ChatGPT

Guía de comunicación Ethernet con el reconectador NOJA OSM15-16-800

El reconectador **NOJA Power OSM15-16-800** (junto con su unidad de control RC-10 o RC-15) permite monitorear datos a través de Ethernet usando protocolos industriales estándar. A continuación se detalla cómo establecer comunicación Ethernet con este equipo, qué protocolos soporta, la configuración necesaria, cómo integrarlo con una aplicación Node.js para leer datos, y consideraciones de seguridad.

1. Protocolos de comunicación soportados

El control **RC-10/RC-15** del OSM15 soporta múltiples protocolos de comunicación sobre Ethernet para SCADA/telemetría:

Protocolo	**Puerto TCP** (típico)	**Descripción y uso**
DNP3 (IEEE 1815)	20000 (por defecto)	Protocolo SCADA orientado a objetos (entradas binarias/analógicas, eventos). El OSM soporta DNP3 incluyendo autenticación segura DNP3-SA [1] [2]. Permite tanto operación por sondeo como *unsolicited responses* (reportes espontáneos).
IEC 60870-5-104	2404 (estándar)	Protocolo SCADA europeo basado en IEC 60870-5-101 pero sobre TCP/IP. Soporta múltiples maestros conectados simultáneamente [3]. Útil para telecontrol en sistemas de distribución.
IEC 61850 (MMS/ GOOSE)	102 (MMS estándar)	Protocolo de automatización de subestaciones basado en objetos (IED). El RC implementa servidor **MMS** (Manufacturing Message Specification) y mensajería **GOOSE** (publicador/suscriptor) [4]. Requiere configuración de modelos de datos (archivo ICD/CID) acorde a la norma.
Modbus TCP/ IP	502 (por defecto)	Protocolo sencillo de tipo request/response. Implementado en firmware recientes del RC-10/15 [5] para integración con sistemas Modbus legados. Los datos (bits y registros) se mapean a direcciones Modbus estándar.
Otros	N/A	**Serial:** IEC 60870-5-101 (sobre RS232) también soportado. **Wi-Fi:** RC-15 ofrece Wi-Fi (modo punto de acceso o cliente) para conectar con la misma funcionalidad que Ethernet cableado. **Celular:** RC-15 incluye módem 3G/4G para comunicaciones remotas. **FTP:** el RC ofrece un servidor FTP de solo lectura (p.ej. descarga de oscilografías) [6]. No dispone de API HTTP/WebSocket nativa para monitoreo en tiempo real (se usan los protocolos anteriores).

Nota: La controladora RC viene con interfaz **Ethernet 10/100 Base-T RJ45** estándar [7]. En el RC-10 clásico esta es la principal vía de red; el **RC-15** adicionalmente incorpora Wi-Fi y módem 4G interno,

pero la comunicación cableada Ethernet sigue siendo vía RJ45. Los protocolos DNP3, IEC 104, IEC 61850 y Modbus son opcionales y pueden activarse según las necesidades de integración. Por ejemplo, el RC soporta DNP3 (incluyendo múltiples maestros simultáneos) e IEC 61850 de fábrica [1], mientras que **Modbus TCP** se añadió a partir de ciertas versiones de firmware [5]. No hay un servidor web HTTP integrado en el equipo, por lo que el monitoreo debe realizarse vía los protocolos industriales mencionados (o mediante el software propietario de NOJA si se utiliza).

2. Requisitos previos y configuración del equipo

Antes de que una aplicación JavaScript pueda comunicarse con el OSM15 por Ethernet, se deben cumplir algunos pasos de configuración:

1. **Instalación del** RC **y conexión física:** Asegúrese de contar con el **cubículo de control RC-10 o RC-15** conectado al reconectador. El RC contiene el relé de control/protección con el puerto Ethernet y demás interfaces de comunicación [8]. Conecte un cable Ethernet al puerto RJ45 del RC. Si conecta directamente un PC al RC, use un cable **Ethernet cruzado** o asegúrese de que la tarjeta de red soporte auto-MDIX (la mayoría lo hace). Si conecta el RC a una red LAN, use un cable directo a un switch de la red de control.

2. **Configuración de dirección IP:** Encienda el cubículo de control y configure la interfaz Ethernet con una dirección IP adecuada para su red. Por defecto, el RC puede venir configurado para obtener IP por **DHCP** o con una IP fija de fábrica (consulte el manual del equipo). Es recomendable asignarle una IP estática acorde a su esquema de red (por ejemplo, `192.168.1.100/24`). Esta configuración se realiza a través del panel frontal del RC (menú **Network** o **Comms**) o mediante el software de configuración **CMS** de NOJA Power. Asegúrese de configurar también máscara de subred y gateway si se requiere acceso remoto. *(En el panel HMI del RC15 se puede encontrar estas opciones bajo menús de red, incluyendo soporte IPv6 si fuese necesario.)*

3. **Habilitar el protocolo deseado:** Por seguridad, muchos protocolos están **deshabilitados por defecto**. Mediante el menú del panel del RC o con la aplicación CMS/SGA en PC, vaya a **System Settings → Protocol Settings** y habilite el protocolo que va a utilizar. Por ejemplo, para usar DNP3 sobre Ethernet, active **DNP3: Enabled** y configure "Port: LAN" y "Connection Type: LAN" [9] [10]. Análogamente, para IEC 60870-5-104 habilite **IEC 60870-5-101/104: Enabled**. Si va a usar Modbus TCP, habilítelo en la configuración (en firmwares recientes aparece la opción Modbus en Protocol Settings). Usualmente, el equipo requiere un **reinicio** del controlador después de activar un protocolo para que comience a operar [11].

4. **Parámetros del protocolo:** Configure los parámetros específicos:

5. *DNP3:* Asigne la **Dirección de esclavo (Outstation)** adecuada (por ejemplo, 5 por defecto [12]) y la **dirección de maestro** si se utiliza DNP3 *secure authentication*. El puerto TCP por defecto para DNP3 es **20000** [13]. El OSM15 permite **dos maestros DNP3 simultáneos** (modo multi-master) si se habilita, ya sea diferenciando por dirección fuente IP/puerto (Método 1) o usando puertos TCP separados (Método 2, p.ej. 20000 y 20001) [14] [15]. Para la mayoría de casos, con un solo maestro, se usa un único puerto (20000).
6. *IEC 60870-5-104:* Configure la **Dirección común ASDU** (ej. 5 por defecto) y la **dirección de enlace** si aplica. El puerto por defecto es **2404** (el RC10/15 típicamente usa 2404 o 2405 como predeterminado) [16]. Este protocolo también soporta múltiples conexiones maestros en paralelo.

7. *IEC 61850:* Cargar el archivo de configuración ICD/CID si su sistema lo requiere, y establecer la IP (soporta IPv4/IPv6). El servidor MMS normalmente escucha en el puerto estándar **102**. Configure el **Nombre IED** y habilite las publicaciones GOOSE según lo necesite [4]. (Esta configuración suele ser más avanzada y se realiza con herramientas especializadas o la interfaz del CMS para 61850).

8. *Modbus TCP:* Verifique que el **ID de unidad (Unit ID)** Modbus del RC esté configurado (por defecto suele ser 1). El **puerto TCP 502** es el estándar para Modbus [17], y normalmente no requiere cambios. NOJA recomienda cargar/confirmar el *archivo de puntos Modbus* en el controlador mediante el software (SGA/CMS) para asegurarse de que el mapa de registros esté presente y actualizado [18]. En muchos casos, Modbus ya estará listo con un mapa por defecto si el firmware lo incluye, pero es bueno confirmar.

9. **Verificación:** Compruebe en el panel frontal o en el software CMS que el protocolo esté **"Ready"** o activo. Por ejemplo, en el menú del RC verá el **estatus** de DNP3 o IEC 104 como "Ready" cuando están activos [9]. También puede probar haciendo un *ping* a la IP del equipo para verificar la conexión física. Si todo es correcto, el reconectador está listo para ser consultado por la aplicación.

¿Software CMS/SGA adicional? El **CMS** (Control and Management Software) de NOJA Power es la herramienta oficial para configuración y monitoreo engineering del reconectador. No es estrictamente necesario tenerlo en ejecución para que su aplicación JavaScript se comunique, pero **sí es muy útil para la configuración inicial** (IP, activar protocolos, etc.) y para obtener la documentación de los puntos de datos (p. ej. la lista de puntos DNP3 o mapa Modbus). En resumen, se usa CMS/SGA para configurar, pero su propia app JS puede conectarse directamente al equipo una vez configurado, **sin necesidad de software intermedio propietario**. La presencia del **controlador RC10/RC15** es imprescindible (es parte del equipo, no un software), pues alberga la lógica de comunicaciones; asegúrese de que esté alimentado y en modo **Remoto** (Remote) para permitir lecturas remotas. No se requieren controladores especiales ni SDK propietario: los protocolos son estándar, por lo que su app puede comunicarse mediante sockets TCP/IP normales.

3. Comunicación desde una aplicación JavaScript (Node.js)

Una vez el OSM15 está configurado en la red, una aplicación Node.js puede actuar como **cliente/master** de los protocolos para consultar datos:

- **Modelo de comunicación:** El reconectador (RC) funciona como **servidor** o **esclavo** en estos protocolos, esperando conexiones. Su app Node.js se comportará como **cliente/maestro** que inicia la conexión TCP/IP al IP/puerto del dispositivo y envía solicitudes de lectura. Por ejemplo, para DNP3 su app sería el *Master* DNP3, y para Modbus TCP sería el *cliente Modbus*.

- **Bibliotecas de Node.js recomendadas:** Existen librerías en npm que implementan estos protocolos, facilitando el envío y recepción de tramas sin programarlas a bajo nivel:

- *DNP3:* Es un protocolo más complejo y no tan común en entornos Node, pero hay proyectos en desarrollo. Por ejemplo, la librería `dnp3` de IvanGaravito/Klaus Landsdorf (npm *dnp3*) busca implementar un stack DNP3 en JavaScript [19]. Sin embargo, estas implementaciones pueden estar en etapas tempranas (¡algunas advierten *"NOT READY TO USE"* [20]!). Otra opción es usar **Node-RED** con el nodo `node-red-contrib-dnp3` que encapsula funcionalidad DNP3 [21], o

integrar una biblioteca nativa (C/C++) mediante add-ons si se requiere toda la robustez (por ejemplo, usando **OpenDNP3** a través de un binding).

- *Modbus TCP:* Hay múltiples librerías maduras y fáciles de usar. Por ejemplo `jsmodbus` [22] implementa cliente/servidor Modbus TCP con soporte de funciones 1-6,15,16. Otra muy utilizada es `modbus-serial`, que pese a su nombre soporta también conexión TCP/IP [23]. Estas librerías manejan las peticiones Modbus (funciones de lectura/escritura de registros) y retornan los datos en formatos amigables.
- *IEC 60870-5-104:* No tan populares en npm, pero existen paquetes de la comunidad (ej. `iec101` / `iec104` en Node) o soluciones vía Node-RED. En caso necesario, podrían usarse módulos de Python/Java puenteados, dado que Node puro tiene menos soporte aquí. Para un proyecto centrado en JavaScript, suele preferirse DNP3 o Modbus salvo que IEC 104 sea un requisito de la infraestructura.

- *IEC 61850:* Implementar un cliente MMS/GOOSE en Node desde cero es muy complejo. Normalmente se recurre a librerías especializadas en otros lenguajes o middleware de automatización. No es habitual hacerlo directamente en JS. Si se necesitara, se podría usar una API REST intermedia ofrecida por un gateway 61850, o emplear un módulo nativo compilado que integre una librería 61850 (similar a cómo se abordaría DNP3).

- **Estructura de mensajes y acceso a datos:** Cada protocolo tiene su forma de organizar los datos:

- En **DNP3**, el RC expone puntos **Binary Inputs**, **Analog Inputs**, **Binary Outputs** etc. según su *Device Profile*. Por ejemplo, estados de interruptor (abierto/cerrado) serían Binary Inputs, medidas de voltaje/corriente son Analog Inputs, contadores como operaciones o disparos podrían ser Analog Inputs o Counter objects. Su aplicación DNP3 Master deberá saber los **índices** de estos puntos (definidos en el documento de perfil DNP3 de NOJA [24]) para leerlos. La comunicación DNP3 es orientada a objetos: se pueden leer grupos enteros o puntos específicos, y recibir eventos (unsolicited) si están habilitados. Por simplicidad, inicialmente su app podría enviar lecturas globales (p.ej. función READ de clases 0/1/2/3) para obtener todos los datos, luego filtrar lo necesario.
- En **Modbus**, el RC presenta los datos en direcciones tradicionales: **coils (0x)** para salidas/discretos, **input status (1x)** para entradas digitales, **input registers (3x)** para valores analógicos de solo lectura, y **holding registers (4x)** para valores configurables o comandos. NOJA proporciona una lista de registros Modbus (mapa de memoria) en su *Modbus User Guide*. Por ejemplo, es común que las mediciones de tensión, corriente, potencia, etc. estén en registros **3xxxx** (Input Registers) escalados (p.ej. tensión fase A en 30001, corriente fase A en 30003, etc.), y estados lógicos en bits **1xxxx** o **coil**. Un ejemplo del mapa Modbus muestra registros 30072+ para contadores de operaciones (CO total, desgaste, etc.) [25] [26]. Su aplicación debe leer las direcciones correspondientes usando la función Modbus adecuada (p.ej. FC4 para input registers). La estructura de trama Modbus es sencilla: su librería se encargará de enviar la consulta (dirección inicial y cantidad) y decodificar la respuesta en un buffer de datos. Usted solo deberá indicar qué dirección leer y cómo interpretar el resultado (aplicar escalas, signos, etc., según el manual: por ejemplo, un registro puede requerir multiplicar por 0.001 para obtener kV [27]).
- En **IEC 104**, la comunicación también es orientada a objetos (ASDU) con tipos de información (measured value, single point, etc.) y causa de transmisión. Al igual que DNP3, la app debería usar una librería que se encargue del enlace ACSE y cotas. Dado que es menos trivial encontrar librería, se omite detalle aquí, pero conceptualmente leer un punto en IEC 104 implica conocer su tipo y dirección (identificador de objeto).

- En **IEC 61850**, los datos se organizan en un modelo de objetos jerárquico (Logical Nodes, Data Objects). Normalmente se usaría un toolkit especializado; excede el alcance de esta guía implementarlo directamente en JS.

En la práctica, para una aplicación JavaScript ligera, **Modbus TCP** suele ser la vía más directa para monitorear datos básicos, siempre que el mapa de registros esté disponible, ya que con pocas líneas puede establecerse la conexión y leer registros. Si se requiere **telemetría robusta con eventos** y seguridad, **DNP3** sería preferible, pero implicará más trabajo de integración (o usar un componente externo).

A continuación, presentamos un ejemplo simple usando Node.js con Modbus TCP para ilustrar la conexión y lectura de datos.

4. Ejemplo de código en Node.js para lectura de datos

Supongamos que el reconectador tiene IP `192.168.1.100` y queremos leer algunos valores de medida (por ejemplo, voltajes de fase) expuestos en registros Modbus. Usaremos la librería `modbus-serial` en Node.js, que permite conectarse vía TCP. El siguiente código de ejemplo muestra cómo conectarse y leer registros:

```
// Importar la librería Modbus
const ModbusRTU = require("modbus-serial");
const client = new ModbusRTU();

// Conectar al equipo vía TCP (IP y puerto 502)
client.connectTCP("192.168.1.100", { port: 502 })
  .then(() => {
    console.log("Conexión TCP establecida");
    // Fijar la ID de unidad Modbus (slave ID) si es necesaria. Por defecto
suele ser 1.
    client.setID(1);
    // Leer 6 registros de entrada (Input Registers) comenzando desde la
dirección 0.
    // (Esto típicamente corresponde a los registros 30001-30006 en notación
Modbus)
    return client.readInputRegisters(0, 6);
  })
  .then(response => {
    // `response.data` es un array de valores de 16 bits leído del equipo.
    console.log("Registros leídos:", response.data);
    // Por ejemplo, response.data[0] podría ser la tensión de fase A en
decenas de voltios.
    const voltajeFaseA = response.data[0] * 0.1;  // Ejemplo: aplicar factor
de escala de 0.1
    console.log("Voltaje Fase A:", voltajeFaseA, "V");
  })
  .catch(err => {
    console.error("Error en comunicación Modbus:", err);
  });
```

Explicación: En el código: - Se utiliza `client.connectTCP(host, {port: p})` para establecer la conexión al IP/puerto del RC (502 para Modbus). Esto retorna una *Promise* que se resuelve cuando hay conexión. - Luego `client.setID(1)` configura la dirección de dispositivo Modbus (la mayoría de equipos usan ID 1 por defecto; debe coincidir con la configurada en el RC). - `client.readInputRegisters(0, 6)` envía una solicitud de lectura de 6 registros a partir de la dirección 0 (es decir, registros 30001–30006). La promesa resultante entrega un objeto con la propiedad `data` que contiene una matriz de valores de 16 bits. - En el ejemplo, simplemente imprimimos los valores. Para interpretarlos correctamente, habría que consultar el manual de NOJA sobre qué representa cada registro y qué factor de escala aplicar. En la línea donde calculamos `voltajeFaseA`, asumimos por ejemplo que el valor devuelto necesita multiplicarse por 0.1 para obtener volts reales – esto es solo ilustrativo. - Se incluye un manejo de errores para capturar problemas de conexión o timeouts (por ejemplo, si el equipo no responde en el tiempo esperado, la librería lanzará un error).

Este código se ejecutaría en Node.js (por ejemplo `node app.js`) y debería mostrar por consola los datos leídos. En una aplicación real, en lugar de solo imprimir, probablemente se actualizaría una interfaz web, base de datos o dashboard con estos valores.

Si quisiéramos usar **DNP3** en lugar de Modbus, el código sería más complejo. Habría que utilizar una librería DNP3 para Node (o vía Node-RED). Por ejemplo, con una hipotética librería `dnp3`, uno tendría que establecer un **TCP client** a la IP del RC en el puerto 20000, y luego solicitar los objetos DNP3 deseados. Un pseudo-ejemplo simplificado podría ser:

```
// Pseudocódigo ilustrativo para DNP3 (no ejecutable sin una librería
adecuada)
const dnp3 = require("dnp3");  // Asumiendo existe un módulo DNP3
const client = new dnp3.DNP3Master("192.168.1.100", 20000, { slaveId:
5 }); // conectar al outstation 5
client.on("open", () => {
  // Leer punto DNP3 (ej: Binary Input index 1 = estado del interruptor)
  client.readBinaryInput(1, (err, value) => {
    console.log("Estado del interruptor (BI1):", value);
  });
});
```

Lo anterior es solo conceptual: en la práctica habría que manejar buffers y callbacks propios de la librería. Por esta razón, muchos desarrolladores optan por Modbus o por usar software intermedio que traduzca DNP3 a algo más manejable. No obstante, con las herramientas correctas, es posible integrar DNP3 directamente si se requiere un mayor nivel de detalle en la monitorización.

5. Consideraciones de seguridad y validación de datos

Al conectar un equipo de protección de red eléctrica a una aplicación, es fundamental tener en cuenta aspectos de **seguridad** y **fiabilidad de datos**:

- **Autenticación y control de acceso:** De ser posible, active las funciones de autenticación en el protocolo. El RC soporta **DNP3 Secure Authentication (SA)** versiones 2 y 5 [2], lo cual ayuda a prevenir comandos no autorizados mediante intercambio de claves y desafío/respuesta. Si decide implementar DNP3-SA, deberá generar una clave de actualización con el CMS y cargarla en el equipo (vía USB según manual) [28] [29]. Para **Modbus**, tenga en cuenta que **no posee**

autenticación ni cifrado por diseño – cualquier cliente que alcance el puerto 502 podría enviar comandos [30]. Por tanto, *restrinja el acceso a la red*: ubique el RC en una VLAN de control o detrás de un firewall/VPN si accede remotamente.

- **Roles de usuario y modo local/remoto:** El reconectador NOJA suele tener modos **Local/ Remote**; en local se bloquean comandos remotos. Asegúrese de que el equipo esté en *Remote* solo cuando se requiera control remoto. Para monitoreo (lectura) normalmente no hay restricción, pero los comandos de apertura/cierre sí estarán inhibidos si está en Local. Además, el CMS y el panel tienen contraseñas para cambios de configuración, pero las lecturas vía protocolos SCADA generalmente no requieren autenticación de usuario, por lo que proteger la capa de transporte (red) es clave.

- **Whitelisting de IP:** Utilice las características de filtrado que ofrece el RC. Por ejemplo, en la configuración DNP3 existe la opción **"Check Master IP Address"** para restringir que solo un IP maestro específico se comunique [31]. Ajuste esto a "Yes" e ingrese la IP de su servidor Node.js para que el RC ignore cualquier otro origen. En IEC 104 hay configuración similar de direcciones permitidas. Si el RC-15 está conectado por Wi-Fi, asegúrese de usar WPA2 y credenciales seguras en caso de AP.

- **Integridad de datos y validación:** Al recibir datos en su aplicación, **valide la consistencia**. Por ejemplo, en DNP3 cada punto viene con **flags de calidad** (online/offline, reemplazado, etc.); utilícelos para ignorar valores inválidos. En Modbus, verifique que la respuesta corresponda al pedido (las libs lo hacen internamente, pero si implementa algo personalizado, chequee el código de función y número de bytes). También considere rangos físicos: si recibe una tensión fuera de rango plausible, podría indicar un error de comunicación o un fallo de sensor. Implementar límites o alertas por valores anómalos mejorará la confiabilidad del monitoreo.

- **Seguridad activa (comandos):** Dado que su objetivo es monitoreo, probablemente solo leerá datos y no enviará comandos de control. Aun así, *configure el RC para prevenir acciones no deseadas*. En DNP3, puede dejar **Disabled** las clases de objetos de control si no las usará, y en Modbus evite funciones de escritura desde su app si no es necesario. En caso de que eventualmente también ejecute comandos de apertura/cierre desde la app, implemente doble verificación o confirmación manual para evitar operaciones accidentales.

- **Performance y timeout:** Considere la frecuencia de sondeo de datos. DNP3 e IEC 104 pueden enviar eventos espontáneos, reduciendo la necesidad de sondear constantemente. Modbus en cambio suele operarse por encuesta periódica; ajusta un intervalo que no sature al equipo (ej. leer cada 1 segundo un bloque de registros en vez de leer muy frecuentemente uno por uno). Maneje *timeouts* de comunicación – si el equipo no responde en X segundos, su aplicación debería detectar la pérdida de comunicación y quizá reintentar o notificar una alarma. Las librerías Node suelen permitir configurar estos timeouts.

- **Actualizaciones de firmware:** Mantenga el firmware del RC actualizado a la última versión estable de NOJA Power. Además de nuevas funciones (como Modbus que se incorporó en firmware v1.26 [32]), las actualizaciones suelen corregir vulnerabilidades y mejorar la robustez de comunicaciones. Consulte al soporte de NOJA si hay versiones más nuevas que incluyan parches de seguridad.

En resumen, mediante la configuración adecuada y usando los protocolos soportados (DNP3, IEC 104, IEC 61850 o Modbus TCP) puede integrar el reconectador NOJA OSM15-16-800 a su aplicación JavaScript

para monitoreo de datos. Comience habilitando y probando con un protocolo (Modbus es un buen inicio por su simplicidad), luego evolucione según necesite más funcionalidades (por ejemplo, DNP3 para recibir eventos de forma asíncrona o IEC 61850 si se integra a sistemas de automatización avanzados). Siempre documente las direcciones/índices de datos que lee, apoyándose en la guía de puntos del fabricante, e implemente las medidas de seguridad para asegurar una operación confiable. ¡Con esto, su aplicación podrá visualizar en tiempo real el estado y métricas del reconectador de forma efectiva y segura! [1] [5]

---

[1] [8] NOJA-581-08 NOJA Power OSM15-27-38 Guia de Producto - es - El Reconectador más OSM Seguro del mundo - Studocu
https://www.studocu.com/co/document/universidad-antonio-narino/electricidad-y-magnetismo/noja-581-08-noja-power-osm15-27-38-guia-de-producto-es/77409434

[2] [3] [4] [6] [7] [9] [10] [11] [12] [13] [14] [15] [16] [24] [28] [29] [31] [32] pesvs.com.au
https://pesvs.com.au/wp-content/uploads/2024/06/Noja-Power-RC10-Interface-Test-Set-ITS-04.pdf

[5] [18] [30] Modbus in Reclosers | NOJA Power - Recloser Switchgear Engineers
https://www.nojapower.com/expertise/2021/modbus-in-reclosers

[17] [25] [26] [27] NOJA-5074-00 Modbus User Guide | PDF | Computing | Computer Engineering
https://www.scribd.com/document/724376383/NOJA-5074-00-Modbus-User-Guide

[19] [20] GitHub - IvanGaravito/dnp3: DNP 3.0 library for Node.js
https://github.com/IvanGaravito/dnp3

[21] biancode - NPM
https://www.npmjs.com/~biancode

[22] GitHub - Cloud-Automation/node-modbus: Modbus TCP Client/Server implementation for Node.JS
https://github.com/Cloud-Automation/node-modbus

[23] modbus-serial - npm
https://www.npmjs.com/package/modbus-serial