y son necesarios solamente 4 hilos (2 pares). Estos cables permiten la vinculación entre dos dispositivos activos a una distancia máxima de 100 mts. Existen cables con diferentes coberturas las cuales los hacen aptos para instalaciones con diferentes exigencias medioambientales (industrias alimenticias, marítimas, instalaciones subterráneas, alta resistencia a ácidos, etc.) y también dependiendo del grado de movilidad, siendo posible elegir entre cables para instalaciones fijas (un solo conductor

sólido por cable) o móviles (muchos hilos más finos por cada cable que otorgan flexibilidad). Algunas especificaciones requeridas para los cables adecuados son: sección de cable adecuada para su uso (típicamente AWG22), impedancia característica, alta atenuación de acoplamiento entre pares de cables, resistencia apropiada para reducir la atenuación de la señal, etc.

Cables de fibra óptica y conectores de acuerdo con el estándar PROFINET

Cuando se requieren mayores distancias de cableado o garantizar la inmunidad a la interferencia electromagnética, se emplea como medio de transmisión la fibra óptica. Esta fibra óptica puede ser del tipo multimodo o monomodo dependiendo entre otras cosas de la distancia máxima de transmisión (fibra monomodo para mayores distancias). Siempre que hablamos de fibra óptica se requieren dos hilos para cada enlace (un hilo transmite y el otro recibe). Al igual que el caso del cobre, la cobertura del cable lo hace apto para diferentes características medioambientales. Para su instalación se deben tener en cuenta algunas particularidades como el hecho que la fibra óptica (normalmente construida con vidrio) es más susceptible a roturas por tratamiento inadecuado que un cable de cobre. La longitud típica para fibra multimodo es hasta distancias de 3000 mts (dependiendo el tipo de fibra OM1, OM2, etc. y la velocidad del enlace), para distancias mayores y hasta unos 30 km la fibra óptica debe ser monomodo (OS1 u OS2 dependiendo de la velocidad del enlace).

Los conectores para cables de fibra óptica deben cumplir los estándares SC y LC definidos por el Instituto Nacional Estadounidense para la Normalización (ANSI) e Internacional Electrotechnical Commission (IEC). Debido a la gran cantidad de aplicaciones PROFINET existentes, también hay muchas variedades disponibles en cuanto a dimensiones y materiales de los conectores. Puesto que

la fibra óptica suele emplearse en el interior de los gabinetes eléctricos, donde las instalaciones son fijas, la mayoría de los dispositivos poseen conectores de tipo LC (por cuestiones de espacio, intercambiabilidad y prestaciones). Es importante, al igual que el caso del cobre, elegir el tipo correcto de conector y fibra que sea compatible con las necesidades exactas del sistema PROFINET. PROFINET permite también la comunicación inalámbrica de manera estándar utilizando como métodos tradicionales Bluetooth y WIFI. El primero tiene como ventaja su simplicidad de uso, mientras que el segundo método permite mayor ancho de banda y por lo tanto implementar sistemas más complejos. Actualmente se está trabajando para extender el uso de PROFINET a redes 5G.

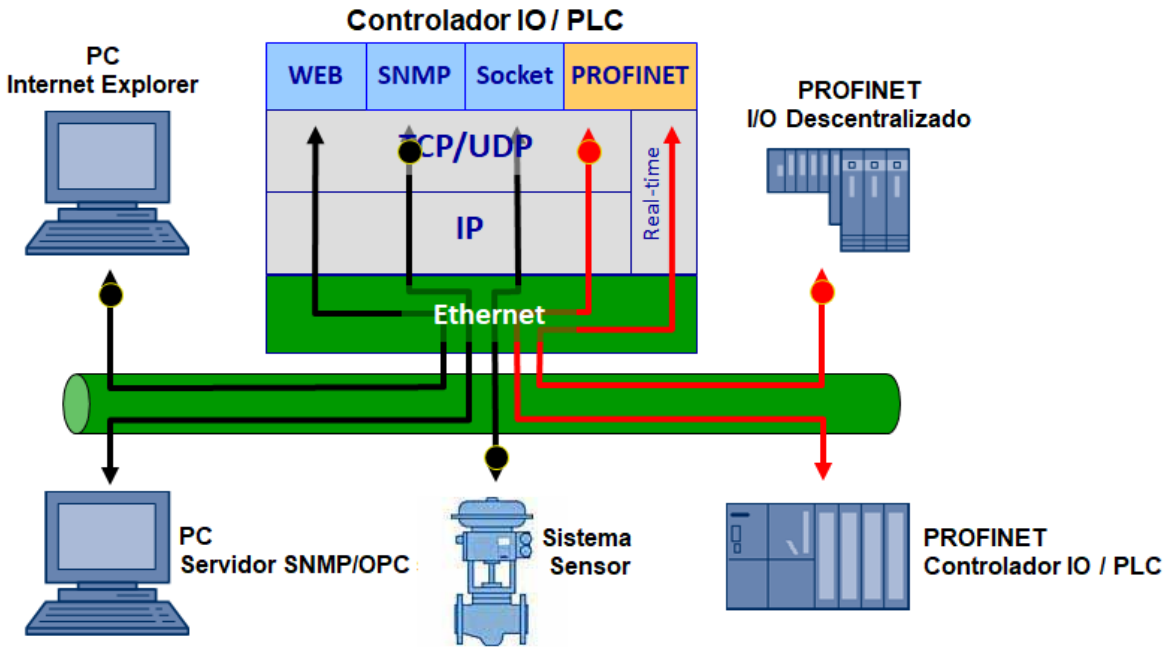
Instalación flexible

PROFINET es una red de área industrial que se encuentra entre los principales estándares de comunicación industriales. El protocolo PROFINET ofrece una instalación flexible para satisfacer los requisitos de conectividad en aplicaciones industriales. Esto se logra gracias a su arquitectura jerárquica, la cual permite una instalación modular y flexible, lo que facilita el mantenimiento y el ensamblaje. La flexibilidad de la instalación también elimina la necesidad de reubicar cableado cuando hay cambios en el lugar o en la configuración del sistema. La arquitectura PROFINET utiliza dos niveles de red: nivel 1 y nivel 2. En el nivel 1, los dispositivos directamente conectados al bus operan como redes locales, mientras que en el nivel 2 existe un controlador para manejar todos los dispositivos conectados a la red. El controlador gestiona los dispositivos basándose en sus direcciones internas (dirección MAC y dirección IP) y en el nombre asignado a cada dispositivo. Estas direcciones identifican y nombran a los dispositivos conectados al bus principal, lo que facilita la conexión entre ellos y hace posible su detección automática durante la inicialización. La configuración del sistema PROFINET también es flexible ya que se pueden agregar o quitar dispositivos sin interferir con el funcionamiento normal del sistema. Esta funcionalidad es posible gracias al uso de medios redundantes para mejorar la disponibilidad general del sistema, además de ofrecer protección contra fallas individuales. Además, las herramientas de diagnóstico incorporadas permiten detectar problemas antes de que se produzcan fallas en el sistema e incluso prevenirlas. Algunas características tales como escalabilidad y amplia disponibilidad permite a PROFINET cumplir con requisitos muy exigentes en términos de fiabilidad y rendimiento a través del uso simultáneo de varios medios físicos redundantes para distribuir información dentro de la red sin comprometer la calidad de servicio entregada por éste. La flexibilidad también le permite a PROFINET extenderse más allá del campo tradicional industrial para abarcar otros escenarios donde sea pertinente implementarlo como parte integrante en soluciones tecnológicas innovadoras.

Fast Start-Up (inicio rápido)

PROFINET es una tecnología de red industrial que ofrece muchas ventajas para la automatización de los procesos y el control. Una de sus principales características es su capacidad para proporcionar **ini cio rápido**, lo que significa que los usuarios pueden configurar e implementar sistemas muy

rápidamente. Esto se logra mediante un conjunto completo de funciones ejecutadas en tiempo real en dispositivos y aplicaciones de redes estándar.



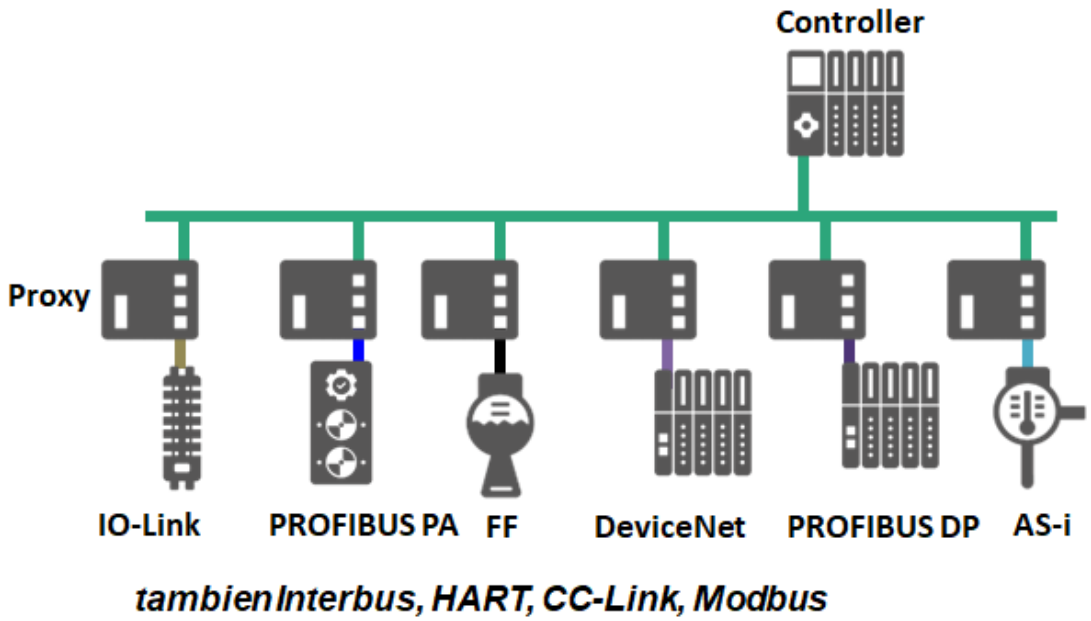
Las empresas buscan constantemente nuevas formas de mejorar la productividad, reducir los costos y aumentar la rentabilidad. PROFINET provee inicio rápido debido a ciertos factores como el tiempo necesario para configurar la red, el uso óptimo del ancho de banda, así como el monitoreo, diagnóstico y recuperación automáticos. Esta tecnología permite configuraciones flexibles y escalables para soportar toda la infraestructura necesaria para sistemas industriales complejos, incluidos sensores, actuadores y otros dispositivos conectados a través de redes industriales. Además, PROFINET ofrece un alto grado de fiabilidad gracias al uso avanzado del protocolo Ethernet; esto le permite a las empresas utilizar un único cable para conectar todos los dispositivos o equipos involucrados en su red industrial. Adicionalmente, PROFINET tiene un reconocimiento universal entre fabricantes líderes en tecnología industrial debido a su estructura modular y alta capacidad escalable. Esta escalabilidad permite que se conecten una cantidad teóricamente ilimitada de dispositivos por cada red sin conflictos o pérdida de calidad o seguridad.

Integración con otros bus de campo

PROFINET es una tecnología de comunicación industrial que se puede utilizar para conectar dispositivos industriales a redes Ethernet. Esta tecnología se integra con otros buses de campo, de niveles jerárquicos inferiores como **Profibus o ASi, IOLink**, o del mismo nivel jerárquico como Modbus TCP, Ethernet/IP, para permitir la comunicación entre los dispositivos y otros equipos en la red. Los dispositivos conectados vía PROFINET pueden ser controladores lógicos programables (PLCs), sensores, actuadores, tarjetas de interfaz, tarjetas de procesamiento digital de señal (DSPs) y tarjetas de adquisición de datos (DAQs). Las características clave del protocolo PROFINET incluyen altas tasas de transferencia de datos, soporte para cableado multipunto y soporte avanzado para tareas tales como el diagnóstico remoto. La comunicación entre dispositivos PLC a través de otros

buses requiere un adaptador que convierte los mensajes entre las dos redes. Este adaptador actúa como intermediario entre los dos buses diferentes, se conoce como “proxy” y está incorporado en el estándar PROFINET, de manera de facilitar su puesta en servicio.

El proxy recibirá mensajes enviados por los dispositivos del bus Profibus por ejemplo y los convertirá en mensajes PROFINET. Por el contrario, cuando hay un mensaje destinado a los dispositivos Profibus, el proxy lo convertirá en formato utilizable por el equipo conectado al bus Profibus. Este proxy funciona como un dispositivo PROFINET mientras que asume la función de maestro Profibus para los dispositivos conectados aguas abajo. Esto facilita los procesos de actualizaciones de sistemas de control, lo cual será tratado en el siguiente apartado. Esta funcionalidad permite que los dispositivos puedan integrarse totalmente en redes PROFINET sin necesidad de cambiar ninguno de sus componentes o configuraciones.



Comparativa de PROFINET vs. PROFIBUS

La adopción de tecnologías que nos permitan implementar soluciones actuales (I4.0, IIoT, etc) hace que la mayoría de las aplicaciones deba resolverse con PROFINET en lugar de Profibus. Para repasar, Profibus fue desde su comienzo allá por el año 1989 hasta entrado el año 2000 la solución típica para buses de campo industrial. La irrupción de PROFINET tuvo como paradigma la necesidad de dotar a los buses de campo de nuevas funcionalidades requeridas por la industria. Afortunadamente la adopción de PROFINET no hace necesario deshacerse de nuestras redes Profibus de la noche a la mañana, ya que a través del alto grado de integración se pueden llevar a cabo migraciones o actualizaciones tecnológicas de manera progresiva, cuidando las inversiones realizadas por los usuarios de estas redes. El concepto del uso de “proxys” estandarizados en PROFINET permite cumplir con estos escenarios. Para establecer una comparación entre ambas redes podemos mencionar que PROFINET transporta datos a una velocidad unos 15 veces más rápidos que Profibus, con un

tamaño de paquetes de datos mayor, permitiendo un número ilimitado de dispositivos unas 6 veces conviviendo en una misma red física. Otra característica de PROFINET es que permite la adopción de diferentes tipos de topologías de red de manera combinada y la posibilidad de contar con alta disponibilidad (redundancia) para aplicaciones críticas es mucho más simple de implementar.
En término de la detección de errores, PROFINET presenta una capacidad mayor gracias al uso de protocolos estándar que le permite entre otras cosas un manejo muy transparente de la topología de la red, mayor granularidad y posibilidad de identificación del tipo de error

	PROFIBUS	PROFINET
Organization	PI	
Application Profiles	same	
Concepts	Engineering, GSDs	
Physical layer	RS-485	Ethernet
Speed	12Mbit/s	1Gbit/s or 100Mbit/s
Telegram	244 bytes	1440 bytes (cyclic)**
Address space	126	unlimited
Technology	master/slave	provider/consumer
Connectivity	PA + others*	many buses
Wireless	Possible*	IEEE 802.11, 15.1
Motion	32 axes	>150 axes
M2M	No	Yes
Vertical integration	No	Yes
No. of products	thousands	thousands

6) **¿Qué es un protocolo CANopen?, ¿Para qué se usa? Ejemplifique**

CANopen es un protocolo de comunicación basado en CAN (Controller Area Network).; fue diseñado originalmente para sistemas de control de máquinas orientados al movimiento como sistemas de automatización industrial, vehículos y aplicaciones médicas.
No solo cómo se transmiten los datos, sino también cómo deben estructurarse, gestionar.
CANopen es una "capa superior" que organiza y da sentido a los mensajes que viajan sobre CAN.

Donde se usan

Se usa para permitir que múltiples dispositivos (motores, sensores, controladores, etc.) se comuniquen entre sí de manera estándar y organizada, sin necesidad de programación específica para cada caso.

por ejemplo:

- Controlar un motor desde un PLC.
- Leer sensores de temperatura, presión, o posición.
- Coordinar actuadores (válvulas, bombas) en tiempo real.
- Supervisar dispositivos médicos como bombas de infusión.

7) ¿Qué es un protocolo PROFIBUS-DP/PA?, ¿Para qué se usa? Ejemplifique

PROFIBUS (PROcess Field BUS) es un protocolo de comunicación industrial estandarizado, que se encarga de la comunicación entre los sensores de campo y el sistema de control o los controladores (como PLC) en sistemas de automatización**.**
Existen dos tipos:

- **PROFIBUS-DP (Decentralized Peripherals):** Solucion de alta velocidad,diseñado para comunicación rápida y eficiente entre un controlador y dispositivos distribuidos (por ejemplo, módulos de E/S, variadores de velocidad).

- **ROFIBUS-PA (Process Automation):** Adaptado para la automatización de procesos industriales (plantas químicas, petroleras), donde los sistemas de automatización y los sistemas de control de procesos se conectan con los equipos de campo, como transmisores de presión y temperatura, convertidores, posicionadores, etc. Ademas ventajas económicas propias de las instalaciones como reducción de costos y de mantenimiento; menor tiempo de puesta en marcha, logrando un gran aumento de la funcionalidad y la seguridad.

Donde se usan

Lo podemos ver en:

- **Conectar sensores y actuadores a** un PLC o sistema de control de manera eficiente.
- **Reducir el cableado,** porque todos los dispositivos comparten el mismo bus de datos.
- **Monitorear y controlar** procesos industriales en tiempo real.
- **Transmitir energía y datos** en entornos peligrosos (en el caso de PROFIBUS-PA).

Ejemplo:

Planta de tratamiento de agua:

- Un PLC supervisa bombas, válvulas, medidores de flujo y sensores de nivel.
- Los sensores de flujo y nivel están conectados mediante **PROFIBUS-PA** (porque también requieren energía en zonas seguras o peligrosas).
- Las bombas y válvulas de operación rápida están conectadas mediante **PROFIBUS-DP** para garantizar comunicación veloz y confiable.
- Todo se conecta en red usando un solo bus, lo que simplifica la instalación y el mantenimiento