Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Detección básica de anomalías en el protocolo DNP3

Autor: Jacinto Jurado Tabares
Tutor: Antonio Estepa Alonso

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Detección básica de anomalías en el protocolo DNP3

Autor:
Jacinto Jurado Tabares

Tutor:
Antonio Estepa Alonso
Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Detección básica de anomalías en el protocolo DNP3

Autor: Jacinto Jurado Tabares

Tutor: Antonio Estepa Alonso

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis profesores

A mis amigos

Agradecimientos

En primer lugar, quería agradecer a mi familia, a mis padres por darme la oportunidad de poder realizar estos estudios gracias a su esfuerzo, por formarme como persona y por apoyarme y animarme siempre en lo que me he propuesto, a mi hermana María que ha supuesto desde pequeño para mi formación académica un ejemplo a seguir y a mi hermana Laura por su trato especial conmigo desde pequeños.

En segundo lugar, agradecer a las personas con las que más tiempo he pasado durante esta etapa y que siempre han supuesto un apoyo para mí, Manolo, Celia, Pablo, Fernando, Ángel, Vicente, Julio, Rubio, Juanjo y todas esas personas que han estado en mi vida durante estos 4 años en Sevilla y que han ayudado a que esta experiencia haya sido única. Sin vosotros esto nunca hubiera sido lo mismo y nunca lo será.

En tercer lugar, agradecer a mis compañeros de piso José María y Jorge, por ser las personas que han convivido conmigo a diario durante los últimos 4 y 3 años respectivamente, por todo su apoyo durante estos años y por haber pasado de ser dos desconocidos que vivían conmigo a dos amigos para toda la vida.

Gracias también a mis amigos de Córdoba que han supuesto un apoyo continuo desde la lejanía y que a pesar de la poca frecuencia con la que nos vemos nunca se han olvidado de mí.

Por último y no menos importante, agradecer a todos los profesores por su formación impartida y, en especial a mi tutor para este trabajo, Antonio Estepa, por ayudarme y permitirme llevar a cabo este trabajo y agradecer también a Agustín Walabonso por su gran ayuda en la parte final técnica del trabajo.

Jacinto Jurado Tabares

Sevilla, 2021

Resumen

El objetivo de este trabajo es la investigación y desarrollo de técnicas de detección de anomalías de la capa de aplicación en el tráfico de las redes de ambiente industrial, más concretamente en la detección de anomalías en el protocolo de aplicación DNP3.

Se ha diseñado para ello un sistema que consta de un disector de paquetes del protocolo DNP3 integrado en la herramienta tranalyzer. A continuación, la salida del disector es procesada para identificar las variables de interés y almacenarlas en una base de datos. Las series temporales de las variables almacenadas en la base de datos constituyen el tráfico de entrada de un detector de anomalías basado en EWMA.

Se han realizado pruebas de detección tanto con tráfico limpio como con tráfico contaminado. Los resultados obtenidos indican que el sistema es capaz de detectar los verdaderos positivos en el 100% de los casos y detecta falsos positivos en un 15% de las muestras.

Abstract

The objective of this document is to research and develop techniques to identify anomalies of the application layer in the industrial networks traffic, more specifically in identifying anomalies in the DNP3 application layer protocol.

A system has been designed that consists of a packet dissector of the DNP3 protocol integrated in the tranalyzer tool. Following, the dissector output is processed to identify the useful variables and store them in a database. The time series of the variables stored in the database make the input traffic of an EWMA-based anomaly detector.

Screening tests have been carried out with clean and poisoned traffic. The results reached point out that the system can detect true positives in 100% of the tests and detect false positives in the 15% of the samples.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En este capítulo se va a realizar una introducción a las redes ICS y a la seguridad del protocolo DNP3.

1.1 Contexto

En un mundo en el que la interconexión y la automatización de procesos cada vez gana más importancia, surge el problema de diseñar una red para que las máquinas de una fábrica puedan intercambiar datos de manera fiable, segura y rápida entre ellas, para que les permita controlarse e informar sobre su estado u operaciones.

En este contexto surgen las redes de sistemas de control industrial (ICS por sus siglas en inglés) que se definen como un conjunto de equipos interconectados entre sí utilizados para supervisar y controlar equipos dentro de un ambiente industrial [1]. Los elementos principales que componen una red ICS son:

- Controlador Lógico Programable (PLC): son dispositivos electrónicos especializados que generalmente constan de una fuente de alimentación, un procesador, un módulo de entrada / salida y un módulo de comunicación. Forman el núcleo de las redes de control industrial. También son conocidos como PC (que no debe confundirse con el anglicismo de ordenador) [1] [23].
- Sistema de Control de Supervisión y Adquisición de Datos (SCADA): un sistema SCADA es una capa de software, normalmente aplicada un nivel por encima del hardware de control dentro de la jerarquía de una red industrial. Como tal, los sistemas SCADA no realizan ningún control, sino que funcionan de manera supervisora. El objetivo de un SCADA es la adquisición de datos y la presentación de una interfaz hombre-máquina (HMI) centralizada, aunque también permiten que se envíen comandos de alto nivel al hardware de control, por ejemplo, la instrucción para arrancar un motor o cambiar un punto de ajuste. El hardware de control que se comunica con un SCADA se denomina Unidad Terminal Remota (RTU) y suele ser un tipo de PLC especializado. El dispositivo con el que se comunica la RTU se conoce como Unidad Terminal Maestra (MTU) [1] [23].
- Sistema de Control Distribuido (DSC): un DCS se asemeja a un SCADA, ya que es un software que realiza la comunicación con el hardware de control y presenta una HMI centralizada para el equipo de control. La diferencia entre los dos es sutil, especialmente con los avances tecnológicos que permiten que la funcionalidad de cada uno se superponga. La diferencia clave entre los dos es que los DCS son impulsados por procesos en lugar de eventos y generalmente se enfocan en presentar un flujo constante de información de procesos. Esto significa que, aunque los dos sistemas parecen similares, su funcionamiento interno es bastante diferente [1] [23].

Estas redes se diferencian significativamente de las redes convencionales por sus requisitos específicos de funcionamiento, reflejados en la Tabla 1-1:

Características	Redes ICS	Redes convencionales
Función principal	Control de los equipos físicos	Procesamiento y envío de datos
Dominio de aplicación	Fabricación, procesamiento y distribución de servicios públicos	Entornos de oficinas y hogares
Jerarquía	Profunda y funcionalmente separadas con muchos protocolos y estándares físicos	Superficial y con protocolos y estándares físicos únicos
Gravedad ante errores	Muy alta	Baja
Fiabilidad requerida	Muy alta	Media
Retardo	250 µs- 10 ms	+50 ms
Composición de los datos	Tráfico de paquetes pequeños periódicos o repentinos	Tráfico de paquetes grandes y no periódicos
Consistencia temporal	Requerida	No requerida
Entorno de operación	Polvo en el ambiente, calor y vibraciones	Entornos libres de ruido e interferencias

Tabla 1-1: Comparativa de las características de las redes ICS y las convencionales [1].

Las redes ICS son mucho más estrictas respecto a retardos, fiabilidad y gravedad ante errores que las redes convencionales, además de que operan en ambientes mucho más hostiles. Si nos paramos a pensarlo, es lógico, por ejemplo, si estamos en nuestra casa u oficina y enviamos un mensaje a alguien, no es importante que ese mensaje se pierda y haya que volver a enviarlo, pero si en una fábrica de lavadoras la máquina que atornilla el tambor avisa que no ha podido realizar bien su trabajo y ese mensaje se pierde, ese error puede provocar incluso daños personales.

A pesar de las diferencias funcionales entre las redes ICS y las redes convencionales hay una tendencia de integración de ambas usando Ethernet/IP. Históricamente las redes ICS han sido reticentes a adoptar el uso de Ethernet/IP por su poca fiabilidad y seguridad, pero debido a la gran reducción de costes que supone pasarse a una red Ethernet/IP y al avance en materia de seguridad y fiabilidad cada vez más se opta por esta opción.

Muchos de los ICS actuales han evolucionado gracias a la evolución de los sistemas físicos existentes, ya que muchos de estos sistemas físicos han sido sustituidos por controles digitales integrados. Las mejoras en el costo y el rendimiento han alentado esta evolución, lo que ha dado lugar a muchas de las tecnologías "inteligentes" actuales, como la red eléctrica inteligente o la fabricación inteligente. Si bien esto aumenta la conectividad de estos sistemas, también crea una mayor necesidad de su adaptabilidad, resistencia, protección y principalmente de seguridad [23].

La ingeniería de las ICS continúa evolucionando para proporcionar nuevas capacidades además de para abordar estos problemas emergentes. A continuación, vamos a analizar los problemas de seguridad de las ICS actuales

1.2 Problemas de seguridad en las redes ICS

Las redes de sistemas de control industrial (ICS) desempeñan un papel importante en la industria actual al proporcionar automatización de procesos, control distribuido y supervisión de procesos. Pero debemos tener en cuenta que las ICS inicialmente fueron diseñadas para ser utilizadas en un área aislada o conectado a otros sistemas a través de mecanismos o protocolos de comunicación especializados, lo que permite a los fabricantes gestionar sus procesos de producción con gran flexibilidad y seguridad.

Sin embargo, este diseño no cumple con los requisitos comerciales actuales para trabajar con tecnologías actuales como el Internet de las cosas (IoT) o el análisis de grandes volúmenes de datos (big data). Para

cumplir con los requisitos empresariales, muchos ICS se han conectado a las redes empresariales (redes que monta la empresa en sus oficinas) permitiendo a los trabajadores acceder a datos en tiempo real generados por las plantas de producción. Al mismo tiempo, este nuevo diseño abre varios desafíos de ciberseguridad para las ICS [2].

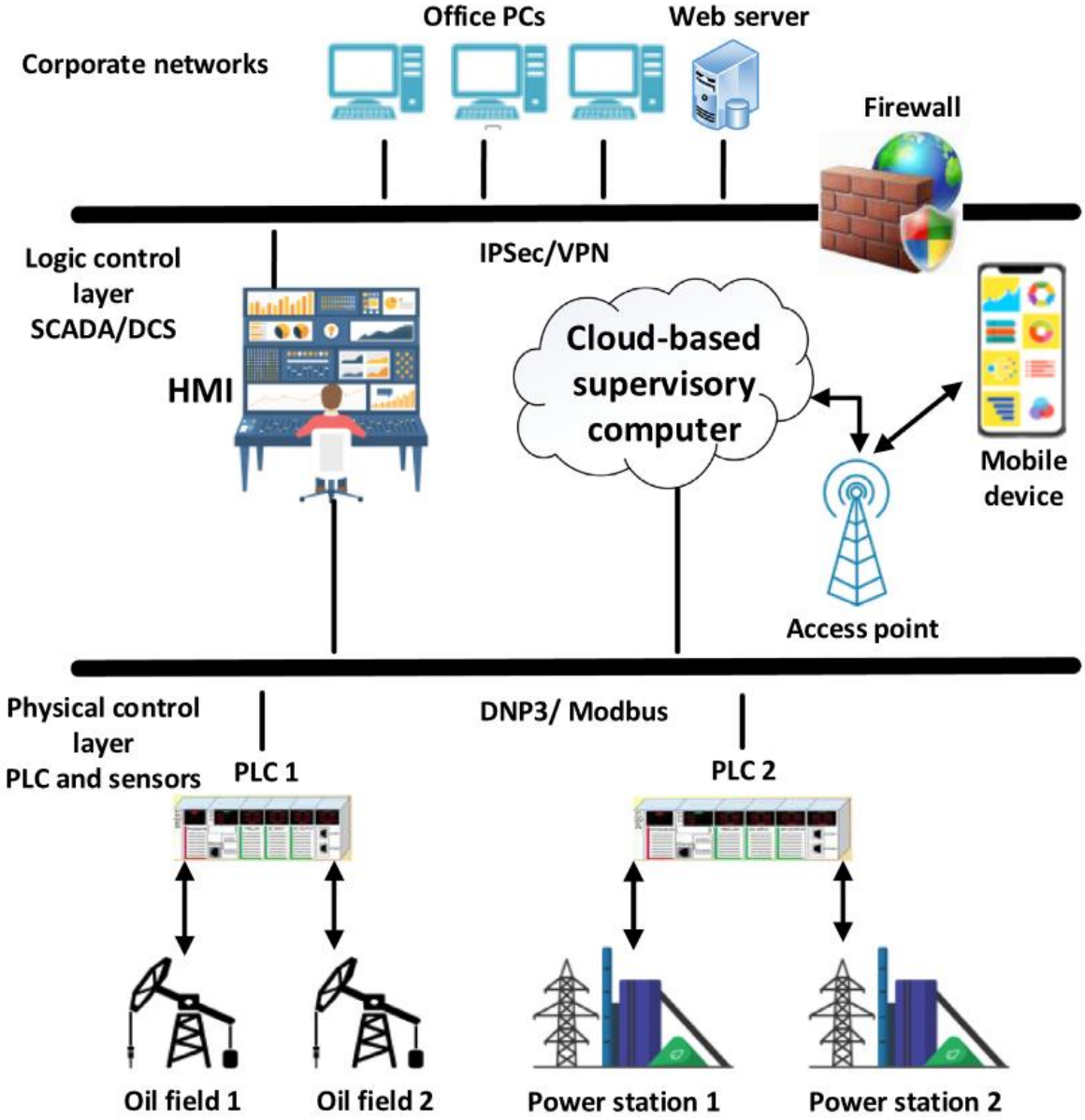

Figura 1-1: Red ICS integrada con red corporativa [2].

Estos desafíos de seguridad vienen generados de 3 fuentes diferentes. Imaginemos que disponemos de una red ICS conectada a la red corporativa de la empresa como la de la Figura 1-1. Podemos diferenciar 3 puntos donde la red puede ser vulnerable, que son:

- Vulnerabilidades en la red corporativa de la empresa: pueden entrar virus en los ordenadores de la oficina por un mal uso de ellos y pueden sufrir ataques de ingeniería social como "phishing" o "rogue access point" entre otros.
- Vulnerabilidades en los protocolos de comunicación entre elementos SCADA y PLC: fallos en la configuración de los dispositivos, arquitectura de seguridad débil (no tener "firewalls" en funcionamiento) y debilidades del software (ataques como "buffer overflow" o "deny of service").

- Vulnerabilidades en el entorno de producción: debilidades en los protocolos de comunicación (DNP3 o Modbus) y debilidades del hardware (acceso remoto inseguro a los componentes del ICS, protección física inadecuada de los sistemas importantes y personal no autorizado con acceso físico al equipo).

Sufrir ataques en un entorno industrial puede tener consecuencias muy graves e intolerables, por ejemplo, en 2015 la mitad de la población de Ucrania se quedó sin electricidad debido a un ciberataque contra la planta eléctrica de la empresa Prykarpattyaoblenergo [2].

Dentro de todos estos factores que comprometen la seguridad de la red, nos vamos a centrar en los ataques y debilidades de los protocolos de comunicación entre los SCADA y PLC, más específicamente los que sufre el protocolo DNP3.

1.2.1 Problemas de seguridad en DNP3

DNP3 es un protocolo desarrollado a principios de los años 90, cuando el entorno de las ICS era meramente industrial y no se concebía la idea de que los datos de este protocolo circularan por Internet, por lo cual, no está diseñado pensando en la seguridad. Esto genera una serie de carencias en la estructura del protocolo que vamos a analizar [3] [4].

Los ataques que sufre DNP3 se clasifican en función del objetivo atacado (el centro de control, las estaciones y las rutas que se crean en la red) o en función de lo que se hace con los paquetes atacados (intercepción, interrupción, modificación o creación). Según el estudio [4] se han detectado 28 ataques posibles que puede sufrir el protocolo, de los cuales vamos a desarrollar los 15 más importantes. Vamos a clasificar los ataques en función de la capa que se ve comprometida. Se recomienda leer el punto 3.1 para entender mejor la estructura del protocolo. En DNP3 se definen 2 capas (la de aplicación y la de enlace de datos) y una subcapa o subfunción de la capa de aplicación (la función de transporte). Los principales ataques sufridos por capa y comunes a las 3 son:

- En la capa de enlace de datos: la mayoría de los ataques involucran la intercepción de mensajes DNP3, la modificación de los valores de los mensajes y su envío a la estación maestra. Algunos de los ataques afectan a la confidencialidad al obtener datos de configuración e información sobre la topología de la red. Los ataques de integridad insertan datos erróneos o reconfiguran las estaciones receptoras. Los ataques a la disponibilidad del servicio hacen que los dispositivos receptores pierdan la funcionalidad clave o interrumpan las comunicaciones con la estación maestra. En esta capa se han detectado 12 ataques (contando los 3 comunes). A continuación, vamos a analizar 5 ataques de capa de enlace de datos con más detalle:
    - Length Overflow Attack (Ataque por desbordamiento de tamaño máximo): este ataque modifica el valor del campo longitud por uno incorrecto. El ataque puede provocar corrupción de datos, acciones inesperadas o fallos en el dispositivo.
    - DFC Flag Attack (Ataque a la bandera DFC): la bandera DFC se utiliza para indicar que una estación remota está ocupada y que una solicitud debe reenviarse en un momento posterior. Este ataque activa la bandera DFC, que hace que una estación remota aparezca ocupada para el maestro y así retrasa el envío de datos importantes.
    - Reset Function Attack (Ataque código reiniciar): este ataque envía un mensaje con el código de función de enlace 1 (reiniciar el proceso del usuario) a la estación receptora. El ataque hace que el dispositivo se reinicie, dejándolo indisponible durante un periodo de tiempo y, además, probablemente restaurándolo con un estado inconsciente.
    - Unavailable Function Attack (Ataque código no disponible): este ataque envía un mensaje

con el código de función de enlace 14 o 15 (servicio no disponible o no implementado en estación remota). El ataque hace que la estación maestra no envíe solicitudes a la estación remota porque asume que el servicio no está disponible.

- Destination Address Alteration (Alteración del destinatario del mensaje): al cambiar el campo destinatario, el atacante puede redirigir las solicitudes o respuestas al dispositivo que el quiera. También puede usar la dirección de transmisión como destinatario para enviar solicitudes erróneas a todas las estaciones remotas del enlace. Este ataque es difícil de detectar porque no se devuelven mensajes de resultado a una solicitud de transmisión.

- En la función de transporte [5]: la función de transporte provee menos funcionalidad que las otras capas, lo cual provoca que haya menos ataques en esta subcapa. Aun así, se han encontrado 5 ataques (incluidos los 3 ataques comunes) en esta capa. Los dos propios de esta capa son:
  - Fragment Message Interruption (Interrupción de fragmentos del mensaje): las banderas FIR y FIN indican el primer y último fragmento de un mensaje fragmentado. Cuando llega un mensaje con la bandera FIR activada, todos los fragmentos incompletos recibidos previamente se tiran, por lo cual si se activa en el ataque estamos provocando error durante el procesamiento del mensaje obteniendo un mensaje incompleto. La recepción de un fragmento con la bandera FIN activada, finaliza el reensamblaje de fragmentos, por lo cual si se activa en el ataque estamos provocando error durante el procesamiento del mensaje obteniendo un mensaje parcialmente correcto.
  - Transport Sequence Modification (Modificación de la secuencia de transporte): el campo secuencia de la cabecera de transporte se utiliza para garantizar la entrega ordenada de los fragmentos. El número de secuencia aumenta con cada fragmento enviado, por lo que predecir el siguiente valor es trivial. Un atacante que inserta mensajes fabricados en una secuencia de fragmentos puede inyectar cualquier dato y causar errores de procesamiento.

- En la capa de aplicación: la capa de aplicación proporciona la mayor parte de la funcionalidad de DNP3, en consecuencia, la mayor cantidad de ataques están asociados con esta capa. En esta capa se han detectado 17 ataques posibles incluidos los 3 ataques comunes, de los cuales los 5 más llamativos son:
  - Outstation Write Attack (Ataque de escritura en estación remota): este ataque envía un mensaje generado por la estación maestra con el código de función 2 (WRITE) que escribe objetos en una estación remota. El ataque puede dañar la información almacenada en la memoria de la estación remota, provocando errores o desbordamientos.
  - Clear Objects Attack (Ataque de limpieza de objetos): este ataque envía un mensaje generado por la estación maestra con el código de función 9 o 10 (FREEZE_CLEEN) para congelar y borrar datos de una estación remota. El ataque puede borrar datos críticos o hacer que la estación remota se bloquee o funcione mal. Si se usa el código de función 10 el problema es aún más difícil de detectar porque es un mensaje con este código no precisa de respuesta por parte de la estación remota.
  - Outstation Data Reset (Ataque de reinicio de datos en estación remota): este ataque envía un mensaje generado por la estación maestra con el código de función 15 (INITIALIZE_DATA). El ataque hace que la estación maestra que reciba el mensaje reinicialice los objetos de datos a valores inconsistentes con el estado actual del sistema.
  - Outstation Application Termination (Terminación de comunicación con la estación remota): este ataque envía un mensaje generado por la estación maestra con el código de función 18 (STOP_APPL). El ataque hace que la estación remota deje de responder a las solicitudes normales de la estación maestra.
  - Configuration Capture Attack (Ataque de intercepción de configuración): este ataque envía un mensaje alterando un bit de la cabecera que sirve para indicar que el archivo de configuración de la estación remota está dañado. El ataque hace que la estación maestra transmita un nuevo archivo de configuración, que es interceptado por el atacante y luego se ejecuta otro ataque para modificar y cargar el archivo interceptado en la estación remota.

- Ataques comunes: las implementaciones de DNP3 normalmente no usan encriptación, autenticación y autorización. Los dispositivos DNP3 simplemente asumen que todos los mensajes son válidos. Tres ataques aprovechan estas debilidades y, debido a su flexibilidad, se dirigen a las tres capas de DNP3:
  - Passive Network Reconnaissance (Reconocimiento pasivo de la red): un atacante con los permisos apropiados captura y analiza los mensajes DNP3. Este ataque proporciona al atacante información sobre la topología de la red, la funcionalidad de los dispositivos o las direcciones de memoria utilizadas entre otras cosas.
  - Baseline Response Replay (Falsificación de respuesta): un atacante con conocimiento de los patrones de tráfico normales del protocolo simula respuestas a la estación maestra mientras envía mensajes fabricados a las estaciones remotas.
  - Rogue Interloper (Intruso pícaro): un atacante instala un dispositivo de "Man in the middle (intermediario)" entre la estación maestra y la estación remota y, puede leer, modificar y fabricar los mensajes DNP3 que desee.

En conclusión, las consecuencias de los ataques van desde la obtención de la topología de nuestra red hasta la corrupción de las estaciones remotas y la toma de control de las estaciones maestras. Llama la atención la gran proporción de ataques de alto impacto que desarrollan en [4] y en [5], especialmente aquellos que involucran la interrupción, modificación y fabricación de mensajes.

El sistema desarrollado en este trabajo puede ser empleado como una pieza en la detección de todos los ataques nombrados sobre la capa de aplicación y el ataque por desbordamiento de tamaño máximo, el ataque de reinicio de estación receptora, el ataque código no disponible y el ataque de cambio de dirección destino de la capa de enlace de datos. En ningún caso, con este sistema podremos evitar estos ataques.

2 ESTADO DEL ARTE

En este capítulo se presenta la solución adoptada para resolver el problema propuesto y se compara con otras opciones o soluciones que hay en el mercado actualmente.

2.1 Solución adoptada

Las redes ICS están a la orden del día y además cada vez más se está apostando por la integración de esta con la red corporativa de la empresa, provocando esto como hemos analizado una seria amenaza para la seguridad de la ICS. Todo esto junto con el incipiente uso del protocolo DNP3 en Europa es la motivación para la realización de este proyecto.

La solución aportada (Figura 2-1) consta de 3 módulos:

- Un primer módulo que es un disector de paquetes para el protocolo DNP3 implementado como un plugin de la herramienta tranalyzer que sea capaz de extraer en un fichero de texto plano el número de paquete, emisor, destinatario, longitud, código de función, objeto/s, valor/es y la marca de tiempo de cada paquete a partir de un fichero de tráfico pcap que simula la comunicación de estaciones DNP3 en un enlace.
- Un segundo módulo que procesa el fichero salida del disector, genera una serie temporal para cada objeto y almacena estos datos en una base de datos MongoDB.
- Un tercer módulo que es un detector de anomalías para un objeto concreto, implementado realizando una consulta de los valores de un objeto a esta base de datos y analizando valor a valor si es anómalo o no según la técnica de media móvil EMA. También se comprueba la fiabilidad del detector mediante las métricas de la matriz de confusión.

El uso de la herramienta tranalyzer para desarrollar el disector viene motivado por mantener la misma herramienta ya que la idea de este trabajo es formar parte de un IDS (sistema de detección de intrusos) que está desarrollando el Departamento de Ingeniería Telemática de la Universidad de Sevilla para redes industriales.

El proceso de desarrollo de cada una de estas fases se especifica en el punto 5 y todo el código desarrollado se encuentra disponible en el siguiente repositorio de GitHub [6]:

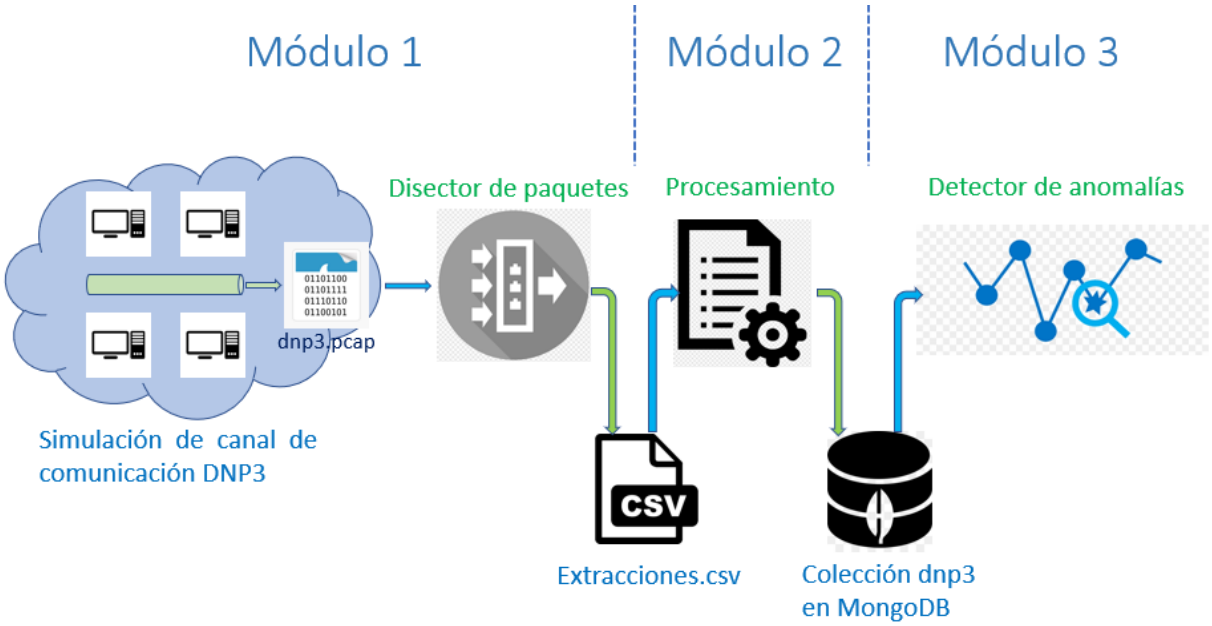

Figura 2-1: Esquema del sistema diseñado.

2.2 Soluciones existentes

Si nos centramos en el sistema completo que se desarrolla en este trabajo, no existe una solución que implemente los 3 módulos completos en conjunto. Sin embargo, si nos centramos en el estudio en cada uno de los módulos si podemos encontrar soluciones alternativas.

Respecto al módulo 1, el disector de paquetes para el protocolo DNP3, existen varios disectores ya realizados con anterioridad, eso sí, ninguno usando la herramienta tranalyzer. Muchos de estos disectores están desarrollados en el lenguaje Python como por ejemplo [7], donde se realiza un módulo para la herramienta Scapy que implmenta un disector diseñando diversas funciones que serán llamadas desde la ejecución principal del programa.

El disector más completo realizado para el protocolo DNP3 es el realizado para la herramienta Wireshark y está desarrollado en lenguaje C [8]. Este disector contempla la salida de todas las variables y todas las posibilidades que presenta cada campo del protocolo ya que su objetivo es analizar el tráfico de la red con el mayor nivel de detalle posible.

En la Figura 2-2 se describe la forma de trabajo que existe en varias herramientas para implementar un disector de paquetes para el protocolo DNP3, si queremos usar Wireshark, deberemos crear un nuevo disector y desarrollarlo en lenguaje C, al igual que si queremos usar tranalyzer tendremos que crear un plugin y desarrollar el disector en lenguaje C mientras que si queremos usar la herramienta Scapy deberemos crear un módulo nuevo y desarrollar el disector en lenguaje Python.

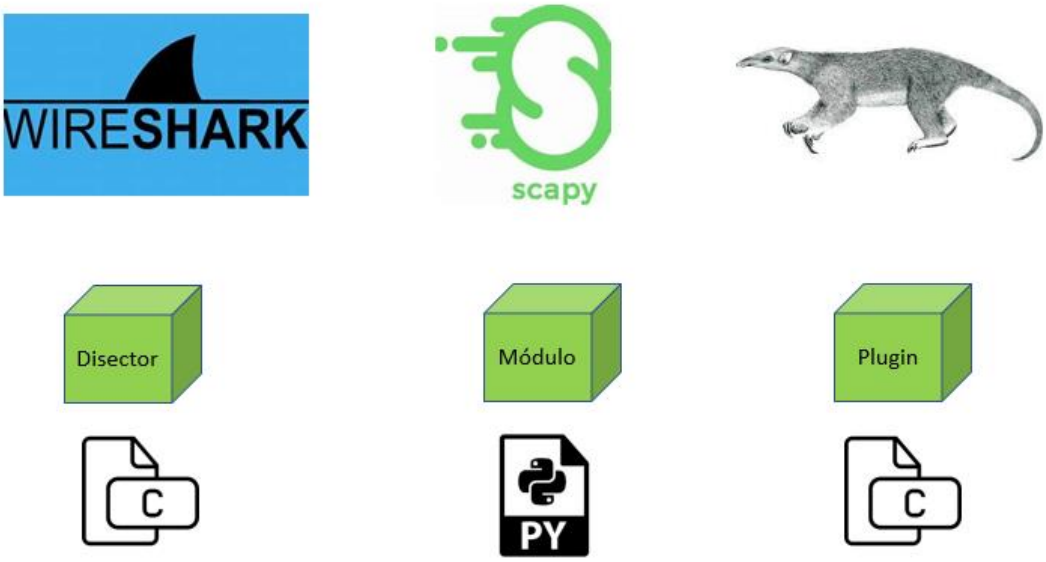

Figura 2-2: Diferentes herramientas para crear un disector de paquetes.

Con relación al módulo 2, existen soluciones que procesan un fichero de extensión csv y lo introducen en una base de datos Mongodb [24] [25]. La misma circunstancia ocurre con el módulo 3, realizar consultas a una base de datos Mongodb [26] y aplicar la técnica de media móvil exponencial (EMA), como los ejemplos expuestos por Agustín Walabonso Lara Romero en tutorías.

3 FUNDAMENTOS TEÓRICOS

En este apartado se van a explicar los conocimientos teóricos necesarios para entender el desarrollo del proyecto, para ello se va a hablar sobre el protocolo DNP3 y técnicas de detección de anomalías básicas.

3.1 DNP3

El protocolo de red distribuida (DNP, Distributed Network Protocol) fue desarrollado por WESTRONIC a inicios de los años 90 con la finalidad de obtener un protocolo libre y orientado a soluciones dentro de la industria eléctrica [9]. Los protocolos de comunicaciones existentes eran propietarios y no eran compatibles con dispositivos de diferentes fabricantes, lo cual motivó que WESTRONIC diseñara su propio protocolo de comunicaciones considerando las principales características de los ya existentes e intentando resolver las principales carencias de estos [10].

Es el protocolo dominante en el sector eléctrico principalmente en Estados Unidos y Canadá, aunque en Europa no lo es tanto debido al uso de otros estándares como IEC-60870 101 o IEC60870 104 pero hay una tendencia al alza de uso. También se puede encontrar en otros campos como el sector del agua y del gas [11].

Las principales características del protocolo DNP3 son [10]:

- Direccionamiento: Se utilizan 2 Bytes para identificar la dirección origen y 2 Bytes para identificar la dirección destino, con lo cual se tiene una capacidad de direccionamiento de hasta 65,000 dispositivos. Se puede hacer difusión con dirección destino los 16 bits a 1 (número 65535).
- Mecanismo de fiabilidad mediante CRC múltiple: implementa un CRC de 16 bits de longitud por cada bloque de 16 bytes.
- Tipo de trama: soporta transmisión asíncrona y síncrona.
- Reconocimiento: Utiliza 10 Bytes para indicar el éxito de la comunicación y el establecimiento del enlace.
- Flexibilidad: soporta las topologías Maestro/Esclavo, red de peer-to-peer y aplicaciones de red.
- Separación por capas: segmentación de capas que permite interactuar con diferentes funciones SCADA.
- Time-Stamped-Data: datos con marca de tiempo que permite registrar la secuencia histórica de eventos.
- Quality Flags: bandera que permite validar los datos que se reciben. En DNP3 este campo será siempre un byte.
- Múltiples formatos de datos: en DNP3 existen distintos formatos de datos, por ejemplo:16-bit, 32-bit, bandera, punto flotante…
- Informar si hay cambio: habilidad de generar mensajes (de respuesta no solicitados) sólo sí las mediciones que se están monitorizando cambian.

La arquitectura de protocolos propia DNP3 es la mostrada en la Figura 3-1:

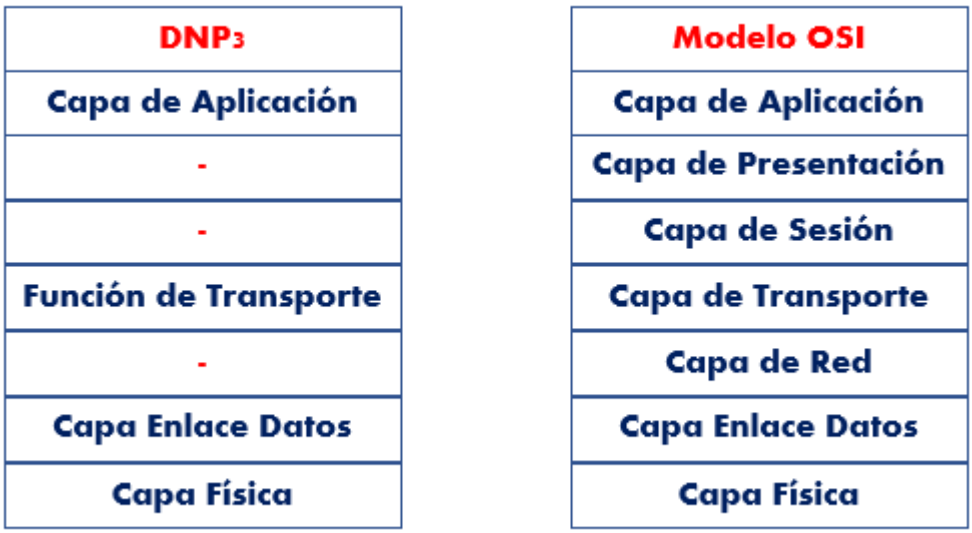

Figura 3-1: Arquitectura de protocolos de DNP3.

Para entender la arquitectura de DNP3 la comparamos con la del modelo OSI, DNP3 define únicamente dos capas que son las de aplicación y la capa de enlace y física (estas dos se entienden muchas veces como una única capa) y además define una subcapa o función para el transporte de las tramas en vez de las 7 capas tradicionales que define el modelo OSI.

No obstante, no podemos olvidar que DNP3 es un protocolo de la capa de aplicación. Para nuestra experimentación la arquitectura de protocolos es la siguiente:

Figura 3-2: Arquitectura de protocolos del escenario.

Usaremos DNP3 encapsulado en IP/Ethernet bajo el protocolo fiable de transporte TCP como se muestra en la Figura 3-2.

En todos los enlaces de comunicación en los que se usa DNP3, se definen estaciones maestras que son las estaciones que piden datos, sincronizan y llevan el mando y estaciones esclavas o receptoras que simplemente

contestan a las peticiones que le hacen las estaciones maestras o informan inmediatamente a la estación maestra cuando un valor cambia. Para simplificar el estudio siguiente, vamos a suponer que nos encontramos en un canal con un enlace DNP3 en el que hay una única estación maestra (EM) y una única estación esclava o receptora (ER).

En una comunicación básica de DNP3 (Figura 3-3), se realiza una petición de datos a la capa de aplicación de la EM y esta automáticamente genera el/los fragmentos necesarios para la petición de datos, seguidamente la función de transporte "etiqueta y numera" los fragmentos para que se puedan ordenar en el destino y la capa de enlace y datos genera la correspondiente trama que es lanzada al canal. Cuando la ER recibe la trama primero manda un ACK confirmando la llegada de la trama a la EM, ya que la capa de enlace es la que proporciona el mecanismo de confirmación, a la vez que le pasa el mensaje a la función de transporte que ordena los fragmentos y se los pasa a la capa de aplicación la cual obtiene la información que se le ha requerido.

La capa de aplicación de la ER genera los fragmentos necesarios con la información requerida por la EM y se repite el proceso explicado arriba de manera inversa. Una vez que la EM recibe los datos que quería manda un mensaje de Confirmación de que ha recibido los datos.

A continuación, se va a describir la composición y el funcionamiento de cada una de las capas que define el protocolo DNP3.

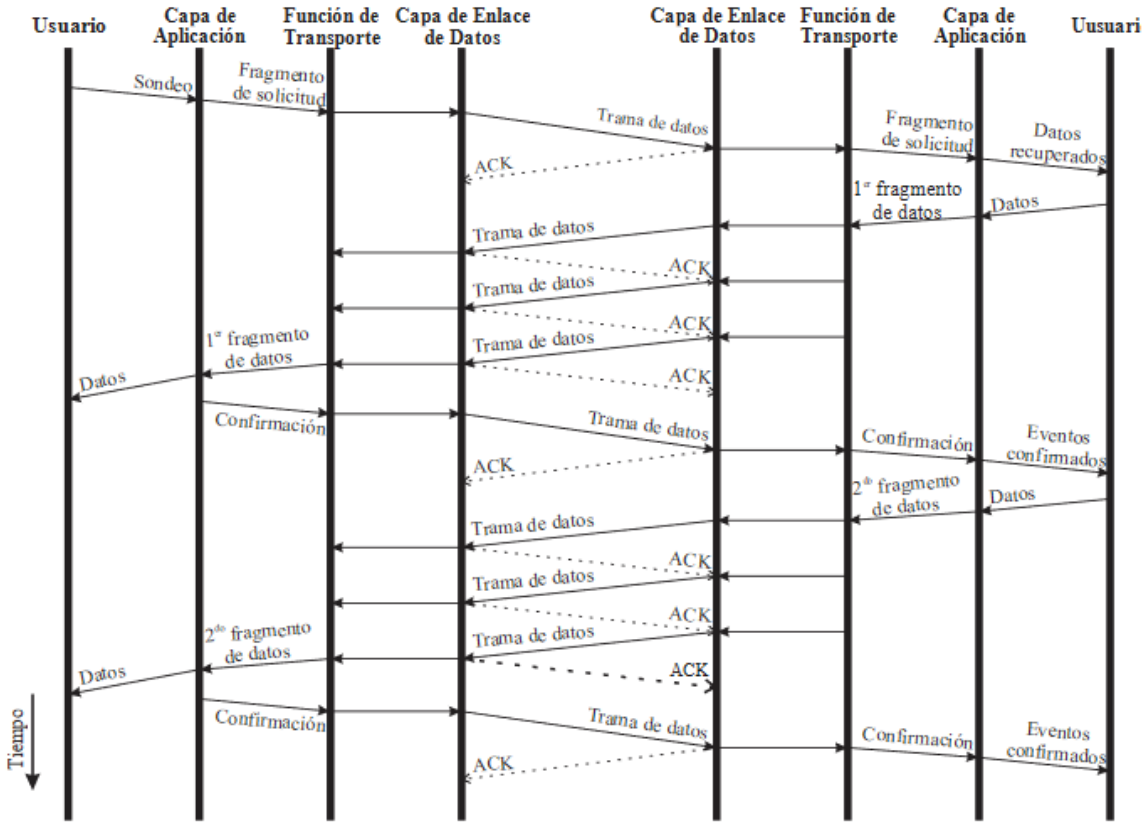

Figura 3-3: Comunicación básica entre dos estaciones DNP3 [10].

3.1.1 Capa de Aplicación DNP3

En DNP3 utiliza el término Punto para asociar una entrada/salida analógica o binaria, o bien a un contador con el valor específico de su medición. Un punto se identifica como un índice, un grupo de pertenencia y una variación que dice que tipo de dato es el del punto.

Un Objeto DNP3 se define como la representación codificada de la medición de un Punto, de acuerdo con el grupo y su variación, para ser transportada en el mensaje. Un mensaje puede contener varios objetos y cada objeto representa el valor de un Punto en un instante dado.

En DNP3 los datos pueden ser estáticos o dinámicos (eventos). Los datos estáticos se refieren al valor actual de un punto. Los eventos son un cambio significativo por ejemplo en la medición de un punto que cruza un umbral marcado.

3.1.1.1 Fragmentos

Un fragmento es un conjunto de bytes que contiene la información solicitada de una EM a una ER. Veremos que un campo de la cabecera del fragmento (código de función) nos dirá cuál es la operación que realiza el fragmento. En DNP3 podemos fijar un tamaño máximo para los fragmentos. Los fragmentos son los mensajes que genera la capa de aplicación.

Como se ha dicho anteriormente un mensaje puede ser de solicitud o de respuesta. Un mensaje de solicitud solo puede ser enviado por una EM mientras que una ER solo puede enviar respuestas a una solicitud de la EM o lo que se conoce como respuestas no solicitadas si tiene alguna información considerada importante que comunicar. La estructura de los fragmentos de solicitud (Figura 3-4) y de respuesta (Figura 3-5) es:

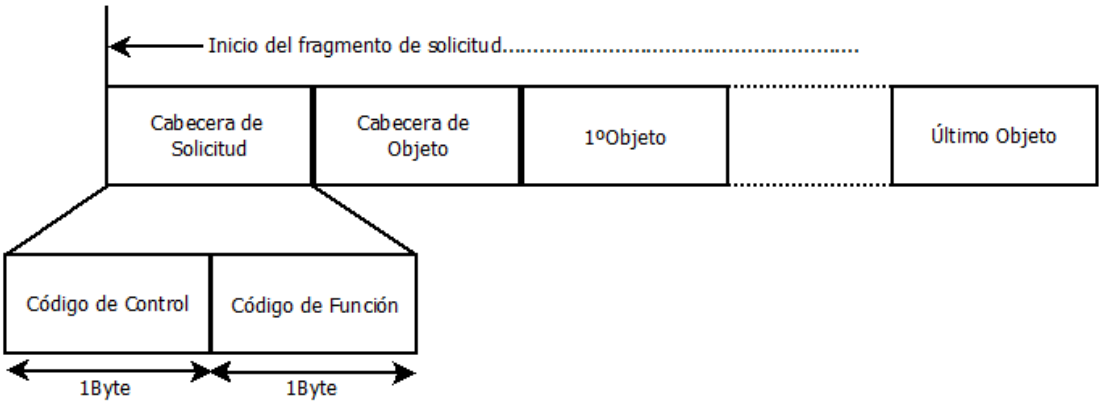

Figura 3-4: Fragmento de solicitud.

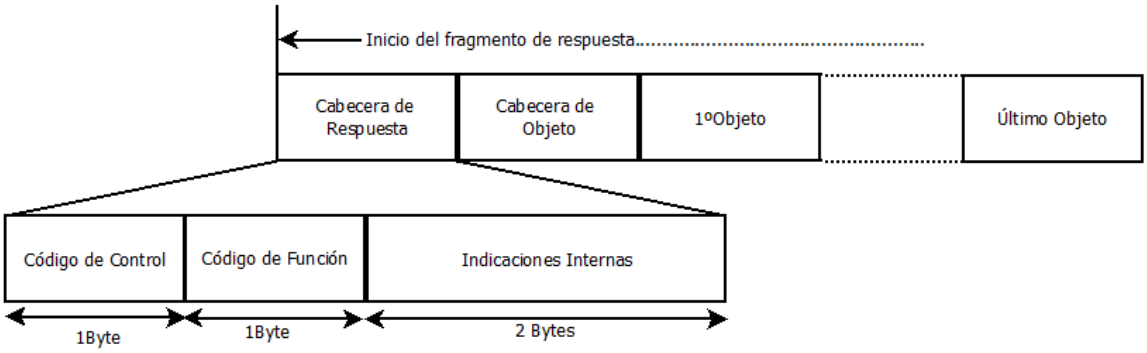

Figura 3-5: Fragmento de respuesta.

La cabecera de la capa de aplicación está compuesta por un código de control de 1 byte y un código de función (FC por sus iniciales en inglés) si es un mensaje de solicitud y si es un mensaje de respuesta a estos 2 bytes habría que añadirle otros 2 de indicaciones internas.

Seguido a esto vienen ya los objetos, es decir, los datos en sí, pero como la cabecera de la capa de aplicación no es suficiente para transmitir la información del mensaje de control, se agregan una cabecera para cada objeto que viaja en el fragmento.

A continuación, se va a detallar el contenido de cada campo de la cabecera de la capa de aplicación:

3.1.1.2 Código de Control

El código de control proporciona información para poder reensamblar fragmentos que se han enviado en diferentes tramas, proporciona un mecanismo para evitar la duplicidad de mensajes e indica si el receptor del mensaje debe contestar a este mensaje. La estructura del byte de control (Figura 3-6) es:

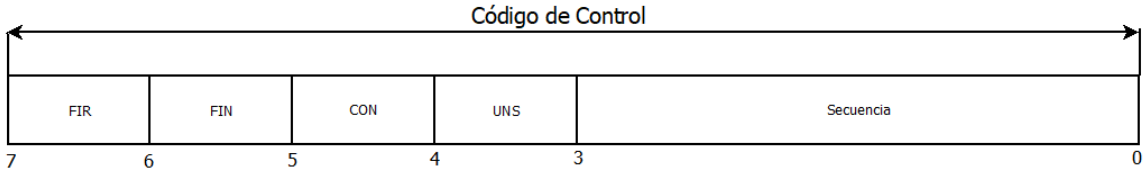

Figura 3-6: Campo código de control.

Donde se diferencian 5 campos, cuyas funciones son:

- FIR (1 bit): indica si es el primer fragmento de un mensaje (FIR=1, activo a nivel alto).
- FIN (1 bit): indica si es el último fragmento de un mensaje (FIN=1, activo a nivel alto).

- CON (1 bit): indica si el receptor debe devolver un mensaje de confirmación desde la capa de aplicación. Este bit lo activa la EM si la EM requiere confirmación de capa de aplicación de la ER y lo activa la ER siempre que el mensaje contiene datos de eventos, además la ER deberá guardar la información almacenada de los datos de eventos hasta que reciba el mensaje de confirmación de la EM. También lo activa la ER cuando envía una respuesta no solicitada para confirmar que la EM ha recibido estos datos mandados con "urgencia".
- UNS (1 bit): indica si el mensaje contiene una respuesta solicitada o la confirmación a una respuesta no solicitada (UNS=1, activo a nivel alto). Si no se activa se entiende que es una respuesta solicitada por la EM.
- SEQ (4 bits): verifica que los fragmentos se reciben en el orden adecuado y además que no están duplicados. Tiene un rango de valores de 0 a 15. Todos los dispositivos deben conocer en todo momento la secuencia que se está utilizando tanto para respuestas solicitadas como para respuestas no solicitadas. Cabe destacar que cada secuencia es independiente de cada comunicación entre dos máquinas.

3.1.1.3 Código de Función

El Código de función es un campo de un byte que identifica el propósito del mensaje. La Tabla 3-1 muestra todos los códigos de función existentes, donde cabe destacar que del 1 al 120 son de mensaje de solicitud, es decir, de la EM, y del 129 al 255 son mensajes de respuesta, es decir, de la ER.

Tipo de mensaje	Código	Nombre	Descripción
Confirmación	0 (0x00)	Confirmación (*CONFIRM*)	La EM envía para confirmar la recepción de un fragmento para la Capa de Aplicación.
Solicitud	1 (0x01)	Lectura (*READ*)	La ER devolverá los datos específicos de los objetos solicitados.
Solicitud	2 (0x02)	Escritura (*WRITE*)	La ER deberá almacenar los datos específicos de los objetos enviados en la solicitud.
Solicitud	3 (0x03)	Selección (*SELECT*)	Una vez que la ER selecciona los puntos de salida especificados por la solicitud, estos están preparados para la siguiente operación. La ER no activa las salidas hasta que se recibe la solicitud de operación.
Solicitud	4 (0x04)	Operar (*OPERATE*)	En la ER se activan las salidas seleccionadas mediante una orden de selección.
Solicitud	5 (0x05)	Operación inmediata (*DIRECT_OPERATE*)	La ER ejecuta inmediatamente los puntos de salida especificados por la solicitud, no esnecesaria una orden de selección.
Solicitud	6 (0x05)	Operación inmediata sin retorno (*DIRECT_OPERATE_NR*)	Es igual al código de función 5, pero en este caso la ER no envía una respuesta.
Solicitud	7 (0x07)	Congelación inmediata (*IMMED_FREEZE*)	La ER copia los valores de los puntos especificados por los objetos en la solicitud que sesepararon por una retención en el búfer.
Solicitud	8 (0x08)	Congelación inmediata sin retorno (I*MMED_FREEZE_NR*)	Es igual al código de función 7, pero la ER no envía una respuesta.
Solicitud	9 (0x09)	Eliminación de la congelación (*FREEZE_CLEEN*)	La ER copia los datos de los puntos especificados por los objetos en la solicitud, los cuales fueron separados por congelación en un búfer. Después de esta operación, los valores se reinicializan a cero.

Solicitud	10 (0x0A)	Eliminación de la congelación sin retorno (*FREEZE_CLEEN_NR*)	Es igual al código de función 9, pero en este caso la ER no envía una respuesta.
Solicitud	11 (0x0B)	Congelamiento por tiempo (*FREZE_AT_TIME*)	La ER copia los datos de los puntos especificados por los objetos en la solicitud, los cuales fueron separados por congelación en un búfer en algún instante y/o en intervalos de tiempos definidos en un objeto especial de tiempo.
Solicitud	12 (0x0C)	Congelamiento por tiempo sin retorno (*FREZE_AT_TIME_NR*)	Es igual al código de función 11, pero en este caso la ER no envía una respuesta.
Solicitud	13 (0x0D)	Reinicio en frío (*COLD_RESTART*)	La ER debe reiniciar completamente el dispositivo.
Solicitud	14 (0x0E)	Reinicio en caliente (*WARM_RESTART*)	La ER sólo reinicia algunas partes del dispositivo.
Solicitud	15 (0x0F)	Inicialización de datos (*INITIALIZE_DATA*)	Obsoleto, no es utilizado en nuevos dispositivos.
Solicitud	16 (0x10)	Inicialización de aplicaciones (*INICIALIZE_APL*)	La ER cambia las aplicaciones a los estados enlistados en los objetos de solicitud.
Solicitud	17 (0x11)	Inicio de aplicaciones (*START_APPL*)	La ER ejecuta las aplicaciones que se especifican en los objetos de la solicitud.
Solicitud	18 (0x12)	Detención de los estados de	La ER detiene las aplicaciones que se especifican en los objetos de la solicitud.
Solicitud	19 (0x13)	Almacenar configuraciones (*SAVE_CONFIG*)	La ER almacena en la memoria no volátil el archivo de configuración ubicado en unamemoria volátil. Este código de función no se utiliza en nuevos dispositivos.
Solicitud	20 (0x14)	Habilitar respuestas no solicitadas (*ENABLE_UNSOLICITED*)	Permite a la ER habilitar respuestas no solicitadas de puntos especificados en los objetosde la petición.
Solicitud	21 (0x15)	Deshabilitar respuestas no solicitadas (*DISABLE_UNSOLICITED*)	Evita a la ER iniciar respuestas no solicitadas de puntos especificados en los objetos de lapetición.
Solicitud	22 (0x16)	Asignación de clases (*ASSIGN_CLASS*)	La ER asigna los eventos generados en los puntos solicitados por los objetos en una de lasclases.
Solicitud	23 (0x17)	Medición de retardos (*DELAY_MEASURE*)	La ER informa el tiempo que le toma procesar e transmitir su respuesta, lo que permite a laEM calcular el retardo programado en el canal de comunicación y sincronizar tiempos[7].
Solicitud	24 (0x18)	Registrar el tiempo del último dato (*RECORD_CURRENT_TIME*)	La ER registra el instante en el que se recibe el último byte para la sincronización detiempo en redes LAN.
Solicitud	25 (0x19)	Abrir un archivo (*OPEN_FILE*)	La ER abre un archivo.
Solicitud	26 (0x1A)	Cerrar un archivo (*CLOSE_FILE*)	La ER cierra un archivo.
Solicitud	27 (0x1B)	Eliminar un archivo (*DELETE_FILE*)	La ER elimina un archivo.
Solicitud	28 (0x1C)	Obtener información de un archivo (*GET_FILE_INFO*)	La ER obtiene información de un archivo.
Solicitud	29 (0x1D)	Autentificación de archivos (*AUTHENTICATE_FILE*)	La ER regresa la clave de autentificación del archivo.
Solicitud	30 (0x1E)	Cancelar transferencia de archivos (*ABOR_FILE*)	La ER cancela la transferencia de archivos.
Solicitud	31 (0x1F)	Activación de configuraciones (*ACTIVATE_CONFIG*)	La ER utiliza la configuración especificada por los objetos transmitidos en la solicitud.
-	32 (0x20) - 128 (0x80)	-	Reservados.
Respuesta	129 (0x81)	Respuesta (*RESPONSE*)	La EM interpreta este fragmento como una respuesta de la Capa de Aplicación a unasolicitud de nivel de aplicación enviada anteriormente.

Respuesta	130 (0x82)	Respuesta no solicitada (*UNSOLICITED_RESPONSE*)	La EM interpreta este mensaje como una respuesta no solicitada, la cual no fue motivadapor una solicitud explícita.
Respuesta	131 (0x83) - 255 (0xFF)	-	Reservados

Tabla 3-1: Códigos de función, descripción, numeración y tipo de mensaje.

3.1.1.4 Indicaciones Internas

Este campo de 2 bytes sólo aparece en la cabecera de los mensajes de respuesta de la capa de aplicación. Se divide en dos campos de un byte cada uno e indica estados de error de la ER. La Tabla 3-2 obtenida de [10] muestra cuales son las diferentes indicaciones internas que pueden comunicar las ER.

Bit	Nombre	Descripción
IIN1.0	Todas las estaciones (*ALL_STATIONS*)	El mensaje fue recibido por todas las estaciones.
IIN1.1	Eventos de clase 1 (*CLASS_1_EVENT*)	La ER no reportó un evento de clase 1.
IIN1.2	Eventos de clase 2 (*CLASS_2_EVENT*)	La ER no reportó un evento de clase 2.
IIN1.3	Eventos de clase 3 (*CLASS_3_EVENT*)	La ER no reportó un evento de clase 3.
IIN1.4	Solicitud de tiempo (*NEED_TIME*)	Se requiere un tiempo de sincronización.
IIN1.5	Control local (*LOCAL_CONTROL*)	Uno o más puntos de la ER se encuentran en modo local.
IIN1.6	Problemas con el equipo (*DEVICE_TROUBLE*)	Indica una condición anormal en una parte de la ER.
IIN1.7	Reinicio de equipo (*DEVICE_RESTART*)	La ER se ha reiniciado.
IIN2.0	Código de función no soportado (*NO_FUNC_CODE_SUPPORT*)	La ER no soporta este código de función.
IIN2.1	Objeto desconocido (*OBJECT_UNKNOWN*)	La estación remota no soporta la operación solicitada enlos objetos de la petición.
IIN2.2	Error de parámetros (*PARAMETER_ERROR*)	Se detectó un error de parámetros.
IIN2.3	Evento de desbordamiento de la memoria de datos (*EVENT_BUFFER_OVERFLOW*)	Existe un evento de desbordamiento de la memoria dedatos en la ER y se ha perdido al menos un caso no confirmado.
IIN2.4	Ejecución en curso (*ALREADY_EXECUTING*)	Se está ejecutando la operación solicitada.
IIN2.5	Configuración incorrecta (*CONFG_CORRUPT*)	La ER detectó una configuración incorrecta.
IIN2.6	Reservado 2 (*RESERVED_2*)	Reservado para uso futuro
IIN2.7	Reservado 1 (*RESERVED_1*)	Reservado para uso futuro

Tabla 3-2: Indicaciones Internas de mensaje de respuesta, descripción y numeración [10].

3.1.1.5 Cabecera de Objeto

En DNP3 los objetos no se pueden incluir en la cabecera de aplicación, por lo cual antes de cada grupo de objetos que se envían en los mensajes solicitud o respuesta se añade una cabecera de objeto. La cabecera de objeto tiene estructura expuesta en la Figura 3-7:

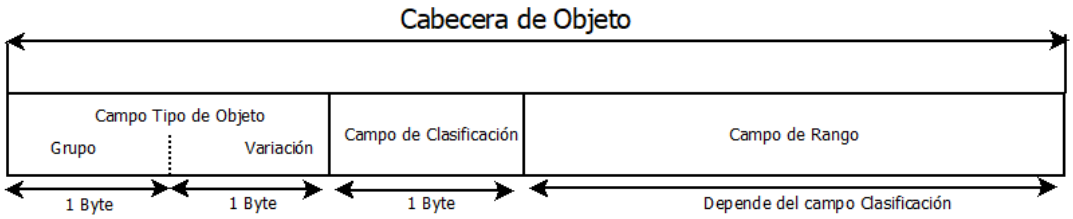

Figura 3-7: Cabecera de objeto.

Es importante destacar que la EM sólo envía información suficiente para que la ER conozca el formato y los valores que requiere. La respuesta por parte de la ER contiene la cabecera de objeto igual o similar al objeto recibido de la EM, seguido por los objetos DNP3. Cada objeto contiene un único valor indexado del punto solicitado.

Vamos a describir detalladamente cada campo de la cabecera de objeto.

3.1.1.6 Grupo y variación

El primer campo de la cabecera de objeto es el campo tipo de objeto que está formado por dos bytes: el primero que define el grupo al que pertenece el objeto y el segundo byte que indica la variación para ese objeto.

El grupo especifica el tipo de dato que la EM solicita a la ER, por ejemplo: valor de un contador, valor de una entrada binaria, valor de una salida binaria…

La variación especifica el formato de los datos del valor indexado del punto solicitado, por ejemplo: entero de 16 bits, entero de 32 bits, bit con bandera…

Juntos, el grupo y la variación definen todos los objetos que existen en DNP3. La Tabla 3-3 muestra los objetos más usados a lo largo de este trabajo indicando el valor del campo grupo y variación, una pequeña descripción de cada uno y el tamaño que ocupa cada uno. Esta última columna será muy útil a la hora de desarrollar el disector de paquetes, cabe destacar que es una columna desarrollada íntegramente por mí con la ayuda de [12], [13], [14] y la prueba de tamaños objeto a objeto realizada con Wireshark, ya que en ningún sitio se proporciona esta información. Para encontrar la tabla completa acudir al Anexo D.

Grupo	Variación	Descripción	Tamaño
01	01	Single-bit Binary Input	1b
01	02	Binary Input With Status	1B
02	01	Binary Input Change Without Time	1B
02	02	Binary Input Change With Time	7B
10	01	Binary Output	1b
10	02	Binary Output Status	1B
11	01	Binary Output Change Without Time	1B
11	02	Binary Output Change With Time	7B
20	01	32-Bit Binary Counter	5B
50	01	Time and Date	6B
60	01	Class 0 Data	0B

Tabla 3-3: Tamaño que ocupa cada punto de cada objeto definido en DNP3.

3.1.1.7 Clasificación

El campo clasificación está formado por un byte que va a indicar también cuanto ocupa y que define el campo rango. El contenido del campo es el mostrado en la Figura 3-8:

- Reservado (1 bit): bit reservado para uso futuro.
- Código de objeto fijo (3 bits): especifica si cada uno de los objetos que siguen a la cabecera de objeto llevan un índice (Prefix) delante de ellos. Todos los valores que pueden tomar estos bits y el tamaño del Prefix se definen en la Tabla 3-4 obtenida de [10]:

Código (Hex)	Descripción	Tamaño de los objetos *Prefix* (Bytes)
0	Los objetos *Pre*fix son empaquetados sin índice.	-
1	Los objetos *Prefix* se han fijado como un índice.	1
2	Los objetos *Prefix* se han fijado como un índice.	2
3	Los objetos *Prefix* se han fijado como un índice.	4
4	Los objetos *Prefix* se han fijado como un tamaño de objeto.	1
5	Los objetos *Prefix* se han fijado como un tamaño de objeto.	2
6	Los objetos *Prefix* se han fijado como un tamaño de objeto.	4
7	Está reservado para un uso futuro.	-

Tabla 3-4: Valores posibles de indexación para los objetos [10].

- Código de especificación de rango (4 bits): especifica el contenido y el tamaño del campo especificación de rango. Todos los valores que pueden tomar estos bits y su descripción se definen en la Tabla 3-5 obtenida de [10]:

Código (Hex)	Descripción	No. de Bytes	Aplicación
0	El campo rango contiene un byte de inicio yun byte de parada de índices.	2	Estos códigos se utilizan cuando los objetos DNP3 se empaquetan de manera ordenada enun índice.
1	El campo rango contiene dos bytes de inicio ydos bytes de parada de índices.	4	
2	El campo rango contiene cuatro bytes de inicioy cuatro bytes de parada de índices.	8	
3	El campo rango contiene un byte de inicio yun byte de parada de direcciones virtuales.	2	Estos códigos se utilizan para especificar lugares contiguos de direcciones específicas deespacio de memoria virtual (depende del proveedor). Estos códigos no se utilizan, pero cuando un proveedor los aplica, a menudo se utiliza con objetos enteros de 8 bits sin signo.
4	El campo rango contiene dos bytes de inicio ydos bytes de parada de direcciones virtuales.	4	
5	El campo rango contiene cuatro bytes de inicioy cuatro bytes de parada de direcciones virtuales.	8	
6	Este campo rango especifica todos los valores, pero no se emplea actualmente.	0	Indica que la EM requiere los valores de todoslos puntos especificados por el grupo de objetos. No hay campo rango.
7	El campo rango contiene un byte que indica la cantidad de objetos.	1	Indica que el campo rango es la cantidad de datos de objeto o índices de objeto. La ER puede responder con un número menor de objetos si la solicitud de la EM utiliza los códigos de clasificación 7, 8 ó 9; ésta es unarazón para elegir un número menor de datos. Una ER debe enviar al menos un objeto de datos sí tiene el tipo de dato solicitado.
8	El campo rango contiene dos bytes que indicanla cantidad de objetos.	2	
9	El campo rango contiene cuatro bytes queindican la cantidad de objetos.	4	
A	Reservado para un uso futuro.	-	Reservado para un uso futuro.
B	Calificador de formato variable. El campo rango contiene un byte que indica la cantidadde objetos.	1	Indica que los datos tienen un formato de longitud variable que no puede ser predefinidopara todos los grupos y variaciones de los objetos.
C	Reservado para un uso futuro.	-	Reservado para un uso futuro.
D	Reservado para un uso futuro.	-	Reservado para un uso futuro.
E	Reservado para un uso futuro.	-	Reservado para un uso futuro.
F	Reservado para un uso futuro.	-	Reservado para un uso futuro.

Tabla 3-5: Campo rango, valores, descripción y tamaño que ocupa [10].

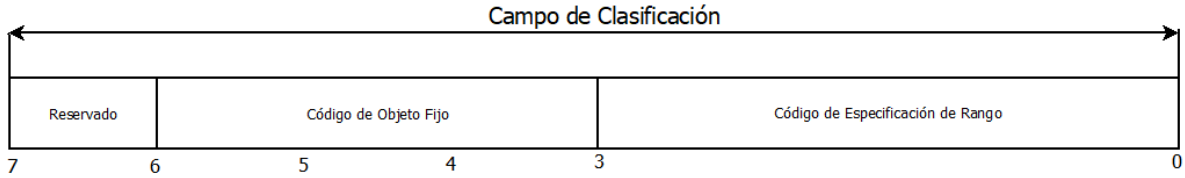

Figura 3-8: Campo de clasificación.

3.1.1.8 Rango

El campo rango viene definido por los 4 bits menos significativos (código de especificación de rango) del campo clasificación. El rango sigue a la cabecera de objetos y si existe sólo aparece tras esta, no antes de cada objeto como el Prefix.

3.1.2 Función de transporte de DNP3

La función de transporte se considera como una subcapa de la capa de aplicación y es el primer byte de cada fragmento. La cabecera de la función de transporte es de 1 byte y está compuesta por los siguientes campos indicados en la Figura 3-9:

- Fin (1 bit): bit de finalización que cuando se activa indica que es el último segmento que se enviará en este fragmento.
- Fir (1 bit): bit de inicio que cuando se activa indica que es el primer segmento del fragmento.
- Secuencia (6 bits): campo utilizado para verificar que los segmentos de cada fragmento se reciben correctamente.

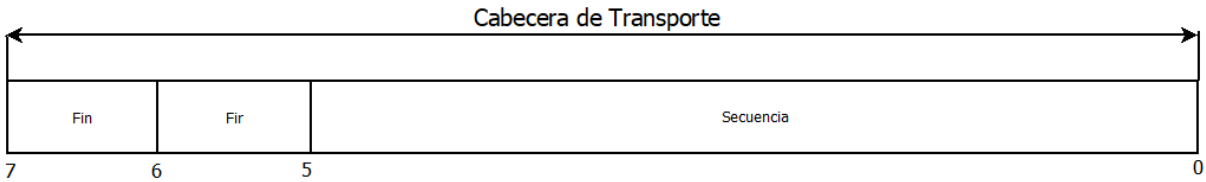

Figura 3-9. Subfunción de transporte.

3.1.3 Capa de Enlace de Datos de DNP3

La Capa de Enlace de Datos proporciona una interfaz entre la Función de Transporte y el canal de comunicación. Las principales funciones de la Capa de Enlace de Datos son el direccionamiento de estaciones y la detección de errores.

La trama en DNP3 tiene un tamaño máximo de carga útil de 255 bytes, donde, carga útil se considera 5 bytes de la cabecera de enlace el byte de la cabecera de función y todo lo que haya generado la capa de aplicación. La trama está compuesta por un bloque de longitud fija de la cabecera de datos (Bloque 0), seguido de los bloques de datos(opcionales). Cada bloque tiene una longitud de 16 bytes como máximo. Entre bloque y bloque nos encontramos con un CRC de 16 bits que no computa para el tamaño máximo de la trama (Figura 3-10).

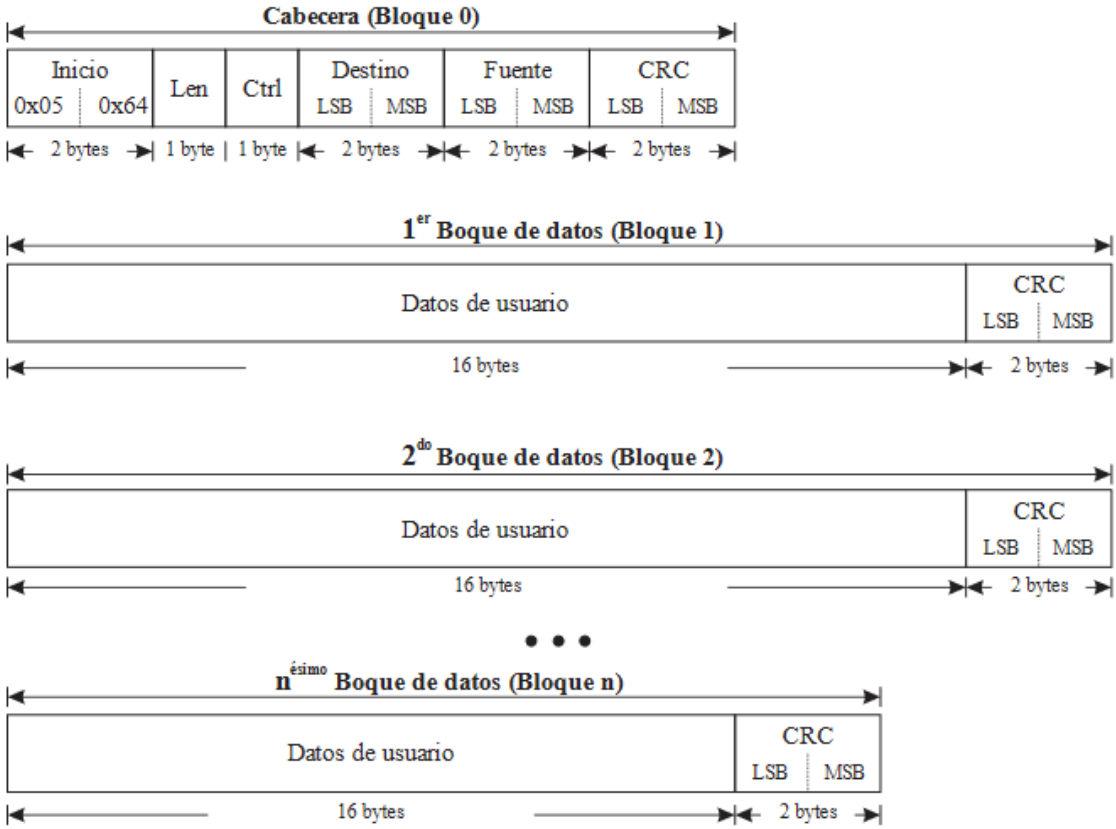

Figura 3-10: Cabecera capa enlace de datos.

La capa de enlace de datos añade los últimos bytes de la cabecera DNP3 que se transmite por el canal de comunicación. La cabecera de la capa de enlace de datos tiene un tamaño de 10 bytes. El formato de la cabecera viene definido por los siguientes campos:

- Inicio: campo de dos bytes que identifica al protocolo. El valor de estos bytes que indican que estamos ante una trama DNP3 en hexadecimal es 0x0564.
- Longitud: campo de un byte que indica el tamaño de la carga útil de la trama. El valor de la longitud incluye los dos bytes de dirección origen, los dos bytes de dirección destino y el byte de control, además del byte de cabecera de función y todos los bytes que genere la capa de aplicación. Tiene un tamaño mínimo de 5 bytes (si no hay mensaje de capa de aplicación) y un máximo de 255 bytes (si la

capa de aplicación ha generado un mensaje de más de 249 bytes) sin contar los CRC que se incluyen por cada bloque. Además, si este campo tiene un valor menor o igual que 10 nos permite saber que el mensaje no incluye objetos ya que es el mínimo número de bytes que nos encontramos antes de cualquier objeto.

- Control: campo de un byte que contiene información sobre si el mensaje ha sido generado por una EM o una ER, sobre la dirección del mensaje (si es una ER respondiendo a una EM) y sobre conteo de tramas.
- Destino: campo de dos bytes que indica la dirección de la estación a la que va enviada la trama. Las direcciones son únicas en cada enlace para poder identificar únicamente a cada estación en un enlace y van desde 0 hasta 65535. La dirección 65535 está reservada para difusión.
- Fuente: campo de dos bytes que indica la dirección de la estación que está transmitiendo la trama.
- CRC: código de redundancia cíclica que se utiliza la estación destino para comprobar que la trama transmitida es igual que la trama recibida. Se añade al final de cada bloque.

3.2 Detección de anomalías

La detección de anomalías tiene como objetivo encontrar patrones en los datos que sean inesperados, es decir, patrones que sean diferentes a la mayoría de los datos y, por lo tanto, sospechosos [15]. Para detectar los valores atípicos deseados, es importante comprender la naturaleza de esas anomalías. Un valor atípico o anomalía se define como un patrón en los datos que no se ajusta al comportamiento normal esperado.

Existen varios métodos de detección de anomalías, se pueden clasificar según diferentes criterios: tipo de entrada, tipo de anomalía, disponibilidad de etiquetas de los datos, tipo de salida del método y técnicas de detección [27].

En función del tipo de entrada un método de detección de anomalías puede ser univariable si analizas una sola variable o multivariable si analizas dos, tres variables a la vez [28].

Según el tipo de anomalía se pueden clasificar en tres categorías [29]:

- Anomalías puntuales: corresponden a anomalías de datos individuales que pueden considerarse anómalos con respecto al resto de datos.
- Anomalías contextuales: es una anomalía contextual si es una anomalía debido al contexto de la observación.
- Anomalías colectivas: es una anomalía colectiva si una colección de instancias es anómala con respecto a todo el conjunto de datos.

Las anomalías más comunes en las series temporales son las anomalías contextuales. Los datos que son anómalos en un contexto específico se denominan anomalías contextuales. Se pueden utilizar dos conjuntos de atributos para definir los datos:

- Atributos contextuales: son aquellos que definen el contexto de una instancia de datos como por ejemplo el atributo timestamp (que determina la posición de una instancia de datos) de los conjuntos de datos de series temporales.

- Atributos de comportamiento: definen cualquier característica no contextual de una instancia de datos.

Un ejemplo de comportamiento es la cantidad de lluvia recogida en cualquier lugar. Una misma instancia de datos podría considerarse normal en un contexto y podría considerarse una anomalía contextual en otro contexto, por ejemplo, que en Sevilla llueva en un año 483 mm no se consideraría anómalo, pero en Santiago de Compostela sí (muy poca lluvia para un año en esta zona).

La detección de anomalías contextuales se investiga comúnmente usando datos de series temporales como la de la serie de temperatura media de cada mes del año de la Figura 3-11:

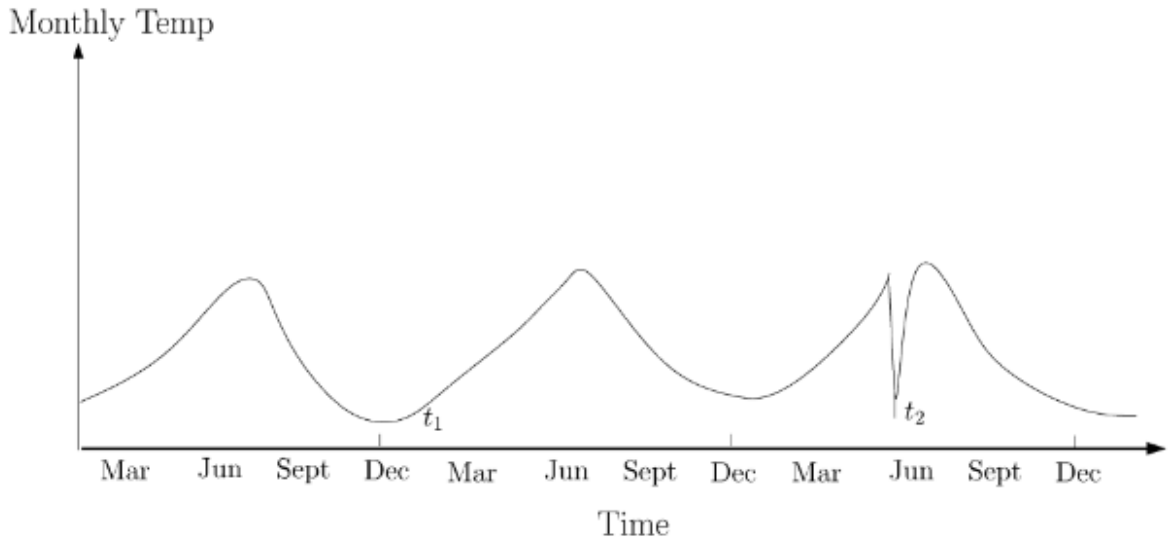

Figura 3-11: Serie temporal de temperatura media mensual [15].

Según la disponibilidad de etiquetas de los datos, los métodos de detección de anomalías se pueden clasificar en supervisados si se necesita la intervención humana para etiquetar, clasificar e introducir los datos en el algoritmo o no supervisados si no hace falta intervención humana [30].

Según el tipo de técnica de detección se pueden clasificar en: análisis estadístico, teoría espectral, teoría de la información, aprendizaje de máquina e híbridas.

Una vez explicadas las diferentes clasificaciones de los métodos de detección de anomalías nos vamos a centrar en el método de detección de la media móvil que es una técnica de tipo univariable, con anomalías de tipo contextual, con disponibilidad supervisada y con técnica de detección de la familia del análisis estadístico.

3.2.1 Media móvil

Un método simple para detectar anomalías contextuales en una instancia de datos es determinar la media móvil del conjunto de datos. Si una instancia de datos difiere considerablemente de la media actual, se puede considerar una anomalía contextual. Una media móvil se realiza analizando los datos en diferentes subconjuntos de la serie de datos completa, es decir, el subconjunto cambia en cada iteración debido a que cuando analizamos un punto nuevo eliminamos el primer punto analizado obteniendo así una media

"actualizada". Al tamaño del subconjunto se le conoce como tamaño de ventana.

Esto a su vez tiene un efecto negativo, estamos introduciendo un retraso en la media debido a que estamos usando valores muy antiguos de la media. Cuanto más largo sea el tamaño de ventana para la media móvil, mayor será el retraso. Por lo tanto, una media móvil de 200 días tendrá un grado de retraso mucho mayor que una media móvil de 20 días. Cuanto más corta sea la ventana utilizada para crear la media, más sensible será a los cambios de valores. Cuanto más larga sea la ventana que cuenta para la media, será menos sensible.

Existen varios tipos de medias móviles, siendo las dos principales la Media Móvil Simple (SMA) y la Media Móvil Exponencial (EMA). La EMA también se hace referencia a menudo como un promedio móvil exponencial ponderada (EWMA) [16].

3.2.1.1 Media móvil simple (SMA)

Una media móvil simple se calcula determinando un período o tamaño de ventana, por ejemplo 5 días. Luego, se calcula una media móvil simple de 5 días haciendo la suma de los últimos 5 valores y dividiéndola por el período. Sin embargo, este tipo de media móvil puede tener un retraso en la detección de anomalías todavía más significativo, debido a que en SMA no se diferencia entre valores más nuevos y valores más antiguos. La fórmula que describe su cálculo es:

$$\mu = \frac{\sum_{t=1}^{n} D_t}{n}$$

donde μ representa el valor de la media móvil simple para este subconjunto, el sumatorio de D_t significa la suma de los n últimos valores de la secuencia de datos y donde n representa el tamaño de la ventana.

El detector que se ha desarrollado en el tercer módulo del sistema implementa esta técnica.

3.2.1.2 Media móvil exponencial ponderada (EWMA)

La EWMA calcula la media de los valores de entrada para un subconjunto o tamaño de ventana que se le especifique dando mayor peso a los datos más recientes. Esto se hace multiplicando un factor de ponderación por cada valor y luego sumando los valores resultantes. El factor de ponderación será menos conforme más antiguo sea el dato que se quiere ponderar [31].

La ventaja de la media móvil ponderada exponencial es que reduce el retraso en la detección que existe con la SMA, lo cual la convierte en una técnica más fiable. Esto se logra con la ponderación de los valores explicada.

Calcular la EWMA implica tomar puntos de datos recientes y asignar un mayor peso en comparación con los puntos de datos anteriores. La suma de todos los pesos debe ser exactamente 1. La fórmula que expresa la EWMA es:

$$\mu = \frac{\sum_{t=1}^{n} W_t \times D_t}{\sum_{t=1}^{n} W_t}$$

donde μ representa el valor de la media móvil exponencial ponderada para este subconjunto, el sumatorio de $W_t \ xD_t$ significa la suma de los n últimos valores (D) de la secuencia de datos multiplicados por el peso específico para cada dato, el sumatorio de W_t representa la suma de todos los pesos y debe ser 1 siempre y donde n representa el tamaño de la ventana.

Para la integración de la técnica EWMA en un detector debemos definir además de la media móvil un umbral que nos indique si un dato es anómalo o no. El umbral se puede definir libremente, aunque el más común y el elegido por nosotros es el que se define como:

$$umbral = (\mu \pm \sigma) \times \rho$$

donde μ representa el valor de media móvil exponencial ponderada calculada, σ representa la desviación típica asociada a dicha media y ρ representa el coeficiente de detección, es decir, el margen de error que permitimos en el detector.

Con lo cual, para determinar un valor como anómalo este valor debe ser mayor que la suma de la media móvil y la desviación típica multiplicada por el coeficiente de detección o menor que la resta de la media móvil y la desviación típica multiplicada por el coeficiente de detección.

4 METODOLOGÍA

En este capítulo se va a analizar la metodología seguida para realizar el trabajo técnico, centrándonos en cuál es el alcance del proyecto y explicando las herramientas utilizadas para realizar el proyecto.

4.1 Alcance

La idea de este proyecto es cubrir las necesidades de seguridad que se demandan en el mundo actual, centrándonos en la seguridad de las redes de entorno industrial y, más concretamente, en el protocolo de comunicación entre SCADAs y dispositivos físicos más usado en la industria eléctrica de Estados Unidos y Canadá: DNP3.

Para poder analizar datos y detectar anomalías en estos, es necesario disponer de un formato de presentación de la información eficiente, sencillo de utilizar y extrapolable a cualquier máquina. Aunque el objetivo principal de este proyecto es detectar anomalías en el tráfico del protocolo DNP3, la parte técnica más difícil de implementar y la más importante es desarrollar un disector de paquetes que ante una comunicación DNP3 en un enlace sea capaz de extraer únicamente las variables útiles para estudiar anomalías y el posterior tratado de la información sacada del disector.

4.2 Herramientas utilizadas

4.2.1 Tranalyzer

Tranalyzer es un inspector de paquetes (como WireShark) y generador de flujo diseñado para profesionales e investigadores en el área de la seguridad de la red. Sus puntos fuertes son la simplicidad, el rendimiento y la escalabilidad. Funciona en terminal, no tiene interfaz gráfica [17].

Como cita en [17] amplía la funcionalidad de Cisco NetFlow y ayuda a los analistas de redes a procesar volcados de paquetes de gran tamaño. Admite el proceso de desglose por flujo o hasta incluso de cada paquete y es capaz de producir rápidamente un pcap reducido, que luego puede analizarse en profundidad mediante su propio modo de paquete basado en texto o simplemente cargarse en tcpdump o Wireshark.

Las principales características de tranalyzer son: la agregación, la geolocalización (mediante IP), soporte de "minería" e inteligencia artificial, la encapsulación, la fácil creación de ficheros de salida pcap, la monitorización o las gráficas que puede generar entre otras.

Esta herramienta es de código abierto (opensource), gratuito, implementado en C y construido sobre la biblioteca libpcap de este lenguaje. La última versión estable la 0.8.10LMW1 del 7 de junio de 2021 [17].

Tranalyzer proporciona funcionalidad para analizar y generar parámetros y estadísticas clave a partir de trazas de IP que se capturan o bien en directo desde interfaces Ethernet o mediante archivos pcap, que será el caso que utilizaré yo. Los usuarios tienen la posibilidad de adaptar la salida a sus necesidades, esto se puede hacer desarrollando plugins adicionales independientemente de la funcionalidad de otros plugins que ya vienen instalados por defecto cuando instalamos tranalyzer.

Los comandos básicos necesarios para ejecutar nuestro disector son [18]:

- t2conf: se usa para configurar tranalyzer.
- t2build: nos permite compilar todos los plugins que indiquemos tras este comando.
- t2: comando que se usa para ejecutar tranalyzer.
- tcol: presenta los ficheros resultados en forma de tabla. Tranalyzer devuelve una fila por cada paquete o flujo con todas las columnas (datos) que tú quieras.

Tranalyzer tiene varios modos de funcionamiento. Los más importantes son:

- Flow mode: modo de funcionamiento enfocado a flujo de paquetes. Se genera una fila por cada dirección de comunicación, es decir, cada vez que cambia el origen o destino en un paquete, mientras el flujo de comunicación sea el mismo tranalyzer agrupa todos los paquetes que se envían en dicho flujo. Este modo de funcionamiento nos permite inspeccionar secuencias y así poder detectar anomalías en el tráfico de paquetes.
- Packet mode: modo de funcionamiento enfocado a un paquete concreto. Se genera una fila por cada paquete que se envía en la red. Este modo de funcionamiento nos permite inspeccionar todos los campos de cada paquete y así poder detectar anomalías en los campos de cada protocolo. Este modo de funcionamiento se usa para realizar Deep Package Inspector (inspectores de paquetes profundos).

Cabe destacar que tranalyzer está centrado en el uso del primer modo de funcionamiento citado, aunque en un futuro su idea es que el servicio sea más útil y completo para todos los modos de funcionamiento.

Para desarrollar un plugin propio [19] dentro del fichero nombre_del_plugin.c hay que definir las siguientes funciones:

- void initialize (): para inicializar la estructura de datos definida en nombre_del_plugin.h debes implementar este método.
- binary_value_t* printHeader (): la primera línea del fichero resultado que genera al ejecutarse suelen ser los campos de las columnas de las tablas. En esta función hay que definirlos.
- void onFlowGenerated (packet_t *packet, unsigned long flowIndex): tranalyzer necesita que le digas lo que debería suceder con el primer paquete. Aquí puedes inicializar todas las estructuras de flujo y máquinas de estados, o almacenar valores iniciales.
- void claimLayer2Information (packet_t *packet, unsigned long flowIndex): solo para los flujos donde nos interese el protocol de nivel 2 (ARP, STP, CDP…) se implementa este método. Todas las operaciones que pertenecen a la capa 2 deben implementarse aquí.
- void claimLayer4Information (packet_t *packet, unsigned long flowIndex): siempre que llega un paquete con capa 4 o superior, el plugin ejecuta este método. Todas las operaciones que quieras realizar sobre un paquete hay que implementarlas aquí. Este método es clave para desarrollar nuestro disector.
- void onFlowTerminate (unsigned long flowIndex): cuando el flujo alcanza el tiempo de espera máximo o se termina antes (por una alarma o lo que sea) se ejecuta este método.

- void onApplicationTerminate (): cuando tranalyzer termina las estructuras y demás deben vaciarse para evitar pérdidas de memoria.

Para instalar tranalyzer en Linux deberá seguir los siguientes pasos:

1. Descargar la última versión del programa ejecutando en la terminal:

   ```
   wget https://tranalyzer.com/download/tranalyzer/tranalyzer2-
   0.8.10lmw1.tar.gz
   ```

2. Descomprimir el fichero descargado

   ```
   tar xzf tranalyzer2-0.8.10lmw1.tar.gz
   ```

3. Acceder al directorio creado tras descomprimir el archivo

   ```
   cd tranalyzer2-0.8.10
   ```

4. Ejecutar el script de instalación que trae el paquete

   ```
   ./setup.sh
   ```

5. Por último, probar a ejecutar el comando "t2", si no lo encuentra ejecutar:

   ```
   source ~/.bashrc
   ```

4.2.2 Pycharm

Pycharm es un entorno de desarrollo integrado (IDE) que proporciona a los desarrolladores principalmente de código en Python la finalización de código y herramientas de reparación rápida [20].

Fue desarrollado por JetBrains en el año 2000. Dispone de una versión completa a la que se accede pagando unos 200$ por la licencia anualmente pero también dispone de una versión gratuita o de estudiantes con una menor funcionalidad, la escogida para este proyecto. La última versión estable es 2021.1.2 lanzada el 2 de junio de 2021.

Se puede usar junto a sistemas de control de versiones. Para este proyecto se ha enlazado con un repositorio del sistema de control de versiones Git donde se encuentra todo el código del proyecto [6].

Para instalar Pycharm en Linux deberá seguir los siguientes pasos:

1. Descargar la última versión de la página oficial.

2. Descomprimir el fichero descargado, para ello desde el directorio donde se ha descargado ejecute:

   ```
   tar xvf pycharm-community-2021.1.2
   ```

3. Para arrancar la aplicación ejecute:

   ```
   cd pycharm-community-2021.1.2/bin
   ```

```
./pycharm.sh
```

La primera vez que entremos nos pedirá un nombre para nuestro proyecto y creará directamente el directorio. Dentro de la carpeta PycharmProjects que genera automáticamente la instalación estarán todos nuestros proyectos.

Para formatear el fichero se ha utilizado la librería pandas. Pandas es una herramienta de manipulación y análisis de datos de código abierto y fácil de usar construida sobre el lenguaje Python. Las principales características de esta librería son: define nuevas estructuras de datos basadas en los arrays de la librería NumPy pero con nuevas funcionalidades, permite leer y escribir fácilmente ficheros en formato CSV, Excel y bases de datos tanto relacionales como no relacionales, permite acceder a los datos mediante índices o nombres para filas y columnas, ofrece métodos para reordenar, dividir y combinar conjuntos de datos, permite trabajar con series temporales y además realiza todas estas operaciones de manera muy eficiente.

Pandas dispone de tres tipos de datos diferentes: las series o datos de una dimensión, los dataframes o estructuras de 2 dimensiones (tablas) y los paneles o estructuras de 3 dimensiones (cubos). En el proyecto se usan los dataframes.

Para integrar pandas en pycharm hay que:

1. Instalar pandas ejecutando desde la terminal:

   ```
   pip3 install pandas
   ```

2. Una vez instalado abrir pycharm ir a File -> Settings -> Project: pythonProject -> Python Interpreter

3. Una vez ahí pulsar en añadir paquete (símbolo +) y buscar e instalar numpy

4. Volver a pulsar añadir paquete y buscar e instalar pandas.

Para que pandas esté integrado y disponible en pycharm tiene que aparecer en los compiladores de Python pandas y numpy como se ilustra en la Figura 4-1.

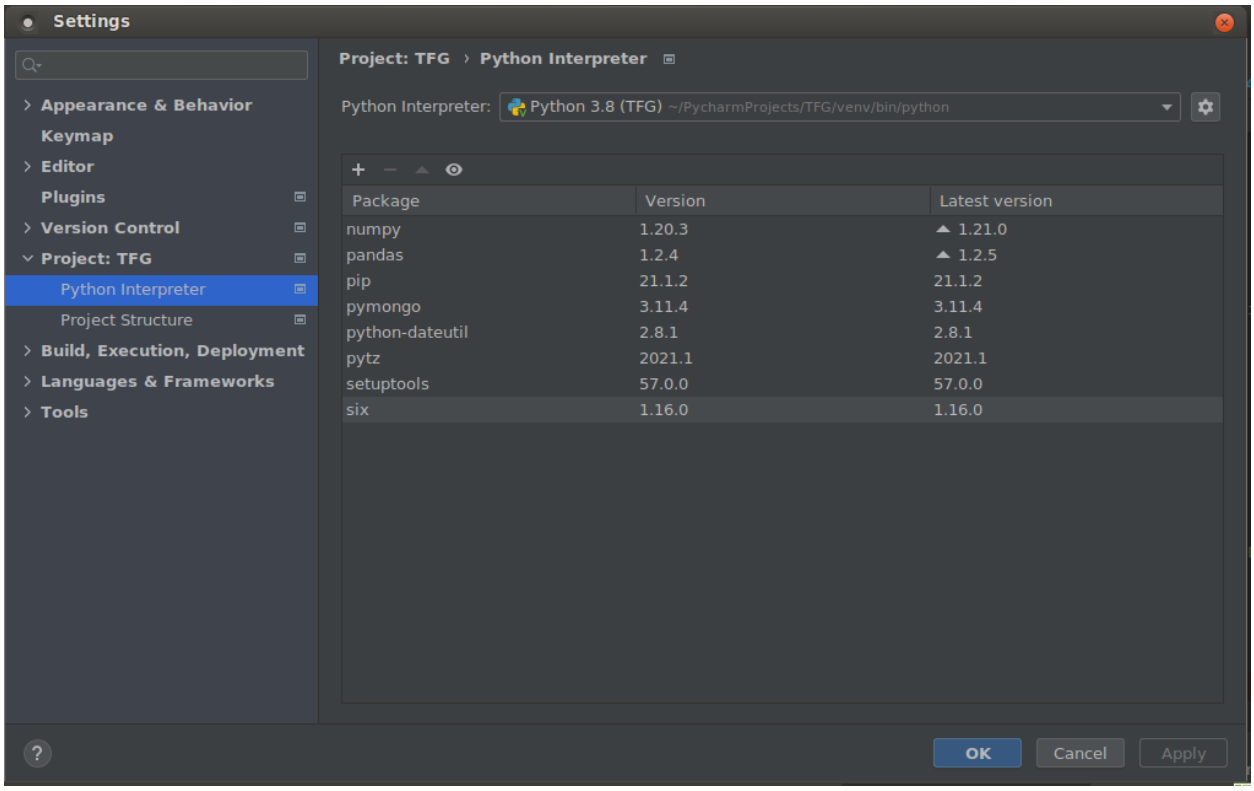

Figura 4-1: Compiladores de Python incluido pandas y numpy.

4.2.3 Mongodb

Mongodb es una base de datos orientada a documentos. Esto quiere decir que, en lugar de guardar los datos en tablas, guarda los datos en documentos, es decir, es una base de datos no relacional. Estos documentos son almacenados en BSON, que es una representación binaria de JSON [32].

Dentro de la estructura de una base de datos Mongodb tenemos las colecciones, que son un concepto similar a las tablas de una base de datos relacional y cada colección se llena con documentos. Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección pueden tener esquemas diferentes, es decir, podemos guardar primero un documento en la colección que tenga por ejemplo tipos string y array de datos y en la misma colección guardar otro documento que solo contenga datos de tipo entero sin que haya ningún conflicto, cosa que no se puede realizar en las bases de datos relacionales.

Se suele utilizar en entornos que requieren de escalabilidad. La principal desventaja de esta base de datos es que no dispone de la operación JOIN, es decir, para consultar datos relacionados de dos o más colecciones deberemos realizar varias consultas.

Fue lanzada al mercado en 2009, está desarrollada en el lenguaje de programación C++. Su última versión estable es la 4.4 lanzada en el año 2020.

Aunque está desarrollada en el lenguaje C++ las consultas se realizan pasando un objeto JSON como

parámetro, tiene sentido ya que los documentos almacenados en las colecciones están en formato BSON (JSON binario). También existen manejadores para poder hacer las consultas en varios lenguajes de programación []. Para este proyecto se ha usado el manejador de Python.

Dispone de dos maneras de trabajo: trabajar con Mongodb en la nube o descargar el servidor en una máquina. Para este proyecto se ha decidido instalar el servidor. Para instalar y gestionar el servicio de Mongodb en Linux deberá seguir los siguientes pasos:

1. Descargar la última versión de Mongodb, para ello ejecute:

   ```
   curl -fsSL https://www.mongodb.org/static/pgp/server-4.4.asc | sudo
   apt-key add –
   ```

2. Ejecutar el siguiente comando para crear un archivo llamado mongodb-org-4.4.list en el directorio sources.list.d.

   ```
   echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
   focal/mongodb-org/4.4 multiverse" | sudo tee
   /etc/apt/sources.list.d/mongodb-   org-4.4.list
   ```

3. Instalar Mongodb

   ```
   sudo apt update
   sudo apt install mongodb-org
   ```

4. Para iniciar/parar el servicio hay que ejecutar

   ```
   sudo systemctl start/stop mongod.service
   ```

Una vez instalado y arrancado el servicio, podemos trabajar de dos maneras: la primera ejecutando mongo en modo Shell, es decir, desde una terminal y la segunda descargándonos un cliente gráfico de Mongodb. Para este proyecto se ha escogido la segunda opción y se ha descargado el cliente Mongodb Compass.

Mongodb Compass nos ofrece una GUI (interfaz gráfica de usuario) que permite gestionar bases de datos mongo de una manera intuitiva. Permite conectarnos a cualquier servidor de mongodb a través de una URI, en nuestro caso como trabajamos en local no hay que proporcionarle ninguna para conectarnos al servidor. Para poder conectarnos el servicio debe estar iniciado y pulsar connect (Figura 4-2).

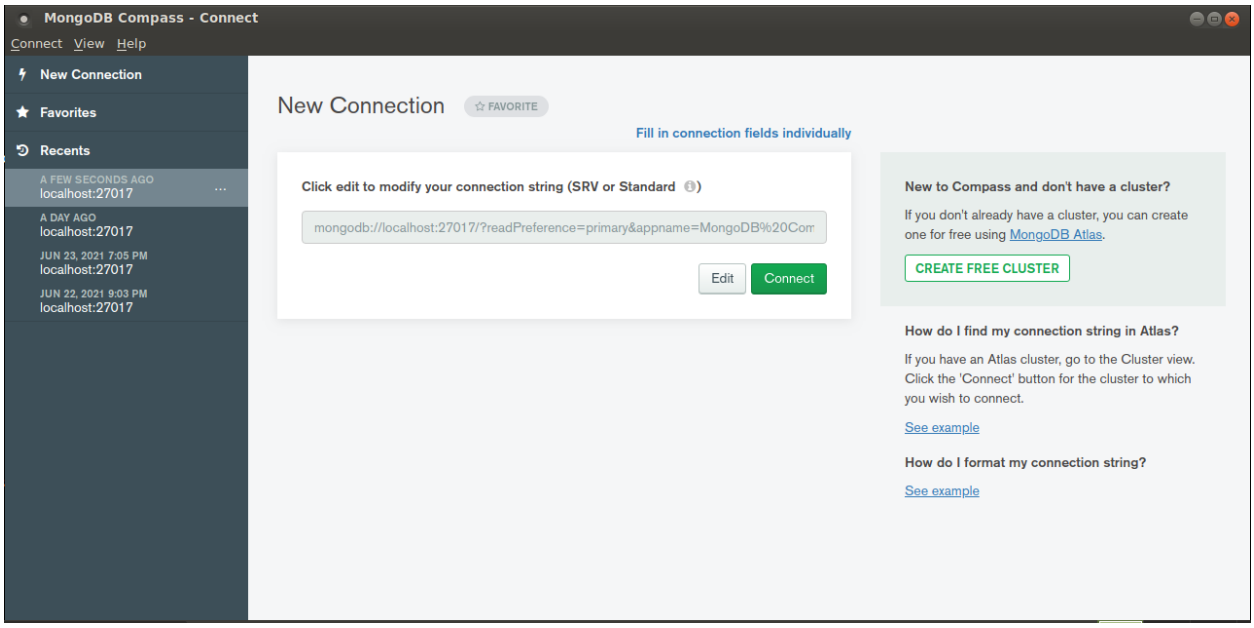

Figura 4-2: Conexión del cliente Mongodb Compass.

Una vez que nos hemos conectado a nuestro servidor podemos escoger que base de datos queremos gestionar. Compass permite realizar todo tipo de operaciones sobre las bases de datos y sobre sus colecciones (Figura 4-3).

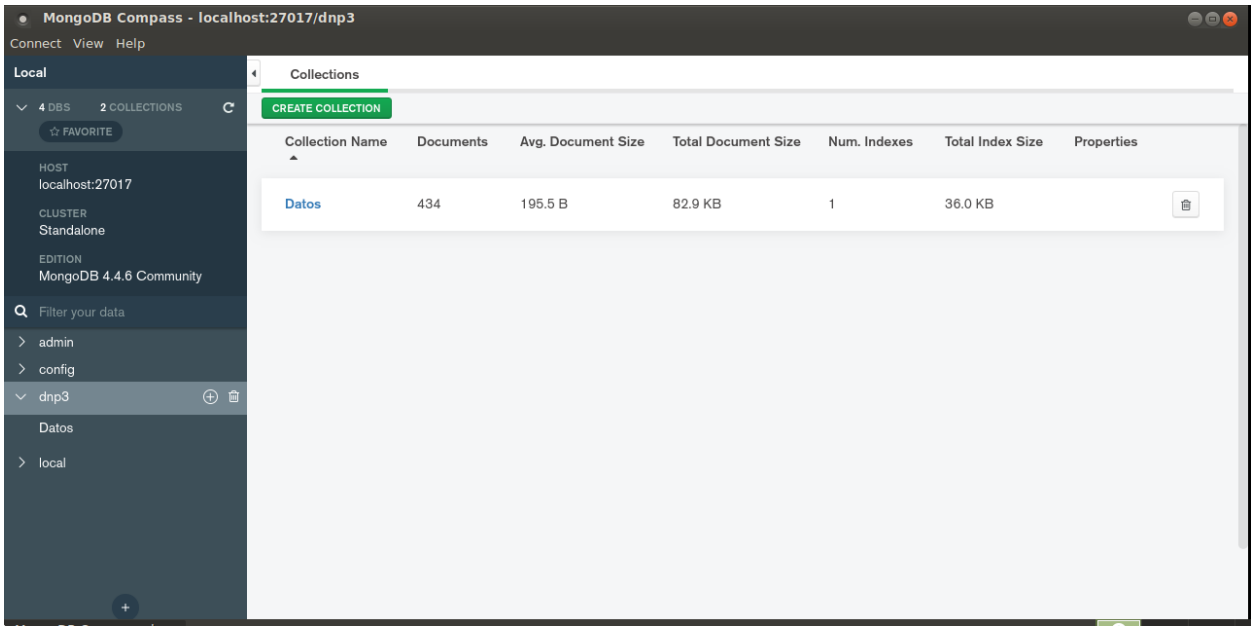

Figura 4-3: Vista de la base de datos dnp3 en Mongodb Compass.

5 DESARROLLO

En este apartado vamos a describir cómo se ha desarrollado el proyecto, para ello vamos a dar una visión global del funcionamiento y características del sistema y una visión más específica de cada una de las fases en las que se ha dividido el proyecto.

5.1 Aspectos generales

La idea principal del proyecto es ser capaces de analizar grandes cantidades de datos del protocolo DNP3 para poder detectar anomalías en ellos. Para poder analizar datos, es necesario tratarlos de manera que podamos ver de una forma sencilla las variables de interés y no ir buscando en cada paquete estas.

Es aquí donde nace la primera gran necesidad y fase del proyecto: realizar un disector de paquetes para el protocolo DNP3 que para cada paquete que hay en un enlace sea capaz de extraer en un fichero únicamente las variables que interesan desde el punto de vista de detección de anomalías. Una vez conseguido este fichero observamos que sigue sin ser cómodo trabajar con un fichero plano de texto.

Por eso surge la segunda necesidad y fase del proyecto: procesamiento del fichero salida del disector e introducir el resultado en una base de datos (en este caso la base de datos escogida es Mongodb, no relacional).

Finalmente, una vez disponemos de los datos almacenados en la base de datos ya podemos extraer las variables que nos interesen y aplicarle las técnicas de detección de anomalías, lo que supone la tercera fase del proyecto.

5.2 Fase 1: Disector de paquetes del protocolo DNP3

5.2.1 Diseño y Desarrollo del disector

Primero es necesario disponer de una comunicación en la que intervenga el protocolo DNP3. Ya que en la escuela no se cuenta con dispositivos que implemente este protocolo, se ha escogido la opción de dar como argumento de entrada al disector ficheros de tráfico generados (extensiones pcap). Los ficheros utilizados están accesibles en [22].

Una vez disponemos de los datos para alimentar a nuestro disector, hay que diseñarlo. A la hora de diseñar el disector hay que tener en cuenta los siguientes factores:

- Cuál es el resultado esperado, es decir, que queremos obtener después de que un flujo de paquetes pase por el disector, por ejemplo, queremos que el resultado sea un fichero de texto plano o no o que variables queremos capturar.
- Como vas a manejar el tráfico que entra en tu disector.

- En que lenguaje de programación vas a desarrollar tu disector (Python, C, Java…).

Este disector se ha desarrollado en el lenguaje C, utiliza la herramienta tranalyzer para manejar el tráfico y su resultado es un fichero de texto plano con los datos deseados.

El uso de tranalyzer viene motivado por el manejo de la librería libpcap por parte de esta herramienta, cosa que facilita el desarrollo. Tranalyzer permite trabajar en modo flujo (el más usado y para el que está desarrollado) y en modo paquete que es el escogido por nosotros. Esta herramienta permite crear un plugin (un módulo) para cada protocolo de cualquier capa que se quiera analizar. Se ha creado un plugin de nombre dnp3 para analizar este protocolo.

Podemos ver el plugin dnp3 como uno más entre otros en la Figura 5-1.

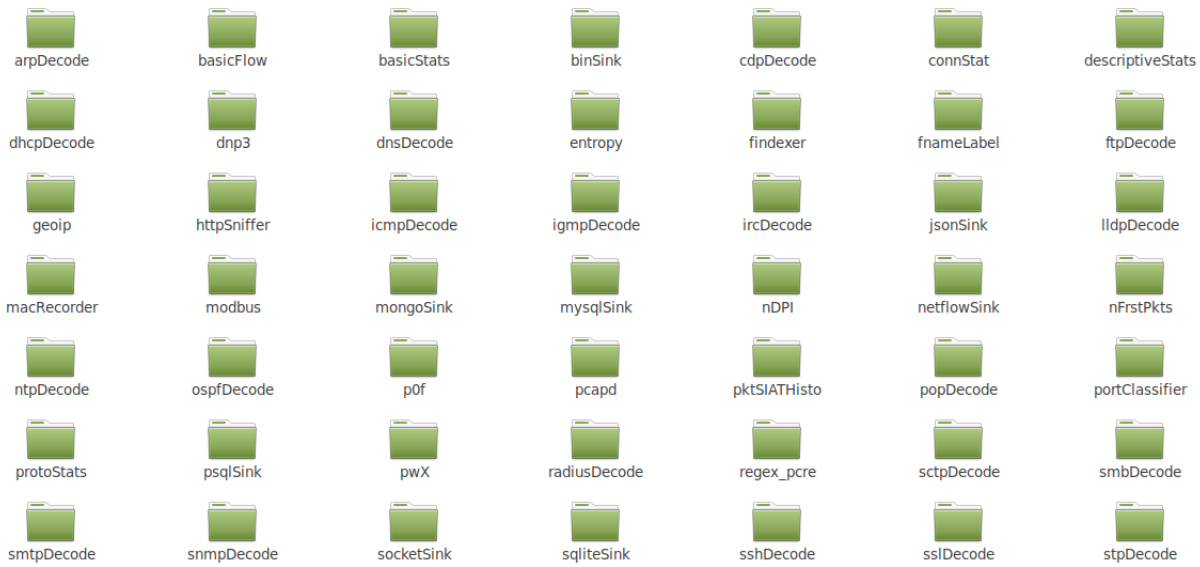

Figura 5-1: Plugins de tranalyzer.

Cuando creamos un plugin la estructura que genera tranalyzer es la indicada en la Figura 5-2:

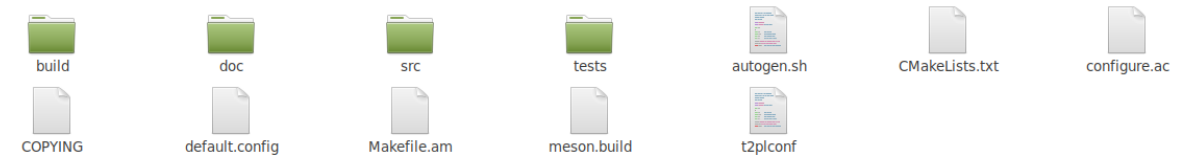

Figura 5-2: Estructura del plugin tranalyzer.

Dentro de la carpeta del plugin tranalyzer genera los archivos necesarios para poder compilar y construir este

plugin como otro más. En la carpeta doc podemos documentar nuestro plugin como queramos. Dentro de la carpeta src es donde está el código del disector propiamente como se muestra en la Figura 5-3.

Figura 5-3: Ficheros de código del disector/plugin.

En esta carpeta están los dos ficheros que definen el disector: dnp3.h y dnp3.c además de un makefile. En el primero se definen las constantes necesarias y la estructura del paquete dnp3 y en el segundo el funcionamiento que va a tener el disector para cada paquete que llegue.

La estructura definida en dnp3.h tiene dos partes (Figura 5-4). La primera unos campos definidos como enteros de un tamaño concreto que siempre aparecen en los mensajes del protocolo:

- Inicio: campo de 2 bytes que corresponde al campo inicio de la cabecera de enlace de datos.
- Len: campo de 1 byte que corresponde al campo longitud de la cabecera de enlace de datos
- Ctrl: campo de 1 byte que corresponde al campo control de la cabecera de enlace de datos
- Destino: campo de 2 bytes que corresponde al campo destino de la cabecera de enlace de datos.
- Fuente: campo de 2 bytes que corresponde al campo fuente de la cabecera de enlace de datos.
- Crc: campo de 2 bytes que corresponde al campo CRC de la cabecera de enlace de datos.
- Transporte: campo de 1 byte que se corresponde con el byte de la subfunción de transporte.
- Ctrl_code: campo de 1 byte que corresponde al campo código de control de la cabecera de aplicación.
- Function_code: campo de 1 byte que corresponde al campo código de función de la cabecera de aplicación.

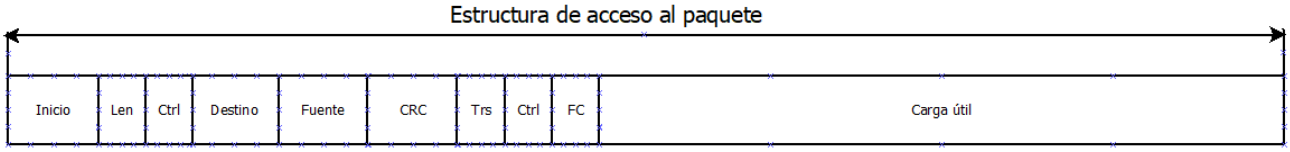

Figura 5-4: Estructura de acceso a un paquete dnp3.

La segunda parte se define como un campo string de longitud variable que puede aparecer o no y tener diferente tamaño en función de cada paquete.

- Carga_util [CARGA_UTIL]: campo de longitud máxima 247 bytes. Aunque 255 es la longitud máxima de carga útil de DNP3, pero hay que restar los 2 bytes de destino, los 2 de fuente, el de control, el de la subfunción de transporte, el del código de control y el del código de función. Este

campo comprende las cabeceras de objetos y los valores de estos. Un mensaje puede contener 1, varios objetos o ninguno, y cada objeto puede recoger 1 valor, varios o ninguno. Esto provoca la variación de campos y tamaños entre un mensaje y otro que es el principal problema que se solventará en la fase 2.

A continuación, vamos a explicar el funcionamiento del disector, para ver el código específico del disector consultar el Anexo A y para entender los detalles del protocolo DNP3 que se han tenido en cuenta se recomienda leer el punto 3.1.

El funcionamiento principal del disector se desarrolla en el método claimLayer4Information () en la parte de como procesamos el paquete si ejecutamos tranalyzer en modo paquete, línea if(sPktFile). Antes de procesar ningún paquete tenemos que instanciar la estructura definida en dnp3.h y crear el fichero de salida añadiendo a este los campos que serán la cabecera de la tabla. Una vez llega el primer paquete y se llama a este método el proceso es el siguiente:

- Comprobamos si el paquete de capa 4 o superior tiene capa de aplicación que es el lugar que ocupa DNP3. Sabemos si hay capa de aplicación o no comprobando si los dos primeros bytes en hexadecimal de la estructura son 0x0564. Si no es paquete con capa de aplicación ni siquiera lo procesamos.
- Cuando se trate de un paquete que contiene datos del protocolo estudiado abrimos el fichero de texto ya creado.
- Luego se elimina la redundancia recorriendo mediante un bucle la carga útil y saltándonos 2 bytes cada 16 bytes. Recuerdo que la capa de enlace añade 2 bytes de redundancia cada 16 bytes de carga útil.
- Filtramos si el paquete recibido contiene objetos o no. Para ello nos aprovechamos de que un mensaje con un objeto debe tener como mínimo una longitud de 11 bytes (2 de destino + 2 de fuente + 1 de control + 1 de transporte + 1 de código de control + 1 de código de función +3 de cabecera identificadora de objeto como mínimo).
- Cuando el paquete no tiene objetos (lo que significa campo de carga útil vacío) directamente imprimimos las variables de interés en el fichero y acabamos el tratamiento de ese paquete. Por conveniencia para el posterior tratamiento de los datos, cuando un fichero no tiene objetos ni valores el campo número de objetos se pone a 0 y el campo que define los objetos y valores o puntos se pone a “ND” (No Definido).
- Cuando el paquete contiene objetos, el siguiente paso es separar los mensajes en función de su tipo: de solicitud y de respuesta porque en los mensajes de respuesta el acceso a los datos se realiza con 2 bytes de desplazamiento ya que cuentan con otro campo en la cabecera de aplicación llamado indicaciones internas. Podemos filtrar los mensajes según el campo código de función de la cabecera de aplicación, debido a que si el código de función tiene un valor en hexadecimal 0x00 < FC < 0x80 se trata de un mensaje de solicitud mientras que si tiene un valor mayor o igual que 0x80 se trata de uno de respuesta. El tratamiento que sufre el paquete si es de respuesta o de solicitud es prácticamente el mismo, la diferencia radica en los dos bytes de desplazamiento y en que en un mensaje de respuesta un objeto puede incluir más de un punto (valor).
- Seguidamente se introduce al paquete en un bucle que va leyendo byte a byte objeto por objeto hasta que ha leído todos los octetos que indica el campo longitud. Para saber cuándo hemos terminado de leer un objeto y toca leer el siguiente nos aprovechamos simplemente del conocimiento del tamaño que ocupa el objeto, es decir, sabemos que la cabecera de objeto está formada por 2 bytes que identifican al objeto más 1 byte de clasificación, campo que nos indica si disponemos de campo rango o no en la cabecera de objeto y que tamaño ocupa y si disponemos de prefijo antes de cada punto o no y que tamaño ocupa. Esto significa que dinámicamente sabemos cuánto ocupa cada objeto.
- Si el objeto contiene varios puntos (caso únicamente de mensajes de respuesta) este número de puntos

nos será indicado en el campo rango de la cabecera de objeto. Este hecho junto con el conocimiento del tamaño de un punto para todos los objetos, ilustrado en la tablaxx, nos permite ser capaces de leer un objeto de manera correcta.

- Una vez procesados los mensajes con objetos se imprimen en el fichero de salida los valores de interés.
- Por último, cerramos el fichero de salida hasta que tranalyzer vuelva a llamar a la función claimLayer4Information ().

Este procedimiento se refleja en el siguiente diagrama de flujo (Figura 5-5):

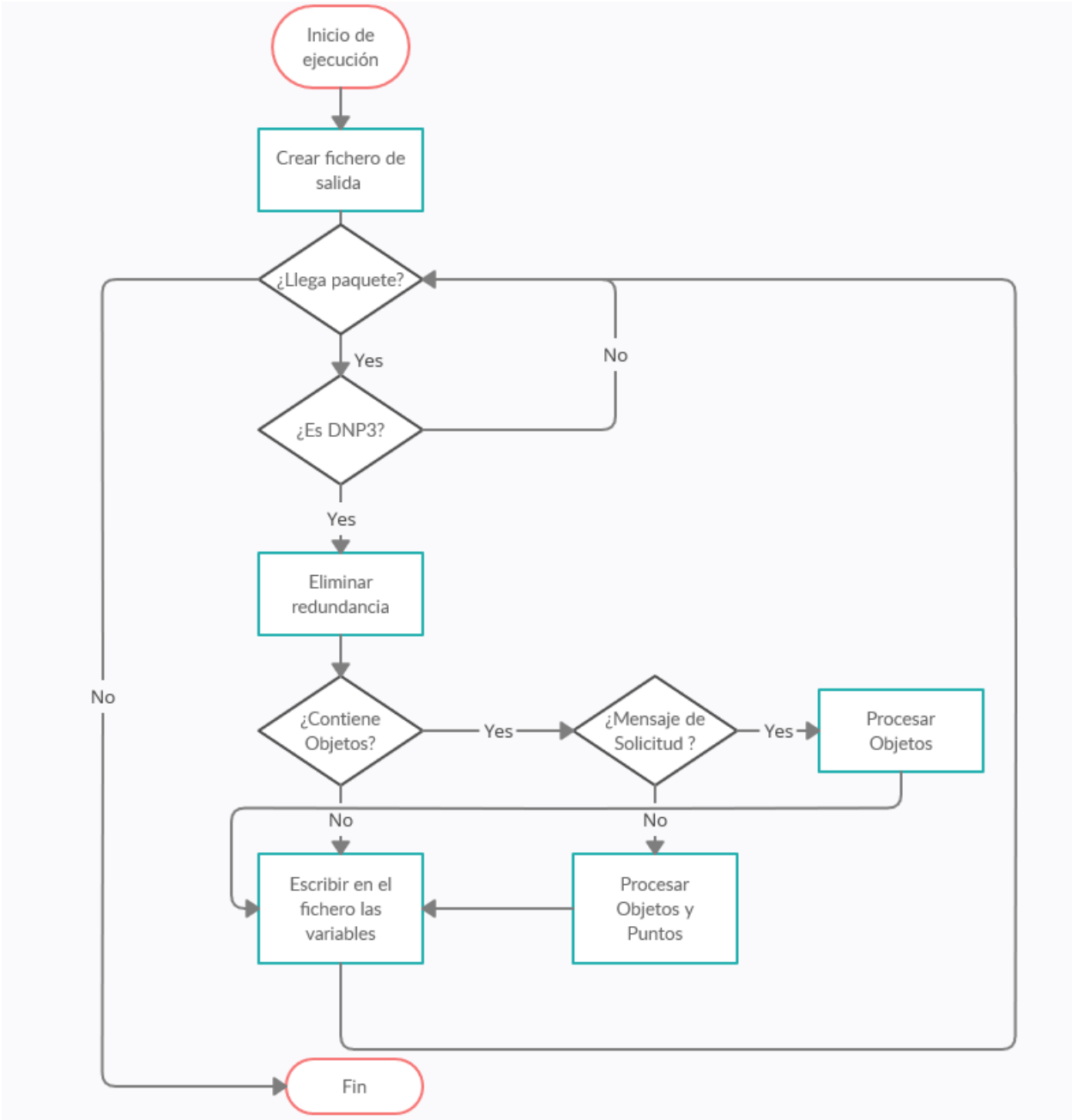

Figura 5-5: Diagrama de flujo del disector de paquetes.

De cada paquete procesado nos interesa solo una serie de variables. Vamos a extraer el número de paquete

procesado, el origen, el destino, la longitud, el código de función, el número de objetos y un campo variable único que contenga objetos y valores o este vacío. El valor de cada campo estará separado del siguiente únicamente por ‘,’ y en el campo variable cada nombre de objeto y cada valor estará separado por ‘;’. Este formato se ha escogido para dejar el fichero en formato de tabla, es decir, que todas las filas tengan el mismo número de campos o columnas, hecho que nos facilitará el trabajo a la hora del procesamiento de los datos. De esta forma el fichero de salida tiene el estilo mostrado en la Figura 5-6:

```
NumeroPaquete,Origen,Destino,Longitud,CodigoFuncion,CampoVector,NumeroObjetos
1,4,3,10,Unsolicited Response,ND,0
2,3,4,8,Confirm,ND,0
3,3,4,18,Write,Time and Date (Obj:50  Var:01);ffffffffffff00;,1
4,4,3,10,Response,ND,0
5,3,4,17,Disable Spontaneus Messages,Class 1 Data (Obj:60  Var:02);Null;Class 2 Data (Obj:60  Var:03);Null;Class 3 Data (Obj:60  Var:04);Null;,3
6,4,3,10,Response,ND,0
7,4,3,10,Unsolicited Response,ND,0
8,4,3,10,Unsolicited Response,ND,0
9,3,4,8,Confirm,ND,0
10,3,4,18,Write,Time and Date (Obj:50  Var:01);ffffffffffff00;,1
11,4,3,10,Response,ND,0
12,3,4,17,Enable Spontaneus Messages,Class 1 Data (Obj:60  Var:02);Null;Class 2 Data (Obj:60  Var:03);Null;Class 3 Data (Obj:60  Var:04);Null;,3
```

Figura 5-6: Ejemplo de fichero salida del disector.

La herramienta tranalyzer también proporciona un fichero de salida tras la ejecución. Vamos a aprovechar este fichero para extraer la columna de marca de tiempo del paquete que incluiremos en nuestros datos en el procesado posterior.

5.2.2 Explicación de funciones del código destacadas

En el código se desarrollan varias funciones para descargar la funcionalidad del método claimLayer4Information () o para “traducir al lenguaje humano” campos del protocolo. Se va a describir el desarrollo de las más importantes.

La función getFCString (Figura 5-7) a la que se le pasa como parámetro el valor en hexadecimal del código de función y que devuelve una cadena de caracteres con el nombre de la función que se realiza correspondiente al valor pasado. Por eso en el fichero de salida el código de función se imprime como una cadena de caracteres y no como su valor en hexadecimal.

```
char* getFCString (uint8_t fc){
  char* cadena  = "";
  switch (fc) {
  case 0x00:
    cadena = "Confirm";
    break;
  case 0x01:
    cadena = "Read";
    break;
  .
  .
  .
  case 0x83:
    cadena = "Authentication Response";
    break;
  }

  return cadena;
}
```

Figura 5-7: Parte del código de la función getFCString.

La función getObjectString (Figura 5-8) a la que se le pasa como parámetros el campo grupo y el campo variación de la cabecera de objeto y que devuelve una cadena de caracteres con el nombre del objeto que representa esos dos campos en hexadecimal. Por eso en el fichero de salida el objeto se identifica con una cadena de caracteres y no con el valor de estos dos campos en hexadecimal.

```
char* getObjectString (uint8_t grupo, uint8_t variacion) {
  char* cadena = "";
  uint16_t object = (grupo << 8) | variacion;
  switch (object) {
  //Binary Input Objects
  case 0x0100:
    cadena = "Binary Input Default Variation (Obj:01  Var:Default)";
    break;
  case 0x0101:
    cadena = "Single-Bit Binary Input (Obj:01  Var:01)";
    break;
  .
  .
  .
  case 0x5001:
    cadena = "Internal Indications (Obj:80  Var:01)";
    break;
  }
  return cadena;
}
```

Figura 5-8: Parte del código de la función getObjectString.

La función getObjectSize (Figura 5-9) a la que se le pasa como parámetros el campo grupo y el campo variación de la cabecera de objetos y devuelve un entero con el tamaño que ocupa un punto o valor de ese objeto. Es una función muy útil para el estudio de los bytes que se van leyendo y para poder acotar el tamaño de cada objeto.

```
int getObjectSize (uint8_t grupo, uint8_t variacion){
  int size = 0;
  uint16_t object = (grupo << 8) | variacion;
  switch (object) {
  //Binary Input Objects
  case 0x0100:
    size = 1;
    break;
  case 0x0101:
    size = 1;
    break;
  .
  .
  .
   //Class Data Objects
  case 0x3c01:
    size = 0;
    break;
  }
  return size;
}
```

Figura 5-9: Parte del código de la función getObjectSize.

La función getClassValue (Figura 5-10) a la que se le pasa como parámetro el campo clasificación y devuelve el tamaño del prefijo (prefix) que va a tener el objeto antes de cada punto o valor. Opera con los bits del 3 al 6 del campo clasificación y compara con los valores de la Tabla 3-4 para saber el tamaño de prefijo para este objeto. Esta función nos permite saber si el valor o valores que trae un objeto vienen indexados o no y cuanto ocupa ese índice. También es muy útil para el estudio de los bytes que se van leyendo y para saber en qué posición nos encontramos los valores.

```
int getClasValue (uint8_t clasificacion){
  int tam_prefix = 0;
  if ((clasificacion & 0x70) == 0x00){
    tam_prefix = 0;
  }
  else if ((clasificacion & 0x70) == 0x10){
    tam_prefix = 1;
  }
  else if ((clasificacion & 0x70) == 0x20){
    tam_prefix = 2;
  }
  else if ((clasificacion & 0x70) == 0x30){
    tam_prefix = 4;
  }
  else if ((clasificacion & 0x70) == 0x40){
    tam_prefix = 1;
  }
  else if ((clasificacion & 0x70) == 0x50){
    tam_prefix = 2;
  }
  else if ((clasificacion & 0x70) == 0x60){
    tam_prefix = 4;
  }
  return tam_prefix;
}
```

Figura 5-10: Código de la función getClassValue.

La función getRangeValue (Figura 5-11) a la que se le pasa como parámetro el campo clasificación de la cabecera de objeto y devuelve el tamaño del campo rango de la cabecera de objeto. Opera con los bits del 0 al 3 del campo clasificación y compara con los valores de la Tabla 3-5 para saber si existe el campo rango, que significa el valor que trae este campo y cuanto ocupa. Esta función nos ayuda en el estudio de los bytes que se van leyendo, y además en los mensajes de respuesta cuando el contenido del campo rango indica cuántos puntos o valores contiene el objeto.

```
int getRangeValue (uint8_t clasificacion){
  int rango = 0;
  if ((clasificacion & 0x0F) == 0x00){
    rango = 2;
    cantidad = false;
  }
  if ((clasificacion & 0x0F) == 0x01){
    rango = 4;
    cantidad = false;
  }
  if ((clasificacion & 0x0F) == 0x02){
    rango = 8;
    cantidad = false;
  }
  if ((clasificacion & 0x0F) == 0x03){
    rango = 2;
    cantidad = false;
  }
  if ((clasificacion & 0x0F) == 0x04){
    rango = 4;
    cantidad = false;
  }
  if ((clasificacion & 0x0F) == 0x05){
    rango = 8;
    cantidad = false;
  }
  if ((clasificacion & 0x0F) == 0x06){
    rango = 0;
    cantidad = false;
  }
  if ((clasificacion & 0x0F) == 0x07){
    rango = 1;
    cantidad = true;
  }
  if ((clasificacion & 0x0F) == 0x08){
    rango = 2;
    cantidad = true;
  }
  if ((clasificacion & 0x0F) == 0x09){
    rango = 4;
    cantidad = true;
  }
  if ((clasificacion & 0x0F) == 0x0B){
    rango = 1;
    cantidad = false;
  }
  return rango;
}
```

Figura 5-11: Código de la función getRangeValue.

Por último, la función printStaticObject (Figura 5-12) a la que se le pasa como parámetros el campo grupo y el campo variación de la cabecera de objeto, un punto o valor del objeto y el descriptor del fichero de salida y no devuelve nada. Esta función es la encargada de dar el formato de impresión a todos los tipos de datos que se manejan en DNP3. Hay que destacar que los datos son enviados en orden inverso al que corresponde y los valores vienen en complemento a 2 por defecto, por eso el uso de unsigned char en lugar de char asecas.

```
void printStaticObject (uint8_t grupo, uint8_t variacion, unsigned char valor[], FILE * fich) {
  //En función del tamaño del objeto y de su formato hacer su propio formato de impresión
  uint16_t object = (grupo << 8) | variacion;
  switch (object) {
  case 0x3201:
    fprintf(fich, "0x%x%x%x%x%x;", valor[4], valor[3], valor[2], valor[1], valor[0]);
    break;
  case 0x3301:
    fprintf(fich, "0x%x%x%x%x%x;", valor[4], valor[3], valor[2], valor[1], valor[0]);
    break;
  case 0x3402:
    fprintf(fich, "0x%x%x", valor[1], valor[0]);
    break;
  case 0x3c01:
    fprintf(fich, "Null;");
    break;
  case 0x3c02:
    fprintf(fich, "Null;");
    break;
  case 0x3c03:
    fprintf(fich, "Null;");
    break;
  case 0x3c04:
    fprintf(fich, "Null;");
    break;
  case 0x0203:
    if (valor[0] & 0x80){
      fprintf(fich, "1;");
    }
    else {
      fprintf(fich, "0;");
    }
    break;
  case 0x2001:
    fprintf(fich, "%d%d%d%d;", valor[4], valor[3], valor[2], valor[1]);
    break;
  default:
    fprintf(fich, "No implementado;");
    break;
  }
}
```

Figura 5-12: Código de la función printStaticObject.

5.3 Fase 2: Formateo del resultado del disector e introducción de los datos en una base de datos

Una vez disponemos del resultado del disector de paquetes vemos como la presentación de los datos no es eficiente ni correcta para trabajar con ellos. Una manera eficiente para trabajar con un gran volumen de datos es almacenando estos en una base de datos y obteniendo mediante consultas los datos que nos interesen. Por todo esto surge la necesidad de formatear el fichero de salida y dejarlo en un formato apto para almacenarlo en una base de datos.

Primero tenemos que escoger un formato para la representación de los datos. El formato actual es una tabla

con 8 columnas que son respectivamente el número de paquete procesado, el origen, el destino, la longitud, el código de función, un campo que aglutina todo lo relacionado con los objetos y el campo número de objetos. Como se ha comentado previamente, estos campos están separados entre sí por ‘,’ y dentro del campo vector los objetos de los valores están separados entre sí por ‘;’. Esto nos permitirá poder acceder en todo momento al campo que queramos.

NumeroPaquete	Origen	Destino	Longitud	CodigoFuncion	CampoVector	NumeroObjetos
valor 1	valor 1	valor 1	valor 1	valor 1	valor1; valorn	valor 1
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
valor n	valor n	valor n	valor n	valor n	valor1; valorn	valor n

Tabla 5-1: Formato tabla de salida del disector.

El formato escogido es una tabla con 8 columnas que son respectivamente el número del paquete procesado, la etiqueta de tiempo para ese paquete, el origen, el destino, la longitud, el código de función, el objeto y un valor. Cada fila contendrá únicamente un valor de un objeto.

NumeroPaquete	Timestamp	Origen	Destino	Longitud	CodigoFuncion	Objeto	Valor
valor 1	valor 1	valor 1	valor 1	valor 1	valor 1	valor 1	valor 1
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
valor n	valor n	valor n	valor n	valor n	valor n	valor n	valor n

Tabla 5-2: Formato tabla tras el procesamiento de los datos.

Para pasar de un formato a otro tenemos que codificar un algoritmo que realice lo siguiente:

- Primero insertar la columna de marcas de tiempo.
- Luego leer el campo número de objetos y dentro del campo que aglutina todo lo relacionado con objetos ser capaces de leer el número de puntos o valores que contiene cada objeto.
- Imprimir una fila por cada punto de cada objeto de cada paquete, para ello se deberá mantener todas las columnas con el mismo valor salvo la de objeto y valor, es decir, vamos a detectar un paquete con n valores como n paquetes con el mismo número de paquete, misma etiqueta de tiempo, mismo

origen, mismo destino, misma longitud y mismo código de función y donde el objeto y el valor irá cambiando. Por ejemplo, si en el fichero de entrada teníamos una fila que identificaba un paquete con 3 objetos el primero con 1 punto, el segundo con 5 puntos, y el tercero con 3 puntos ahora vamos a tener 9 filas con 6 de las 8 columnas iguales y donde solo van a ir variando el valor o el objeto y el valor de una fila a otra.

El algoritmo se ha implementado usando el espacio de trabajo Pycharm, con el lenguaje Python apoyándonos en su librería Pandas [10]. Para formatear más cómodamente los datos la salida del disector es un fichero de extensión csv. Puede verse el código completo del script en el Anexo B. El funcionamiento del script que implementa el algoritmo es el mostrado en la Figura 5-13:

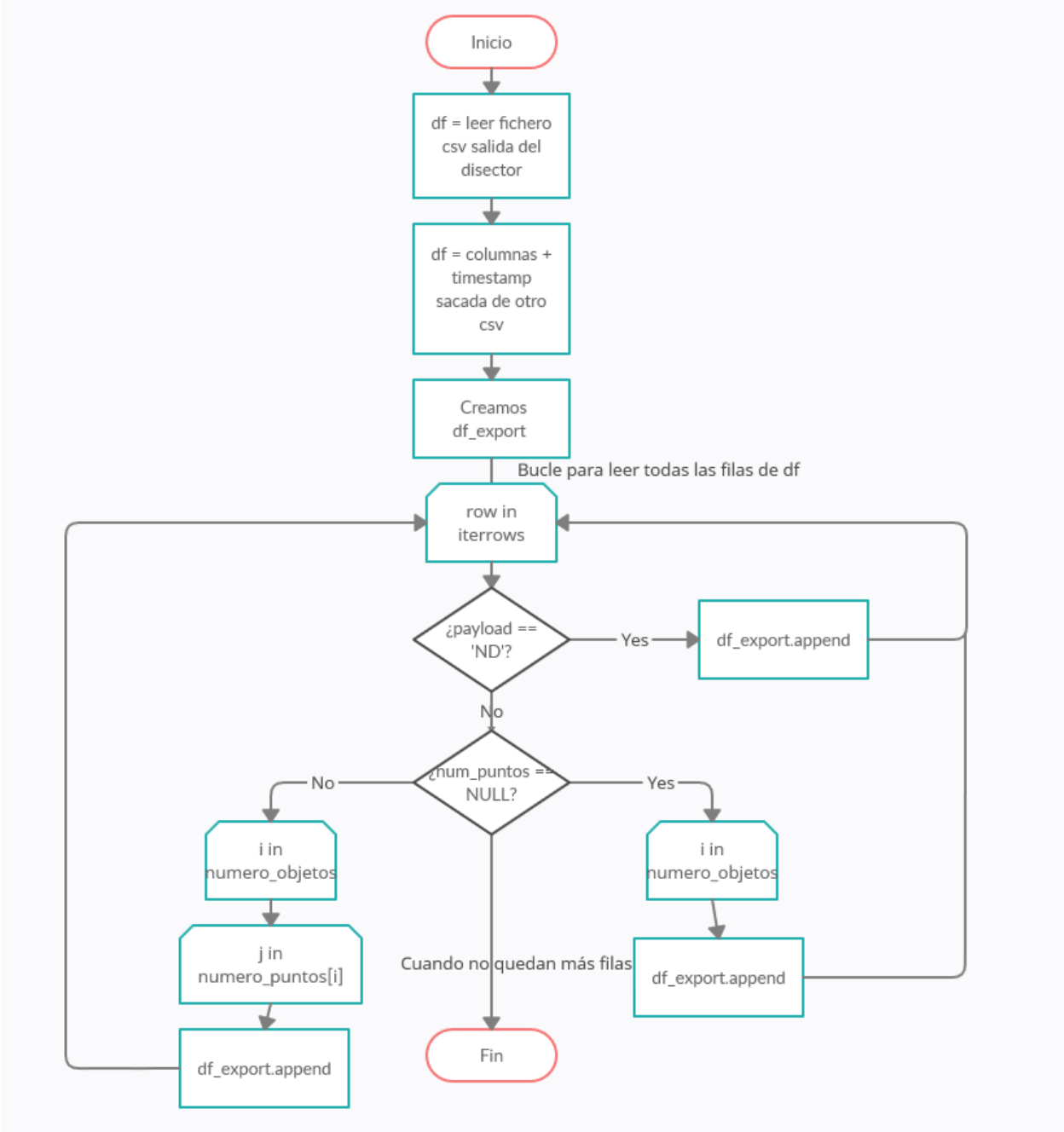

Figura 5-13: Diagrama de flujo del script de formateo de datos.

- Leemos el fichero resultado del disector y lo guardamos en una variable llamada “df” (Figura 5-14).

La importancia de que el fichero fuera de extensión csv radica en que hay una función de la librería pandas "read_csv" a la que tú le proporcionas el fichero, el delimitador que separa los campos y automáticamente te genera una tabla de n filas x m columnas. A partir de aquí el fichero se va a tratar como una tabla de n filas x 7 columnas, además las columnas quedarán identificadas con el valor de los 7 campos de la primera fila del fichero proporcionado, ya que panda lo realiza así por defecto.

	NumeroPaquete	Origen	Destino	Longitud	CodigoFuncion	CampoVector	NumeroObjetos	Timestamp
0	1	4	3	10	Unsolicited Response	ND	0	1097501938.50484
1	2	3	4	8	Confirm	ND	0	1097501941.54702
2	3	3	4	18	Write	Time and Date (Obj:50 Var:01);ffffff...	1	1097501941.54782
3	4	4	3	10	Response	ND	0	1097501941.56913
4	5	3	4	17	Disable Spontaneus Messages	Class 1 Data (Obj:60 Var:02);Null;Cl...	3	1097502061.90550
5	6	4	3	10	Response	ND	0	1097502061.91209
6	7	4	3	10	Unsolicited Response	ND	0	1097502623.04742
7	8	4	3	10	Unsolicited Response	ND	0	1097504102.25740
8	9	3	4	8	Confirm	ND	0	1097504103.39794
9	10	3	4	18	Write	Time and Date (Obj:50 Var:01);ffffff...	1	1097504103.39872
10	11	4	3	10	Response	ND	0	1097504103.40907
11	12	3	4	17	Enable Spontaneus Messages	Class 1 Data (Obj:60 Var:02);Null;Cl...	3	1097504186.59230
12	13	4	3	10	Response	ND	0	1097504186.66711
13	14	4	3	76	Unsolicited Response	Time and Date CTO (Obj:51 Var:01);...	3	1097504195.10626
14	15	3	4	8	Confirm	ND	0	1097504195.16278
15	16	4	3	71	Unsolicited Response	Time and Date CTO (Obj:51 Var:01);...	3	1097504196.56649
16	17	3	4	8	Confirm	ND	0	1097504196.57482
17	18	4	3	76	Unsolicited Response	Time and Date CTO (Obj:51 Var:01);...	3	1097504197.88773
18	19	3	4	8	Confirm	ND	0	1097504197.89904
19	20	4	3	71	Unsolicited Response	Time and Date CTO (Obj:51 Var:01);...	3	1097504199.59708
20	21	3	4	8	Confirm	ND	0	1097504199.61954
21	22	4	3	76	Unsolicited Response	Time and Date CTO (Obj:51 Var:01);...	3	1097504200.71951
22	23	3	4	8	Confirm	ND	0	1097504200.82128
23	24	4	3	50	Unsolicited Response	Time and Date CTO (Obj:51 Var:01);...	2	1097504202.51361
24	25	3	4	8	Confirm	ND	0	1097504202.56331

df Format: %s

Figura 5-14: Variable df.

- Leemos el fichero de marcas de tiempo (Figura 5-15) de cada paquete DNP3 y lo añadimos a la variable "df" como si fuera otra columna más. También añadimos otra columna "AnomalyDB" inicializada toda a false que nos será útil más tarde cuando estemos detectando anomalías.

	Timestamp
0	1097501938.50484
1	1097501941.54702
2	1097501941.54782
3	1097501941.56913
4	1097502061.90550
5	1097502061.91209
6	1097502623.04742
7	1097504102.25740
8	1097504103.39794
9	1097504103.39872
10	1097504103.40907
11	1097504186.59230
12	1097504186.66711
13	1097504195.10626
14	1097504195.16278
15	1097504196.56649
16	1097504196.57482
17	1097504197.88773
18	1097504197.89904
19	1097504199.59708
20	1097504199.61954
21	1097504200.71951
22	1097504200.82128
23	1097504202.51361
24	1097504202.56331
25	1097504203.32424
26	1097504203.42690
27	1097504204.66306

Figura 5-15: Columna de marcas de tiempo de los paquetes.

- Creamos un dataframe con la estructura de representación de datos comentada anteriormente de

nombre “df_export”. Va a ser una tabla de 8 columnas.

- Realizamos un bucle que recorra la tabla “df” fila por fila. Por cada fila almacenamos en 8 variables los 8 campos y a partir de ahí:
  - Cuando el CampoVector vale “ND” significa que estamos ante un paquete sin objetos, por lo cual directamente añadimos una fila a “df_export” haciendo un append a esta variable indicando el valor de cada campo. Los campos objetos y valor en este tipo de filas se ponen a ‘ND’.
  - Cuando el paquete contiene objetos, obtenemos el número de objetos y obtenemos un array con el valor del número de puntos que trae cada objeto. Si el array está vacío significa que estamos ante un mensaje con objetos que contienen 1 valor o punto y añadimos una fila a “df_export” por cada objeto, recorriendo un bucle de número de objetos iteraciones. Por el contrario, si el array no está vacío tendremos n objetos donde cada objeto tendrá m puntos. Aquí añadiremos una fila a “df_export” (Figura 5-16) por cada punto o valor de cada objeto, recorriendo un bucle de número de objetos iteraciones y otro bucle de número de puntos para un objeto iteraciones.

	NumeroPaquete	Timestamp	Origen	Destino	Longitud	CodigoFuncion	Objeto	Valor
0	1	1097501938.50484	4	3	10	Unsolicited Response	ND	ND
1	2	1097501941.54702	3	4	8	Confirm	ND	ND
2	3	1097501941.54782	3	4	18	Write	Time and Date (Obj:50 Var:01)	ffffffffffffff00
3	4	1097501941.56913	4	3	10	Response	ND	ND
4	5	1097502061.90550	3	4	17	Disable Spontaneus Messages	Class 1 Data (Obj:60 Var:02)	Null
5	5	1097502061.90550	3	4	17	Disable Spontaneus Messages	Class 2 Data (Obj:60 Var:03)	Null
6	5	1097502061.90550	3	4	17	Disable Spontaneus Messages	Class 3 Data (Obj:60 Var:04)	Null
7	6	1097502061.91209	4	3	10	Response	ND	ND
8	7	1097502623.04742	4	3	10	Unsolicited Response	ND	ND
9	8	1097504102.25740	4	3	10	Unsolicited Response	ND	ND
10	9	1097504103.39794	3	4	8	Confirm	ND	ND
11	10	1097504103.39872	3	4	18	Write	Time and Date (Obj:50 Var:01)	ffffffffffffff00
12	11	1097504103.40907	4	3	10	Response	ND	ND
13	12	1097504186.59230	3	4	17	Enable Spontaneus Messages	Class 1 Data (Obj:60 Var:02)	Null
14	12	1097504186.59230	3	4	17	Enable Spontaneus Messages	Class 2 Data (Obj:60 Var:03)	Null
15	12	1097504186.59230	3	4	17	Enable Spontaneus Messages	Class 3 Data (Obj:60 Var:04)	Null
16	13	1097504186.66711	4	3	10	Response	ND	ND
17	14	1097504195.10626	4	3	76	Unsolicited Response	Time and Date CTO (Obj:51 Var:01)	-3067125-121-10
18	14	1097504195.10626	4	3	76	Unsolicited Response	Binary Input Change With Relative Time...	0 00
19	14	1097504195.10626	4	3	76	Unsolicited Response	Binary Input Change With Relative Time...	1 00
20	14	1097504195.10626	4	3	76	Unsolicited Response	Binary Input Change With Relative Time...	1 00
21	14	1097504195.10626	4	3	76	Unsolicited Response	Binary Input Change With Relative Time...	1 -133
22	14	1097504195.10626	4	3	76	Unsolicited Response	Binary Input Change With Relative Time...	0 -133
23	14	1097504195.10626	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Time (...	0000
24	14	1097504195.10626	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Time (...	0000

df_export Format: %s

Figura 5-16: Variable df_export.

Una vez tenemos los datos en un formato adecuado para su inserción en la base de datos tenemos que escoger una base de datos. La base de datos escogida es Mongodb. Para ello simplemente hay que:

- Conectarse al servicio de Mongodb (dirección, puerto, nombre de la base de datos y nombre de la colección), en nuestro caso se está ejecutando en local (dirección 127.0.0.1) y en el puerto 27017, la base de datos nombrada como “dnp3” y una colección nombrada como “Datos”.
- Guardar en una variable el resultado de leer un fichero csv con “read_csv”.
- Convertir esta variable a un diccionario JSON del tipo clave: valor. Podemos convertir la variable anterior usando el método “to_json” indicando que la queremos en modo clave valor con el parámetro “orient=’records’”.
- Insertar la variable convertida a JSON diccionario tipo clave-valor usando el método “insert_many”.

Este procedimiento se puede repetir para cualquier fichero resultado del disector de paquetes. Para observar el código de inserción de datos en la base de datos consulte el Anexo B.

5.4 Fase 3: Aplicación de técnicas de detección de anomalías a los datos almacenados

Una vez disponemos de la información almacenada en la base de datos es turno de extraer únicamente los valores de los objetos que nos interesen para su posterior estudio de anomalías. Para ello hemos realizado otro script. Este script consta principalmente de 3 partes.

La primera parte está destinada a la conexión con la base de datos que contiene la información y con la consulta de datos a dicha base de datos. Con este fin se ha definido una función que, pasándole la dirección de la máquina, el puerto, el nombre de la base de datos y el nombre de la colección devuelve un dataframe con el resultado de la consulta. Este resultado contiene n filas. A nosotros nos interesa obtener valores de un único tipo de objeto para poder analizarlo y además que estos estén ordenados por tiempo de obtención, para lo cual la consulta a la base de datos tiene el siguiente formato mostrado en la Figura 5-17:

```
filter={
    'Objeto': 'Time and Date (Obj:50  Var:01)'
}
sort=list({
    'Timestamp': 1
}.items())

result = client['dnp3']['Datos'].find(
  filter=filter,
  sort=sort
)
```

Figura 5-17: Consulta desde Python a Mongodb.

Cuando tenemos nuestros valores de estudio, toca aplicarle la técnica SMA para detectar si hay comportamientos anómalos. De esto se encarga la segunda parte. Para poder decir que un valor es anómalo este debe cumplir que es mayor que la suma de su media móvil con su desviación típica móvil multiplicada por un coeficiente de detección. Decimos media y desviación típica móvil porque no son de la serie de datos total, si no únicamente de los n últimos valores que los fijamos con el tamaño de ventana de estudio que queramos. Para calcular la media y desviación móvil usamos el método “rolling” de pandas al que se le pasa como argumento el tamaño de la ventana de estudio.

Posteriormente creamos una columna “anomaly” inicializada a false que nos va a indicar que dato es anómalo cuando valga true. Comprobamos si un dato es anómalo con el criterio anterior, y si es así cambiamos el valor de anomaly a true para esa fila.

Si bien hemos definido el criterio de valor anómalo como si el dato es superior a la suma de su media móvil con su desviación típica móvil y multiplicada por un coeficiente de detección, hay algo que falla en este concepto. Si nosotros comprobamos si un valor es mayor que el umbral definido incluyendo la modificación que este dato produce en el umbral, podemos estar ocultando un positivo, por ejemplo, si el umbral de nuestro detector es de 80 y de repente analizamos un valor de 120 y la ventana de estudio escogida es pequeña (5 por ejemplo), estaremos aumentando significativamente el valor del umbral y puede hacer que el valor de 120 esté incluido en ese nuevo umbral. Para evitar este comportamiento, debemos comprobar si el dato actual es anómalo comparándolo con el valor del umbral generado anterior a este dato. Para implementar esto en el script se ha utilizado el método “shift” que desplaza hacia arriba o hacia abajo los valores de una columna el número de posiciones que le indiques.

Por último, como tercera parte, para comprobar los resultados se hace una gráfica que muestra el valor para cada punto y el umbral que hay en ese momento. Si en algún momento la curva del valor supera a la del umbral, significa que nos encontramos ante un dato anómalo. Además, se calculan los falsos positivos (resultado en el que el modelo predice incorrectamente la clase positiva), los falsos negativos (resultado en el que el modelo predice incorrectamente la clase negativa), los verdaderos positivos (resultado en el que el modelo predice correctamente la clase positiva) y los verdaderos negativos (resultado en el que el modelo predice correctamente la clase negativa) comparando el valor de la columna "anomaly" con la columna "AnomalyDB" y analizando valor a valor el resultado. Este proceso lo realiza sólo el método "confusión_matrix". La matriz de confusión tiene la forma ilustrada en la Figura 5-18.

Figura 5-18: Matriz de confusión.

Tanto la segunda parte como la tercera parte se encierra en dos bucles que realizan este procedimiento para 4 valores de ventana para realizar la media y desviación y para 2 valores del coeficiente de detección. El código del script se puede consultar en el Anexo C.

6 PRUEBAS

Este capítulo se dedica a la descripción, diseño y ejecución de una prueba a la que se somete al sistema diseñado.

6.1 Diseño de la prueba

La prueba será realizada en una máquina virtual con sistema operativo Ubuntu 20.0.4 y con una memoria RAM de 4 GB. Para la realización de la prueba será necesario tener instalado: tranalyzer, pycharm y Mongodb.

La prueba consiste en poner a prueba el sistema desarrollado en este trabajo, y constará de los siguientes pasos:

1. Arrancar el servicio de Mongodb e iniciar y enlazar Mongodb Compass.
2. Comprobar que tenemos un fichero de extensión pcap en el directorio data.
3. Invocar y ejecutar el disector de paquetes desde la terminal mediante el uso de tranalyzer.
4. Comprobar la salida del disector.
5. Ejecutar el script que formatea los datos y los inserta en la base de datos.
6. Comprobar que el fichero formateado es correcto y que los datos se han insertado correctamente en la base de datos.
7. Ejecutar el script que extrae datos de la base de datos y les aplica la media móvil ponderada exponencial.
8. Comprobar que la consulta realizada es correcta.
9. Comprobar los resultados obtenidos y analizarlos para las diferentes ventanas y coeficientes de detección a los que se someten los datos.
10. Una vez realizada la prueba, como prueba adicional a la detección de anomalías se va a modificar desde Mongodb Compass un valor del contador y se va a poner a un valor superior a 500 que sería anómalo y el valor de “AnomalyDB” a true para ese paquete.

La Figura 6-1 presenta un esquema del sistema durante la realización de la prueba, pudiendo diferenciar cuales son las entradas y salidas de cada uno de los tres módulos.

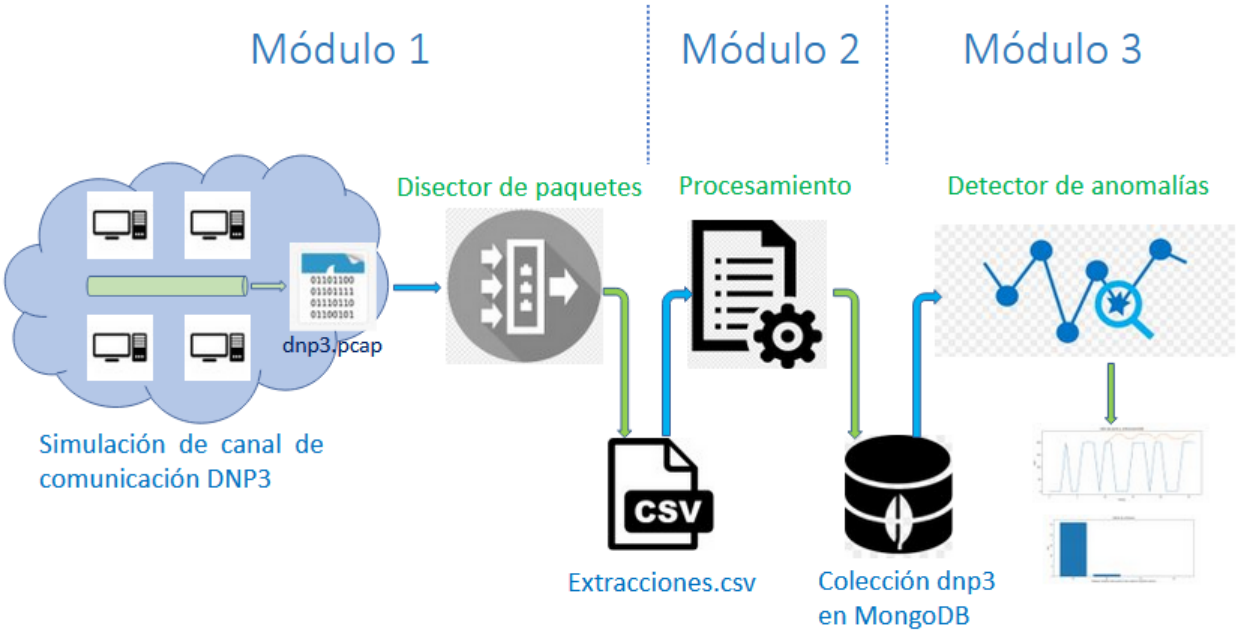

Figura 6-1: Esquema de la prueba realizada.

6.2 Ejecución y resultados

Se va a explicar cómo se realiza cada paso y a demostrar el resultado que se obtiene. Posteriormente se va a analizar y comparar las 3 opciones.

1. Para iniciar el servicio mongodb mirar la Figura 6-2:

```
dit@linux:~$ sudo systemctl start mongod.service
dit@linux:~$ sudo systemctl status mongod.service
● mongod.service - MongoDB Database Server
     Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2021-06-28 11:41:01 CEST; 9h ago
       Docs: https://docs.mongodb.org/manual
   Main PID: 679 (mongod)
     Memory: 120.4M
     CGroup: /system.slice/mongod.service
             └─679 /usr/bin/mongod --config /etc/mongod.conf

jun 28 11:41:01 localhost systemd[1]: Started MongoDB Database Server.
```

Figura 6-2: Arranque y estado del servicio mongod.service.

Para conectarse al servidor de Mongodb desde Mongodb Compass simplemente pulsar en Connect y te lleva a una pantalla de gestión de todas las bases de datos que se encuentran en ese servidor como la mostrada en la Figura 6-3.

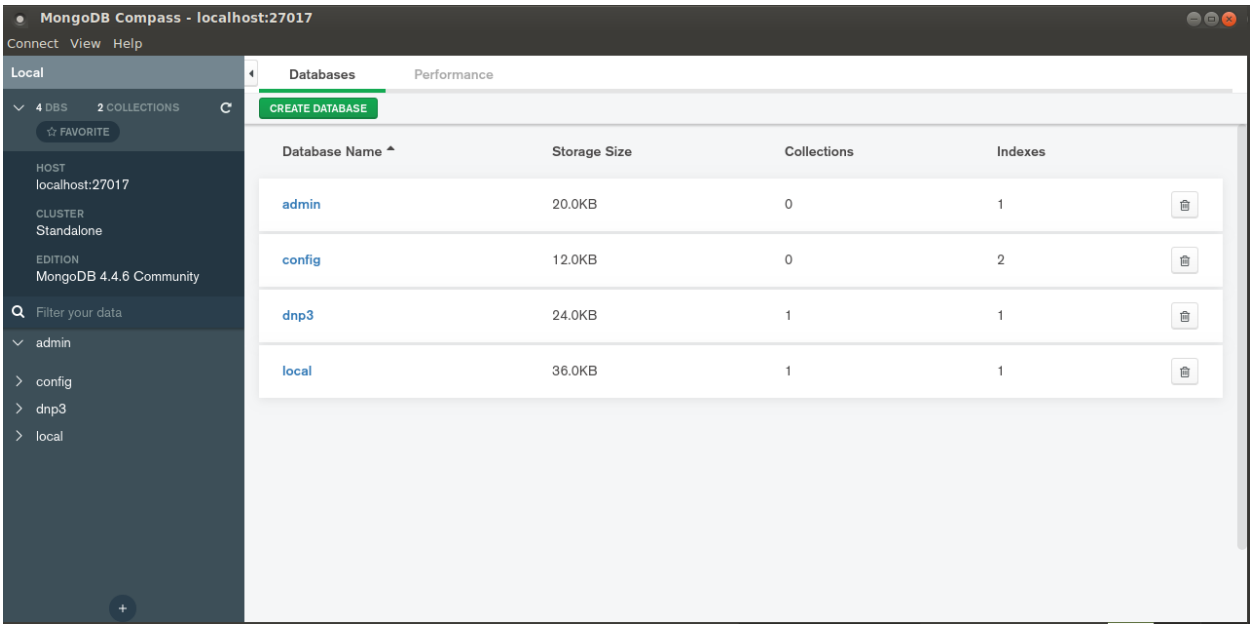

Figura 6-3: Pantalla de gestión de bases de datos de Mongodb Compass.

2. Introducir el fichero pcap que debe leer el disector de paquetes en la carpeta data de manera que quede como en la Figura 6-4.

Figura 6-4: Directorio data con el fichero de entrada al sistema.

3. El disector se invoca a través de la herramienta tranalyzer. Primero hay que compilar los plugins básicos de tranalyzer ejecutando el comando t2build, posteriormente compilar el plugin dnp3 ejecutando el comando t2build dnp3 y ejecutar el programa con el comando t2 -r data/fichero.pcap -w results/ -s. Con la opción -s le estamos indicando que queremos que tranalyzer se ejecute en modo paquete, lo cual invocará al código de nuestro disector. La figura 6-5 muestra la ejecución de los comandos citados.

Figura 6-5: Ejecución del disector de paquetes.

4. El fichero resultado es el esperado separando los campos definidos por “,” y los objetos y valores entre sí con “;” mostrado en la Figura 6-6.

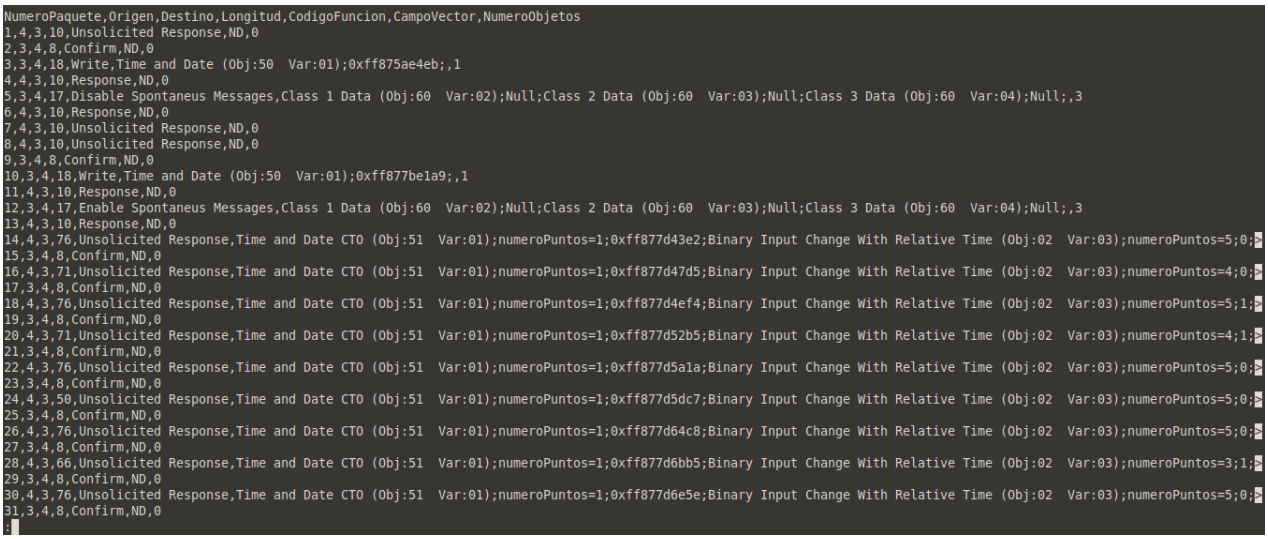

Figura 6-6: Parte del fichero resultado del disector de paquetes.

5. Para ejecutar el script que formatea los datos podemos hacerlo desde el entorno de desarrollo Pycharm pulsando en el triángulo verde.

6. El resultado es el ilustrado con la Figura 6-7:

```
NumeroPaquete,Timestamp,Origen,Destino,Longitud,CodigoFuncion,Objeto,Valor,AnomalyDB
1,2004-10-11 13:38:58.504844032,4,3,10,Unsolicited Response,ND,ND,False
2,2004-10-11 13:39:01.547020928,3,4,8,Confirm,ND,ND,False
3,2004-10-11 13:39:01.547816064,3,4,18,Write,Time and Date (Obj:50  Var:01),0xff875ae4eb,False
4,2004-10-11 13:39:01.569134080,4,3,10,Response,ND,ND,False
5,2004-10-11 13:41:01.905495808,3,4,17,Disable Spontaneus Messages,Class 1 Data (Obj:60  Var:02),Null,False
5,2004-10-11 13:41:01.905495808,3,4,17,Disable Spontaneus Messages,Class 2 Data (Obj:60  Var:03),Null,False
5,2004-10-11 13:41:01.905495808,3,4,17,Disable Spontaneus Messages,Class 3 Data (Obj:60  Var:04),Null,False
6,2004-10-11 13:41:01.912092928,4,3,10,Response,ND,ND,False
7,2004-10-11 13:50:23.047416832,4,3,10,Unsolicited Response,ND,ND,False
8,2004-10-11 14:15:02.257400064,4,3,10,Unsolicited Response,ND,ND,False
9,2004-10-11 14:15:03.397942016,3,4,8,Confirm,ND,ND,False
10,2004-10-11 14:15:03.398715904,3,4,18,Write,Time and Date (Obj:50  Var:01),0xff877be1a9,False
11,2004-10-11 14:15:03.409070080,4,3,10,Response,ND,ND,False
12,2004-10-11 14:16:26.592304000,3,4,17,Enable Spontaneus Messages,Class 1 Data (Obj:60  Var:02),Null,False
12,2004-10-11 14:16:26.592304000,3,4,17,Enable Spontaneus Messages,Class 2 Data (Obj:60  Var:03),Null,False
12,2004-10-11 14:16:26.592304000,3,4,17,Enable Spontaneus Messages,Class 3 Data (Obj:60  Var:04),Null,False
13,2004-10-11 14:16:26.667107072,4,3,10,Response,ND,ND,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,Time and Date CTO (Obj:51  Var:01),0xff877d43e2,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,Binary Input Change With Relative Time (Obj:02  Var:03),0,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,Binary Input Change With Relative Time (Obj:02  Var:03),1,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,Binary Input Change With Relative Time (Obj:02  Var:03),1,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,Binary Input Change With Relative Time (Obj:02  Var:03),1,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,Binary Input Change With Relative Time (Obj:02  Var:03),0,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,32-Bit Analog Change Event w/o Time (Obj:32  Var:01),0000,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,32-Bit Analog Change Event w/o Time (Obj:32  Var:01),0000,False
14,2004-10-11 14:16:35.106256896,4,3,76,Unsolicited Response,32-Bit Analog Change Event w/o Time (Obj:32  Var:01),0000,False
15,2004-10-11 14:16:35.162782976,3,4,8,Confirm,ND,ND,False
```

Figura 6-7: Fichero formateado por el script

Comprobamos desde Mongodb Compass que los datos se han insertado correctamente en la base de datos (Figura 6-8).

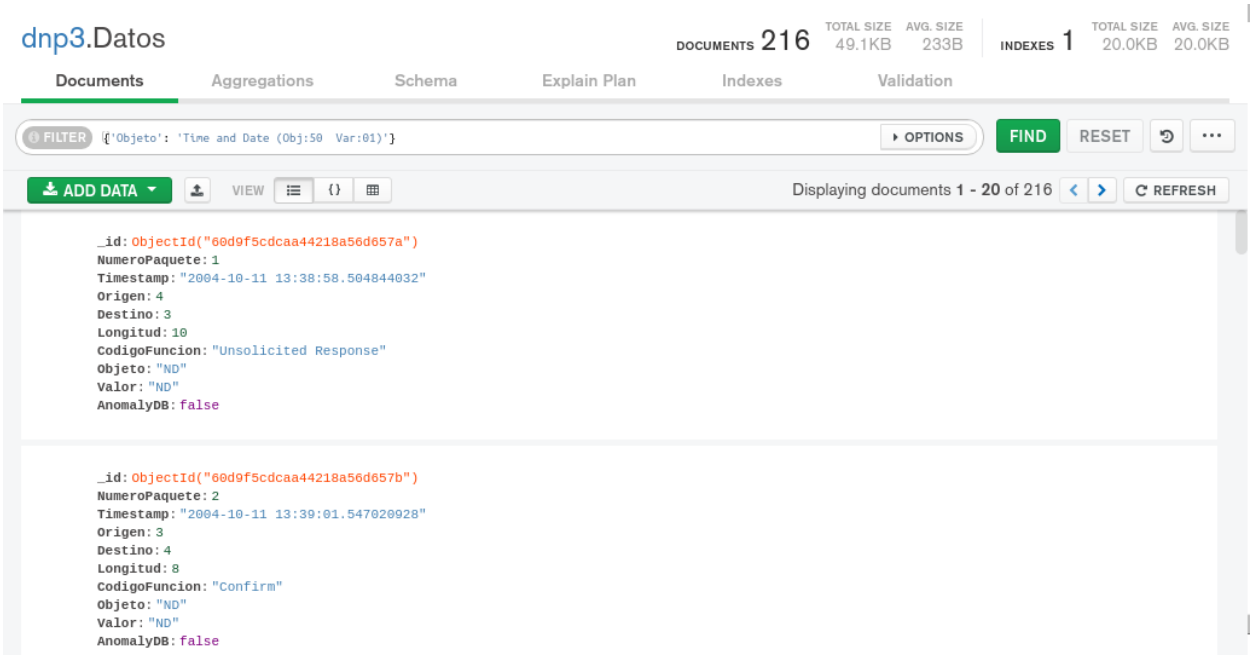

Figura 6-8: Información recogida en la colección Datos.

7. Para ejecutar el script que realiza la consulta a la base de datos y que posteriormente le aplica una técnica de detección de anomalía a esa información se puede ejecutar como el script del paso 5. Como consulta se van a pedir los valores del objeto "32-Bit Analog Change Event w/o Time (Obj:32 Var:01)", como tamaños de ventana se van a proporcionar los valores 3, 5, 8 y 10 y como coeficientes de detección se van a proporcionar los valores 1.05 y 1.10 lo que va a suponer un resultado de 8 gráficas.

8. El resultado de la consulta es el mostrado en la Figura 6-9.

	NumeroPaquete	Timestamp	Origen	Destino	Longitud	CodigoFuncion	Objeto	Valor	AnomalyDB
0	14	2004-10-11 14:16:35.106256896	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
1	14	2004-10-11 14:16:35.106256896	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
2	14	2004-10-11 14:16:35.106256896	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
3	16	2004-10-11 14:16:36.566493056	4	3	71	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000198	False
4	16	2004-10-11 14:16:36.566493056	4	3	71	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
5	16	2004-10-11 14:16:36.566493056	4	3	71	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
6	18	2004-10-11 14:16:37.887726080	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000198	False
7	18	2004-10-11 14:16:37.887726080	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000202	False
8	18	2004-10-11 14:16:37.887726080	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000198	False
9	20	2004-10-11 14:16:39.597084032	4	3	71	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
10	20	2004-10-11 14:16:39.597084032	4	3	71	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000202	False
11	20	2004-10-11 14:16:39.597084032	4	3	71	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000200	False
12	22	2004-10-11 14:16:40.719510144	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
13	22	2004-10-11 14:16:40.719510144	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
14	22	2004-10-11 14:16:40.719510144	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
15	26	2004-10-11 14:16:43.324244992	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000198	False
16	26	2004-10-11 14:16:43.324244992	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000199	False
17	26	2004-10-11 14:16:43.324244992	4	3	76	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000199	False
18	28	2004-10-11 14:16:44.663059968	4	3	66	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	0000	False
19	28	2004-10-11 14:16:44.663059968	4	3	66	Unsolicited Response	32-Bit Analog Change Event w/o Tim...	000202	False

df_query Format: %s

Figura 6-9: Resultado de la consulta a la base de datos.

9. El script genera una gráfica para cada tamaño de ventana y coeficiente de detección con todos los valores y con el umbral para cada punto de la manera que se explicó en el apartado 5.3. Además, genera una gráfica de barras que muestra los resultados de la matriz de confusión. Para no ilustrar las 16 gráficas totales se van a ilustrar las 2 para el tamaño de ventana 10 y coeficiente de detección 1.05 y se va a comparar el resultado de las demás gráficas mediante la Tabla 6-1 que represente el número de falsos positivos con respecto al tamaño de ventana y coeficiente de detección.

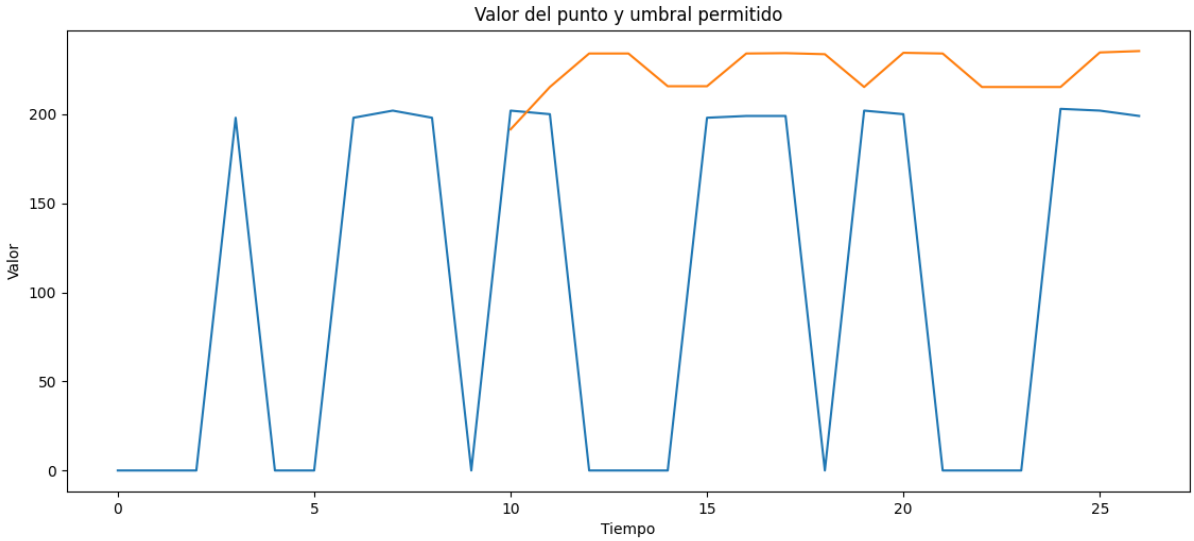

Figura 6-10: Gráfica de valores respecto a umbrales.

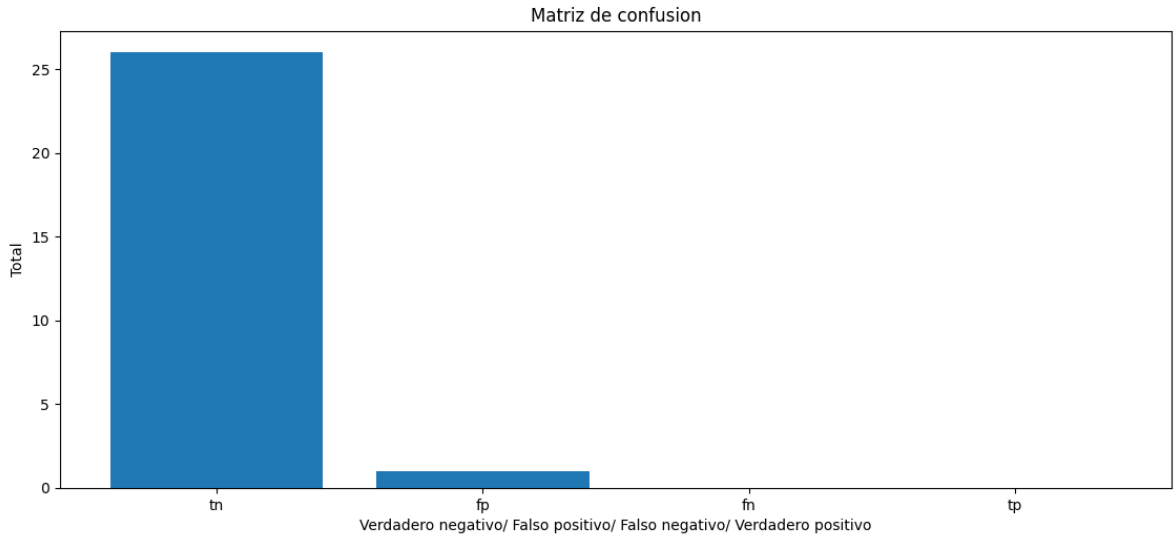

Figura 6-11: Gráfica de la matriz de confusión.

Como podemos observar en la Figura 6-10 hay un valor que se sitúan por encima del umbral, lo cual implicaría que es un valor anómalo. Todos los datos son correctos, como nos indicaba nuestra columna de "anomaly" de la base de datos, sin embargo, el detector dice que hay 1 valor que es anómalo como podemos ver en la Figura 6-11 (barra fp), esto ocurre porque estamos estudiando el valor de un contador, que o bien es 0 o bien tiene cualquier valor entre 0 y 500 y porque al tener una ventana pequeña el umbral llega a situarse en un valor muy bajo al encadenar varios 0 seguidos. El número total de muestras que se analizan son 27.

Tamaño de la ventana	Coeficiente de detección	Número de falsos positivos
3	1.05	7
3	1.1	5
5	1.05	6
5	1.1	1
8	1.05	1
8	1.1	1
10	1.05	1
10	1.1	1

Tabla 6-1: Falsos positivos en función del tamaño de ventana y coeficiente de detección.

La Tabla 6-1 nos muestra que conforme aumentamos el tamaño de la ventana detectamos menos falsos positivos ya que la media es más estable, pero a cambio podemos estar perdiendo alguna anomalía y conforme aumentamos el coeficiente de detección detectamos también menos positivos ya que el umbral aumenta su nivel, pero a cambio también podemos estar perdiendo alguna anomalía.

10. Si modificamos el valor a un dato incorrecto y cambiamos el campo AnomalyDB a true indicando así que es un valor anómalo reconocido (Figura 6-12) los resultados obtenidos para una ventana pequeña (Figura 6-13 y Figura 6-14) y para una ventana grande (Figura 6-15 y Figura 6-16) son:

```
1   _id: ObjectId("60db3b20e3da106aab146ed2")                          ObjectId
2   NumeroPaquete : 16                                                 Int32
3   Timestamp : "2004-10-11 14:16:36.566493056 "                       String
4   Origen : 4                                                         Int32
5   Destino : 3                                                        Int32
6   Longitud : 71                                                      Int32
7   CodigoFuncion : "Unsolicited Response "                            String
8   Objeto : "32-Bit Analog Change Event w/o Time (Obj:32  Var:01) "   String
9   Valor : "001999 "                                                  String
10  AnomalyDB : true                                                   Boolean
```

Figura 6-12: Modificación manual de un valor de la base de datos.

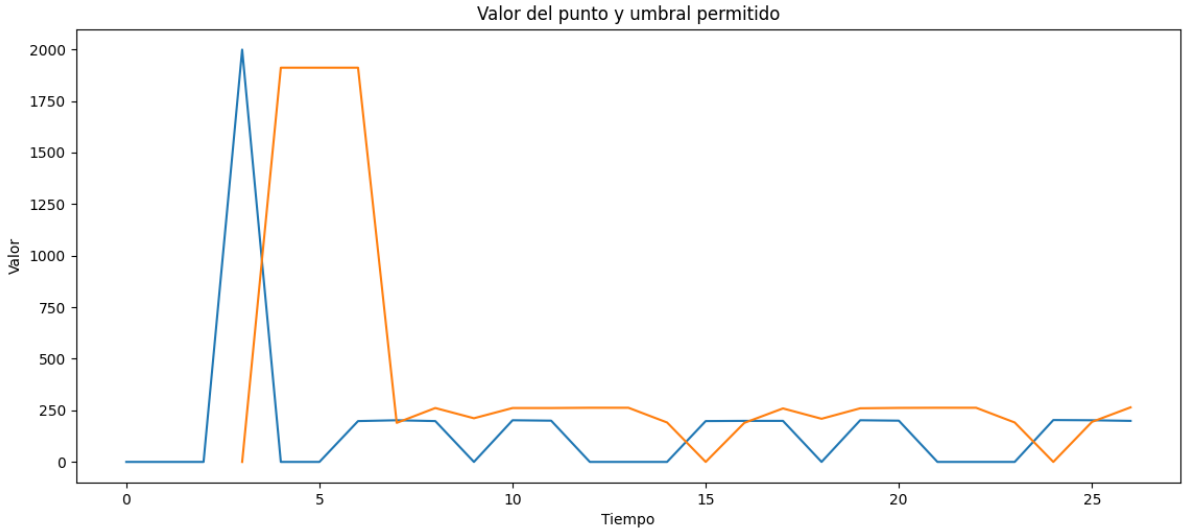

Figura 6-13: Valor anómalo forzado para ventana pequeña.

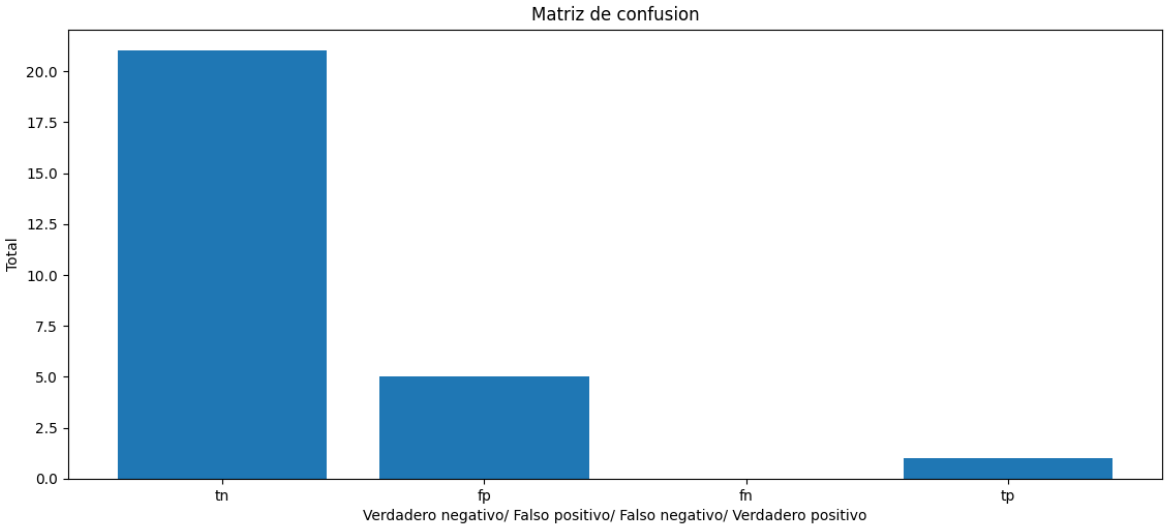

Figura 6-14: True positive detectado para ventana pequeña.

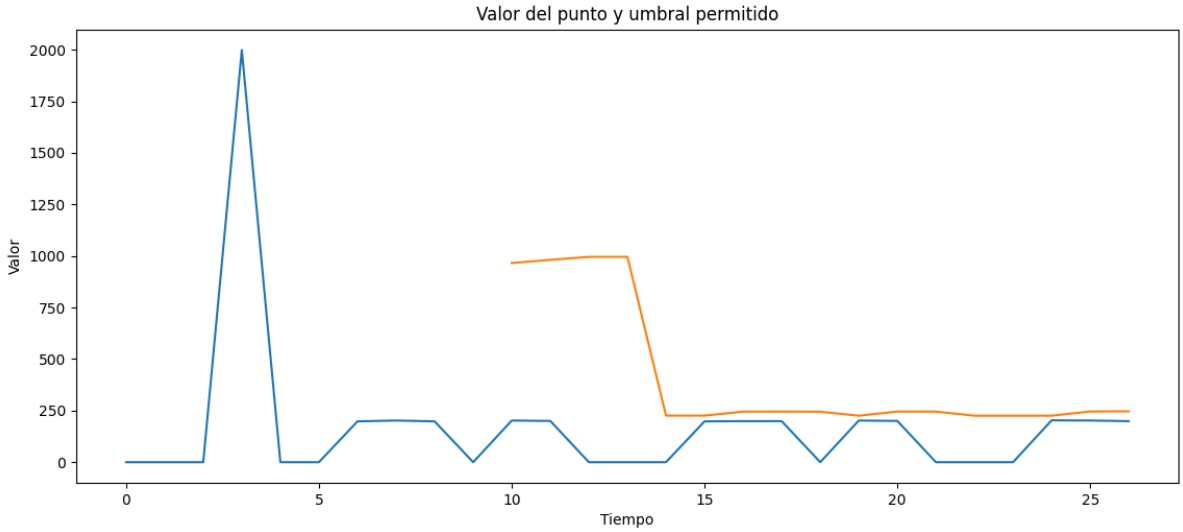

Figura 6-15: Valor anómalo forzado para ventana grande.

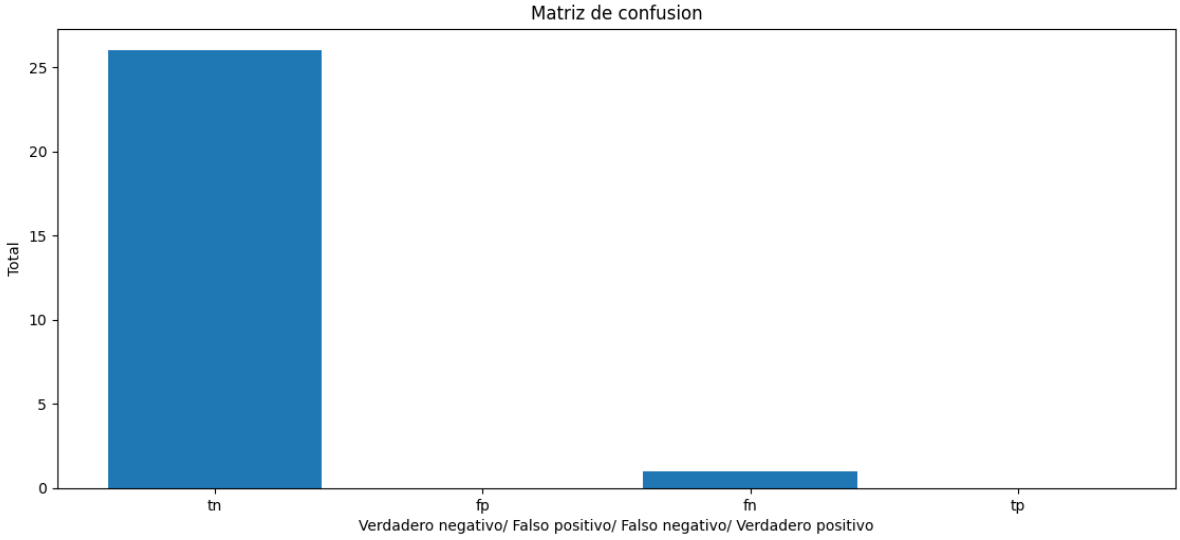

Figura 6-16: False negative detectado para ventana grande.

En el caso de la ventana grande, el dato afecta al umbral con el que se compara significativamente, porque la anomalía se produce antes de realizar el primer umbral, lo cual provoca que el sistema no detecte esa anomalía y por eso obtenemos un falso negativo.

En el caso de la ventana pequeña, el dato no afecta al umbral con el que se compara, ya que la anomalía se produce cuando ya tenemos un valor de umbral, lo cual provoca que el sistema detecte esa anomalía y por eso obtenemos un verdadero positivo.

7 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se van a exponer las conclusiones a las que se han llegado realizando este trabajo y por donde se puede completar y seguir investigando.

7.1 Conclusiones

Debido a los resultados obtenidos en las pruebas y las ideas sacadas durante el diseño y desarrollo del trabajo las conclusiones son:

- Las técnicas de detección de anomalías, más concretamente, las pertenecientes a la rama de las medias móviles permiten descubrir una gran cantidad de datos anómalos principalmente en información que tenga una serie temporal y un valor constante.
- El protocolo DNP3 como se investigó en la introducción contiene una serie de carencias de seguridad a nivel diseño que no se pueden proteger.
- La programación de un disector de paquetes en lenguaje C es una operación complicada que requiere de un alto nivel de programación a nivel de bits y de un conocimiento perfecto sobre el protocolo que se quiere diseccionar.
- Para trabajar con grandes cantidades de datos, debemos apoyarnos en herramientas como la librería pandas de Python que facilitan la tarea del programador para la gestión de información.
- Tras las pruebas realizadas se puede afirmar que el sistema completo funciona correctamente ya que ante un flujo de una comunicación DNP3 de entrada en formato pcap, nuestro sistema es capaz de primero extraer los valores de estudio de interés (gracias al disector), después procesar los datos y por último es capaz de detectar si algún dato de una variable concreta es anómalo o no, además de comprobar la fiabilidad del detector de anomalías mediante el uso de la matriz de confusión.

7.2 Líneas futuras

Debido a que el presente trabajo es un primer análisis en la detección de anomalías del protocolo DNP3 y que ha habido que realizar un sistema que presente los datos para poder analizarlos, el cual ha consumido la mayor parte del tiempo existen líneas de investigación y posibles mejoras sobre este proyecto. Estas son:

- Respecto al disector completar la impresión en el archivo de salida de todos los objetos, contemplando todos sus tipos de datos y mejora de la lectura de los objetos con valores devueltos de un único bit.
- Inclusión de la posibilidad del análisis de datos producidos en tiempo real, contemplando su paso por el disector, su formateo y su posterior introducción en la base de datos.
- Profundizar y estudiar más técnicas de detección de anomalías.
- Poner en funcionamiento el sistema sobre una red industrial para poder realizar estudios de fiabilidad del sistema.

REFERENCIAS

[1] B. Galloway and G. P. Hancke, "Introduction to Industrial Control Networks," in IEEE Communications Surveys & Tutorials, vol. 15, no. 2, pp. 860-880, Second Quarter 2013, doi: 10.1109/SURV.2012.071812.00124.

[2] Muhammad Rizwan Asghar, Qinwen Hu, Sherali Zeadally, Cybersecurity in industrial control systems: Issues, technologies, and challenges, Computer Networks, Volume 165, 2019, 106946, ISSN 1389-1286, https://www.sciencedirect.com/science/article/pii/S1389128619306292

[3] R. Amoah, S. Camtepe and E. Foo, "Securing DNP3 Broadcast Communications in SCADA Systems," in IEEE Transactions on Industrial Informatics, vol. 12, no. 4, pp. 1474-1485, Aug. 2016, doi: 10.1109/TII.2016.2587883.

[4] East, Samuel & Butts, Jonathan & Papa, Mauricio & Shenoi, Sujeet. (2009). A Taxonomy of Attacks on the DNP3 Protocol. Critical Infrastructure Protection III. 311. 10.1007/978-3-642-04798-5_5.

[5] S. Kwon, H. Yoo and T. Shon, "RNN-based Anomaly Detection in DNP3 Transport Layer," 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), 2019, pp. 1-7, doi: 10.1109/SmartGridComm.2019.8909701.

[6] Jurado Tabares, J. (2021). jacjurtab/TFG. GitHub. Retrieved 29 June 2021, from https://github.com/jacjurtab/TFG.

[7] nrodofile/ScapyDNP3_lib. GitHub. (2021). Retrieved 15 April 2021, from https://github.com/nrodofile/ScapyDNP3_lib.

[8] Combs, G. (2021). boundary/wireshark. GitHub. Retrieved 5 May 2021, from https://github.com/boundary/wireshark/blob/master/epan/dissectors/packet-dnp.c.

[9] Overview of DNP3 Protocol. Dnp.org. (2021). Retrieved 1 June 2021, from https://www.dnp.org/About/Overview-of-DNP3-Protocol.

[10] Donizetti Pérez Ortiz, D. (2011). Especificación del protocolo DNP3 utilizando un lenguaje de descripción formal. Universidad Tecnológica de la Mixteca.

[11] Vamos a conocer el protocolo DNP3 -. Campbellsci.es. (2021). Retrieved 15 May 2021, from https://www.campbellsci.es/blog/getting-to-know-dnp3.

[12] DNP object types supported. Vtscada.com. (2021). Retrieved 29 June 2021, from https://www.vtscada.com/help/Content/D_Tags/Dev_DNPObjTypes.htm.

[13] DNP3 Protocol Specifications Manual (for Emerson FB3000 RTUs). (2020) (1st ed.).

[14] PROTOCOL Translator DNP3 User Manual. (2007).

[15] Novacic, J., & Tokhi, K. (2019). Implementation of Anomaly Detection on aTime-series Temperature Data set.

[16] Z. Zhou and P. Tang, "Improving time series anomaly detection based on exponentially weighted moving average (EWMA) of season-trend model residuals," 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 2016, pp. 3414-3417, doi: 10.1109/IGARSS.2016.7729882.

[17] Tranalyzer - About. Tranalyzer.com. (2021). Retrieved 29 June 2021, from https://tranalyzer.com/about#theanteater.

[18] Tranalyzer - Tranalyzer2 Cheatsheet. Tranalyzer.com. (2021). Retrieved 29 June 2021, from https://tranalyzer.com/tutorial/cheatsheet.

[19] Tranalyzer - The Basics, your first flow plugin. Tranalyzer.com. (2021). Retrieved 29 June 2021, from https://tranalyzer.com/tutorial/buildyourownplugin.

[20] PyCharm: el IDE de Python para desarrolladores profesionales, por JetBrains. Jetbrains.com. (2021). Retrieved 29 June 2021, from https://www.jetbrains.com/es-es/pycharm/.

[21] ¿Qué es MongoDB?. (2021). Retrieved 29 June 2021, from https://www.mongodb.com/es/what-is-mongodb

[22] Yardley, T. (2021). ITI/ICS-Security-Tools. Retrieved 29 June 2021, from https://github.com/ITI/ICS-Security-Tools/tree/master/pcaps/dnp3

[23] Stouffer, K., Stouffer, K., Pillitteri, V., Lightman, S., Abrams, M., & Hahn, A. Guide to industrial control systems (ICS) security. NIST Special Publication 800-82

[24] Reading and Writing CSV Files in Python with Pandas. Stack Abuse. (2021). Retrieved 17 July 2021, from https://stackabuse.com/reading-and-writing-csv-files-in-python-with-pandas.

[25] Store CSV data into mongodb using python pandas. Gist. (2021). Retrieved 17 July 2021, from https://gist.github.com/mprajwala/849b5909f5b881c8ce6a#file-import_csv_to_mongo.

[26] Cuan, J. (2021). python — ¿Cómo importar datos de mongodb a pandas? It-swarm-es.com. Retrieved 17 July 2021, from https://www.it-swarm-es.com/es/python/como-importar-datos-de-mongodb-pandas/1071888036/.

[27] López-Avila, L., Acosta-Mendoza, N., gago-Alonso, A., López-Avila, L., Acosta-Mendoza, N., & gago-Alonso, A. (2021). Detección de anomalías basada en aprendizaje profundo: Revisión. Scielo.sld.cu. Retrieved 17 July 2021, from http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992019000300107.

[28] Hall, S. (2021). Similarities of Univariate & Multivariate Statistical Analysis. Sciencing. Retrieved 17 July 2021, from https://sciencing.com/similarities-of-univariate-multivariate-statistical-analysis-12549543.html.

[29] Zaragoza Gauchía, J. (2020). Riunet.upv.es. Retrieved 17 July 2021, from https://riunet.upv.es/bitstream/handle/10251/150529/Zaragoza%20-%20An%C3%A1lisis%20y%20comparaci%C3%B3n%20de%20algoritmos%20de%20detecci%C3%B3n%20de%20anomal%C3%ADas.pdf?sequence=1.

[30] Suarez, A. (2021). Diferencias entre el Machine Learning supervisado y no supervisado. Blog.bismart.com. Retrieved 17 July 2021, from https://blog.bismart.com/es/diferencias-machine-learning-supervisado-no-supervisado.

[31] Nikolaev, k. (2021). Media Móvil Ponderada - ¿Qué Es y Cómo Calcularla? Blog Earn2Trade. Retrieved 17 July 2021, from https://blog.earn2trade.com/es/media-movil-ponderada/.

[32] MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no). Genbeta.com. (2021). Retrieved 17 July 2021, from https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no.

ANEXO A: CÓDIGO DISECTOR DE PAQUETES

En este anexo se puede consultar la función claimLayer4Information desarrollada para implementar el disector de paquetes. Si se quiere consultar el código entero se puede ver en [6].

```
/*
 * This function is called for every packet with a layer 4.
 */
void claimLayer4Information(packet_t *packet, unsigned long flowIndex) {
    dnp3_flow_t * const dnp3FlowP = &dnp3_flows[flowIndex];


    if (!dnp3FlowP->stat) return; // not a dnp3 packet


    const dnp3_hdr_t *dnp = (dnp3_hdr_t*)packet->layer7Header;
    if (sPktFile) {

      if ((fich = fopen("extracciones_prueba.csv", "a")) == NULL){
        printf ( " Error en la apertura\n ");
      }
      else {
        uint8_t inicio1 = dnp->inicio[0];
        uint8_t inicio2 = dnp->inicio[1];
        uint16_t inicio = (inicio1 << 8) | inicio2;

        if (inicio == 0x0564){
          num_pqts++;

          char* codigo_funcion = getFCString(dnp->function_code);

          char carga_sin_crc[sizeof(dnp->carga_util)];

          int f = 0;
          int g = 0;
          //Bucle que elimina el crc añadido por dnp3 cada 16 bytes de carga
util
          while (f < dnp->len[0] - 8){  //tamaño de la carga util
            char bloque_datos[17];
            if (f == 0){
              f = 13;
              g = 13;
              memcpy(carga_sin_crc, &(dnp->carga_util[0]), 13);
            }
            else{
              memcpy(bloque_datos, &(dnp->carga_util[f+2]), 16);
              memcpy(&(carga_sin_crc[g]), bloque_datos, 16);
              f = f + 18;
```

```
            g = g + 16;
          }
        }

        //En este caso no hay objetos en el mensaje, solo cabecera, function
code y alomejor internals indication
        if (dnp->len[0] <= 10){

          //Es un mensaje de respuesta asi que hay internal indications
          if (dnp->function_code > 0x80){
            fprintf(fich, "%d,", num_pqts);
            fprintf(fich, "%d,", dnp->fuente);
            fprintf(fich, "%d,", dnp->destino);
            fprintf(fich, "%d,", dnp->len[0]);
            fprintf(fich, "%s,", codigo_funcion);
            fprintf(fich, "ND,");
            fprintf(fich, "0");
            fprintf(fich, "\n");
          }

          //Es un mensaje de solicitud, no hay internal indications
          else{
            fprintf(fich, "%d,", num_pqts);
            fprintf(fich, "%d,", dnp->fuente);
            fprintf(fich, "%d,", dnp->destino);
            fprintf(fich, "%d,", dnp->len[0]);
            fprintf(fich, "%s,", codigo_funcion);
            fprintf(fich, "ND,");
            fprintf(fich, "0");
            fprintf(fich, "\n");
          }
        }
        else{

          //Mensaje de solicitud con objetos
          if (dnp->function_code < 0x80){
            fprintf(fich, "%d,", num_pqts);
            fprintf(fich, "%d,", dnp->fuente);
            fprintf(fich, "%d,", dnp->destino);
            fprintf(fich, "%d,", dnp->len[0]);
            fprintf(fich, "%s,", codigo_funcion);

            int bytes_leidos = 0;
            int num_objetos_leidos = 0;
            while(bytes_leidos < dnp->len[0] - 8){
              char obj[2];
              memcpy(obj,&(carga_sin_crc[bytes_leidos]),2);
              char* object = getObjectString(obj[0],obj[1]);
              int tamanio_objeto = getObjectSize(obj[0],obj[1]);
```

```
                num_objetos_leidos++;

                fprintf(fich, "%s;", object);

                char clasificacion[1];
                memcpy(clasificacion,&(carga_sin_crc[2+bytes_leidos]),1);
                int prefix = getClasValue(clasificacion[0]);
                int rango= getRangeValue(clasificacion[0]);

                 //Si existe el campo rango
                if (rango != 0){
                  char campo_rango[rango];
                  memcpy(campo_rango,&(carga_sin_crc[3+bytes_leidos]), rango);

                  //Si lo que se expresa en el campo rango corresponde a la
cantidad de puntos del objeto
                  if (cantidad == true){
                    unsigned char valor[tamanio_objeto];

memcpy(valor,&(carga_sin_crc[3+rango+prefix+bytes_leidos]),tamanio_objeto);
                    printStaticObject(obj[0], obj[1], valor, fich);
                    bytes_leidos = bytes_leidos + 3 + rango + prefix +
tamanio_objeto;
                  }
                  //Si el campo rango no indica que hay mas de un punto en ese
objeto
                  else {
                    unsigned char valor[tamanio_objeto];
                    memcpy(valor,
&(carga_sin_crc[3+rango+prefix+bytes_leidos]), tamanio_objeto);
                    printStaticObject(obj[0], obj[1], valor, fich);
                    bytes_leidos = bytes_leidos + 3+ rango + prefix +
tamanio_objeto;
                  }

                }
                //Si no hay campo rango
                else {
                  unsigned char valor[tamanio_objeto];
                  memcpy(valor, &(carga_sin_crc[2+prefix+bytes_leidos]),
tamanio_objeto);
                  printStaticObject(obj[0], obj[1], valor, fich);
                  bytes_leidos = bytes_leidos + 3 + prefix + tamanio_objeto;
                }
              }
              fprintf(fich, ",%d", num_objetos_leidos);
              fprintf(fich, "\n");
              }

            //Mensaje de respuesta con objetos
```

```
            else {
              fprintf(fich, "%d,", num_pqts);
              fprintf(fich, "%d,", dnp->fuente);
              fprintf(fich, "%d,", dnp->destino);
              fprintf(fich, "%d,", dnp->len[0]);
              fprintf(fich, "%s,", codigo_funcion);

              int bytes_leidos = 2;
              int num_objetos_leidos = 0;
              while (bytes_leidos < dnp->len[0] - 8){
                char obj[2];
                memcpy(obj,&(carga_sin_crc[bytes_leidos]),2);
                char* object = getObjectString(obj[0],obj[1]);
                int tamanio_objeto = getObjectSize(obj[0],obj[1]);
                num_objetos_leidos++;

                fprintf(fich, "%s;", object);

                // Con el objeto Single-bit binary input hay un problema de
tratamiento de los bytes
                if (((obj[0] << 8) | obj[1]) == 0x0101){
                  char clasificacion[1];
                  memcpy(clasificacion,&(carga_sin_crc[2+bytes_leidos]),1);
                  int prefix = getClasValue(clasificacion[0]);
                  int rango= getRangeValue(clasificacion[0]);
                  fprintf(fich, "No implementado;");
                  bytes_leidos = bytes_leidos + 1 + prefix + rango;
                }
                else {
                  char clasificacion[1];
                  memcpy(clasificacion,&(carga_sin_crc[2+bytes_leidos]),1);
                  int prefix = getClasValue(clasificacion[0]);
                  int rango= getRangeValue(clasificacion[0]);

                  if (rango != 0){
                    char campo_rango[rango];
                    memcpy(campo_rango,&(carga_sin_crc[3+bytes_leidos]),
rango);
                    if (cantidad == true){
                      int num_objetos = campo_rango[0];
                      int i = 1;
                      int puntos_leidos = 0;
                      fprintf(fich, "numeroPuntos=%d;", num_objetos);
                      while (i <= num_objetos){
                        unsigned char valor[tamanio_objeto];

        memcpy(valor,&(carga_sin_crc[3+rango+prefix*i+bytes_leidos+puntos_leid
os*tamanio_objeto]),tamanio_objeto);
                        printStaticObject(obj[0], obj[1], valor, fich);
```

```
                        i++;
                        puntos_leidos++;
                      }
                      bytes_leidos = bytes_leidos +  (3 + rango +
prefix*num_objetos + tamanio_objeto*num_objetos);
                    }
                    else {
                      unsigned char valor[tamanio_objeto];
                      memcpy(valor,
&(carga_sin_crc[3+rango+prefix+bytes_leidos]), tamanio_objeto);
                      printStaticObject(obj[0], obj[1], valor, fich);
                      bytes_leidos = bytes_leidos + 3 + rango + prefix +
tamanio_objeto;
                    }

                  }
                  else {
                    unsigned char valor[tamanio_objeto];
                    memcpy(valor, &(carga_sin_crc[2+prefix+bytes_leidos]),
tamanio_objeto);
                    printStaticObject(obj[0], obj[1], valor, fich);
                    bytes_leidos = bytes_leidos + 3 + prefix + tamanio_objeto;
                  }
                }
              }
              fprintf(fich, ",%d", num_objetos_leidos);
              fprintf(fich, "\n");
            }
          }
        }
        fclose(fich);
      }
    }


    // only 1. frag packet will be processed
    if (!t2_is_first_fragment(packet)) return;

    numDNP3Pkts++;
        dnp3FlowP->nmp++;

}
```

ANEXO B: CÓDIGO SCRIPT FORMATEO

Este anexo incluye el código del script que realiza el formateo de los datos resultado del disector de paquetes y su posterior inclusión en la base de datos.

```
import pandas as pd
import re
import json
import pymongo

# Leemos el fichero salida del disector de paquetes DNP3
df = pd.read_csv('extracciones_prueba.csv', sep=',')

# Leemos el fichero de marcas de tiempo de cada paquete DNP3
colum_time = pd.read_csv('Timestamp.csv')

# Añadimos una columna a nuestro fichero resultado del disector que es la
marca de tiempo del paquete
df['Timestamp'] = colum_time
df['Timestamp'] = df['Timestamp'].multiply(1000)
df['Timestamp'] = pd.to_datetime(df['Timestamp'], unit='ms')

# Creamos otro dataframe que va a ser el fichero ya formateado
df_export =
pd.DataFrame(columns=['NumeroPaquete','Timestamp','Origen','Destino','Longitu
d','CodigoFuncion','Objeto','Valor'])

# Iteramos por cada paquete (cada fila)
for row in df.iterrows():

    #Extraccion a variable de todas las columnas del fichero original
    numpqt = row[1]['NumeroPaquete']
    time = row[1]['Timestamp']
    origen = row[1]['Origen']
    destino = row[1]['Destino']
    len = row[1]['Longitud']
    fc = row[1]['CodigoFuncion']
    payload = row[1]['CampoVector']

    # Si el campo payload está vacío significa que es un paquete sin objetos
    if (payload == 'ND'):

        # Para evitar los valores NaN por defecto los campos vacíos se
rellenan con la cadena ND
        # Añadimos al segundo dataframe creado (el fichero ya formateado) una
fila idéntica a la que había en el primer dataframe
```

```
        df_export = df_export.append({'NumeroPaquete': numpqt, 'Timestamp':
time, 'Origen': origen, 'Destino': destino, 'Longitud': len, 'CodigoFuncion':
fc, 'Objeto': 'ND',
         'Valor': 'ND'}, ignore_index=True)

    # Si el paquete contiene objetos
    else:

        # Averiguamos el numero de objetos que contiene ese paquete
        numObjetos = row[1]['NumeroObjetos']

        # Averiguamos el numero de puntos que obtiene cada paquete
        # Esta sentencia devuelve un array con el valor del entero que queda
encerrado entre la cadena 'numeroPuntos= ;'
        # de todas las veces (findall) que aparece en la variable payload
        num_puntos = re.findall('numeroPuntos=(\d?);', payload, re.DOTALL)

        # Separamos el campo payload en objeto, valor, valor, objeto ...
aprovechando que usamos un delimitador diferente
        objetos = payload.split(';')

        # Si nos encontramos ante un paquete de solicitud no va a haber
valores solo objetos
        if not num_puntos:
            j = 0
            for i in range(1, numObjetos + 1):
                df_export = df_export.append(
                    {'NumeroPaquete': numpqt, 'Timestamp': time, 'Origen':
origen, 'Destino': destino, 'Longitud': len,
                     'CodigoFuncion': fc, 'Objeto': objetos[j],
                     'Valor': objetos[j+1]}, ignore_index=True)
                j = j + 2
        else:
            k = 0
            for i in range(1, numObjetos + 1):
                for j in range (1, int(num_puntos[i - 1]) + 1):
                    df_export = df_export.append(
                        {'NumeroPaquete': numpqt, 'Timestamp': time,
'Origen': origen, 'Destino': destino, 'Longitud': len,
                         'CodigoFuncion': fc, 'Objeto': objetos[k],
                         'Valor': objetos[k + j + 1]}, ignore_index=True)
                k = k + int(num_puntos[i-1]) + 2

df_export.to_csv('formattedfile.csv', index=False)


# A continuación insertar en la base de datos en una nueva coleccion el
fichero formattedfile.csv
mng_client = pymongo.MongoClient('localhost', 27017)
```

```
mng_db = mng_client['dnp3']  # Replace mongo db name
collection_name = 'Datos'  # Replace mongo db collection name
db_cm = mng_db[collection_name]

data = pd.read_csv('formattedfile.csv')
data_json = json.loads(data.to_json(orient='records'))
db_cm.insert_many(data_json)
```

ANEXO C: CÓDIGO SCRIPT DETECCIÓN ANOMALÍAS

Este anexo incluye el código del script que realiza la consulta a la base de datos y que le aplica al resultado de esa consulta la técnica SMA de detección de anomalías.

```
# Consulta
filter = {
    'Objeto': '32-Bit Analog Change Event w/o Time (Obj:32  Var:01)'
}

# Realizamos la consulta a la base de datos
df_query = deteccionAnomalias.read_mongo('dnp3', 'Datos', filter,
'localhost', 27017)

# Deteccion de anomalias en un dato
# Ventana para el rolling
# ventana = 10

# Columna a analizar
columna = 'Valor'

# Convertimos a entero el valor de la columna valor
df_query['Valor'] = pd.to_numeric(df_query['Valor'])

# Porcentaje para detección
# k_deteccion = 1.05
############################

for ventana in [3, 5, 8, 10]:
    for k_deteccion in [1.05, 1.10]:

        # calcular media móvil y su varianza
        df_query['mean'] = df_query[columna].rolling(ventana).mean()
        df_query['std'] = df_query[columna].rolling(ventana).std()

        # desplazar media y desvianza
        df_query['mean'] = df_query['mean'].shift(1)
        df_query['std'] = df_query['std'].shift(1)

        # Añadir columna anomaly
        df_query['anomaly'] = False

        # Añadir columna umbral
        df_query.loc[:, 'umbral'] = (df_query['mean'] + df_query['std']) *
k_deteccion
```

```
        # Detección
        df_query.loc[df_query['Valor'] > (df_query['mean'] + df_query['std'])
* k_deteccion, 'anomaly'] = True

        # Comprobamos si los resultados han sido falsos positivos, falsos
negativos, verdaderos positivos o
        # verdaderos negativos
        try:
            tn, fp, fn, tp = confusion_matrix(df_query['AnomalyDB'],
df_query['anomaly']).ravel()
        except:
            tn = confusion_matrix(df_query['AnomalyDB'],
df_query['anomaly'])[0][0]
            fp = 0
            fn = 0
            tp = 0

        # Representacion de los resultados
        df_query['Valor'].plot()
        df_query['umbral'].plot()
        plt.title('Valor del punto y umbral permitido')
        plt.xlabel('Tiempo')
        plt.ylabel('Valor')
        plt.show()

        # Realizacion de matriz de confusion
        name = ['tn', 'fp', 'fn', 'tp']
        values = [tn, fp, fn, tp]
        plt.bar(name, values)
        plt.title('Numero de positvios detectados bien o mal')
        plt.xlabel('Tipo de positivo/negativo')
        plt.ylabel('Total')
        plt.show()
```

ANEXO D: TABLA TAMAÑOS DE OBJETOS DNP3

Este anexo incluye la tabla que nos indica el grupo, variación, descripción y tamaño de todos los objetos definidos en DNP3. Esta tabla ha sido vital para desarrollar el disector de paquetes.

Grupo	Variación	Descripción	Tamaño
01	00	Binary Input Default Variation	-
01	01	Single-bit Binary Input	1b
01	02	Binary Input With Status	1B
02	00	Binary Input Change Default Variation	-
02	01	Binary Input Change Without Time	1B
02	02	Binary Input Change With Time	7B
02	03	Binary Input Change With Relative Time	3B
03	00	Double-bit Input Default Variation	-
03	01	Double-bit Input No Flags	1b
03	02	Double-bit Input With Status	1B
04	00	Double-bit Input Change Default Variation	-
04	01	Double-bit Input Change Without Time	1B
04	02	Double-bit Input Change With Time	7B
04	03	Double-bit Input Change With Relative Time	3B
10	00	Binary Output Default Variation	-
10	01	Binary Output	1b
10	02	Binary Output Status	1B
11	00	Binary Output Change Default Variation	-
11	01	Binary Output Change Without Time	1B
11	02	Binary Output Change With Time	7B
12	01	Control Relay Output Block	4B
20	00	Binary Counter Default Variation	-

20	01	32-Bit Binary Counter	5B
20	02	16-Bit Binary Counter	3B
20	03	32-Bit Delta Counter	5B
20	04	16-Bit Delta Counter	3B
20	05	32-Bit Binary Counter Without Flag	4B
20	06	16-Bit Binary Counter Without Flag	2B
20	07	32-Bit Delta Counter Without Flag	4B
20	08	16-Bit Delta Counter Without Flag	2B
21	00	Frozen Binary Counter Default Variation	-
21	01	32-Bit Frozen Counter	5B
21	02	16-Bit Frozen Counter	3B
21	03	32-Bit Frozen Delta Counter	5B
21	04	16-Bit Frozen Delta Counter	3B
21	05	32-Bit Frozen Counter w/ Time of Freeze	11B
21	06	16-Bit Frozen Counter w/ Time of Freeze	9B
21	07	32-Bit Frozen Delta Counter w/ Time of Freeze	11B
21	08	16-Bit Frozen Delta Counter w/ Time of Freeze	9B
21	09	32-Bit Frozen Counter Without Flag	4B
21	10	16-Bit Frozen Counter Without Flag	2B
21	11	32-Bit Frozen Delta Counter Without Flag	4B
21	12	16-Bit Frozen Delta Counter Without Flag	2B
22	00	Counter Change Event Default Variation	-
22	01	32-Bit Counter Change Event w/o Time	5B
22	02	16-Bit Counter Change Event w/o Time	3B
22	03	32-Bit Delta Counter Change Event w/o Time	5B
22	04	16-Bit Delta Counter Change Event w/o Time	3B
22	05	32-Bit Counter Change Event with Time	11B
22	06	16-Bit Counter Change Event with Time	9B
22	07	32-Bit Delta Counter Change Event with Time	11B

22	08	16-Bit Delta Counter Change Event with Time	9B
23	00	Frozen Binary Counter Change Event Default Variation	-
23	01	32-Bit Frozen Counter Change Event	5B
23	02	16-Bit Frozen Counter Change Event	3B
23	03	32-Bit Frozen Delta Counter Change Event	5B
23	04	16-Bit Frozen Delta Counter Change Event	3B
23	05	32-Bit Frozen Counter Change Event w/ Time of Freeze	11B
23	06	16-Bit Frozen Counter Change Event w/ Time of Freeze	9B
23	07	32-Bit Frozen Delta Counter Change Event w/ Time of Freeze	11B
23	08	16-Bit Frozen Delta Counter Change Event w/ Time of Freeze	9B
30	00	Analog Input Default Variation	-
30	01	32-Bit Analog Input	5B
30	02	16-Bit Analog Input	3B
30	03	32-Bit Analog Input Without Flag	4B
30	04	16-Bit Analog Input Without Flag	2B
30	05	32-Bit Floating Point Input	2B
30	06	64-Bit Floating Point Input	4B
31	07	32-Bit Frozen Floating Point Input	2B
31	08	64-Bit Frozen Floating Point Input	4B
32	00	Analog Input Change Default Variation	-
32	01	32-Bit Analog Change Event w/o Time	5B
32	02	16-Bit Analog Change Event w/o Time	3B
32	03	32-Bit Analog Change Event w/ Time	11B
32	04	16-Bit Analog Change Event w/ Time	9B
32	05	32-Bit Floating Point Change Event w/o Time	3B
32	06	64-Bit Floating Point Change Event w/o Time	5B
32	07	32-Bit Floating Point Change Event w/ Time	9B
32	08	64-Bit Floating Point Change Event w/ Time	11B

33	05	32-Bit Floating Point Frozen Change Event w/o Time	3B
33	06	64-Bit Floating Point Frozen Change Event w/o Time	5B
33	07	32-Bit Floating Point Frozen Change Event w/ Time	9B
33	08	64-Bit Floating Point Frozen Change Event w/ Time	11B
40	00	Analog Output Default Variation	-
40	01	32-Bit Analog Output Status	5B
40	02	16-Bit Analog Output Status	3B
40	03	32-Bit Floating Point Output Status	5B
40	04	16-Bit Floating Point Output Status	3B
41	01	32-Bit Analog Output Block	4B
41	02	16-Bit Analog Output Block	2B
41	03	32-Bit Floating Point Output Block	4B
41	04	16-Bit Floating Point Output Block	2B
42	00	Analog Output Event Default Variation	-
42	01	32-Bit Analog Output Event w/o Time	5B
42	02	16-Bit Analog Output Event w/o Time	3B
42	03	32-Bit Analog Output Event w/ Time	11B
42	04	16-Bit Analog Output Event w/ Time	9B
42	05	32-Bit Floating Point Output Event w/o Time	3B
42	06	64-Bit Floating Point Output Event w/o Time	5B
42	07	32-Bit Floating Point Output Event w/ Time	9B
42	08	64-Bit Floating Point Output Event w/ Time	11B
50	00	Time and Date Default Variation	-
50	01	Time and Date	6B
50	02	Time and Date	6B
50	03	Last Recorded Time and Date	6B
51	01	Time and Date CTO	6B
51	02	Unsynchronized Time and Date CTO	6B
52	01	Time Delay Coarse	2B

52	02	Time Delay Fine	2B
60	01	Class 0 Data	0B
60	02	Class 1 Data	0B
60	03	Class 2 Data	0B
60	04	Class 3 Data	0B
70	03	File Control - Command	Undefined
70	04	File Control - Status	Undefined
70	05	File Control - Transport	Undefined
70	06	File Control - Transport Status	Undefined
80	01	Internal Indications	Undefined
110	xx	Octet string	Undefined
111	xx	Octet string event	Undefined