UNIVERSIDAD
NACIONAL
DE COLOMBIA

# Construcción de un protocolo de comunicaciones para el sistema TLÖN que permita la inclusión y monitoreo de dispositivos con limitaciones de gestión en un marco de referencia de administración de red

**Edgar Mauricio Tamayo García**

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2018

# Construcción de un protocolo de comunicaciones para el sistema TLÖN que permita la inclusión y monitoreo de dispositivos con limitaciones de gestión en un marco de referencia de administración de red

**Edgar Mauricio Tamayo García**

Tesis presentada como requisito parcial para optar al título de:
**Magister en Ingeniería de Telecomunicaciones**

Director:
Ph.D., Jorge Eduardo Ortiz Triviño

Línea de Investigación:
Redes y sistemas de telecomunicaciones
Grupo de Investigación:
TLÖN - Grupo de Investigación en Redes de Telecomunicaciones Dinámicas y Lenguajes de Programación Distribuidos

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2018

A mi esposa Jisela quien me inspira y da sentido a mi vida. Sigamos montando el viento...

# Agradecimientos

Quiero agradecer a mi esposa y mi hijo, quienes me apoyaron en la consecución de este trabajo, por regalarme parte de su tiempo y brindarme sonrisas cuando las cosas parecían no salir bien, este logro es de ellos también. A Dios, cuya esencia trasciende las fronteras de mi conocimiento. Es valorado el aporte de los profesores Jorge Ortíz, cuyo conocimiento y experiencia me ayudaron a ir por el rumbo adecuado, Oscar Agudelo quien alumbro el entendimiento de la ingeniería de protocolos, y Henry Zarate cuya contribución facilitó la solución de problemas durante la programación del protocolo. Los tres excelentes profesionales y mejores personas. A los miembros del grupo de investigación TLÖN, cuyos comentarios ayudaron a dar foco al desarrollo de este trabajo

# Resumen

La construcción de un protocolo de comunicaciones para gestión de dispositivos con limitaciones de monitoreo que operen en una red Ad Hoc deben soportan algunos de los desafíos naturales de este tipo de redes como auto-configuración, distribución y heterogeneidad. El protocolo diseñado ofrece un servicio de traslación de información de gestión asíncrona y no orientada a conexión entre dos protocolos, SNMP y UDP/SERIAL que tienen sintaxis y semántica diferente, cuyo diseño se basa en un modelo de conversión de protocolos mediante un máquina de estado finitas. El desarrollo de su servicio y funciones están orientadas a resolver problemas de gestión en dispositivos heredados y pequeñas máquinas en redes de sensores, el cual ofrece una solución integral que opera sin importar la topología de red en la cual se implemente, y donde se han teniendo en cuenta los requerimientos de usuario.
**Palabras clave: gestión de red, Ad Hoc, equipos heredados, protocolo, especificación de servicio, especificación de protocolo, SNMP, serial, conversión de protocolos**.

# Abstract

The construction of a communications protocol for the management of devices with monitoring limitations that operate in an ad hoc network must support some of the natural challenges of this type of networks, such as self-configuration, distribution and heterogeneity. The designed protocol offers a translation service of asynchronous management information and not oriented to connection between two protocols, SNMP and UDP / SERIAL that have different syntax and semantics, whose design is based on a model of protocol conversion through a state machine. finite The development of its service and functions are aimed at solving management problems in legacy devices and small machines in sensor networks, which offers a comprehensive solution that operates regardless of the network topology in which it is implemented, and where it has been implemented. consider the user requirements.
**Keywords: network management, Ad Hoc, legacy devices, protocol, service specification, protocol specification, SNMP, serial communication, protocol conversion**

# Contenido

# Lista de Figuras

# Lista de Tablas

# 1. Introducción

El proyecto del grupo de investigación TLÖN propone un esquema de computación inspirado en modelos sociales, basado en conceptos de justicia, inmanencia, paradigma, Estado, existencia y esencia. Su objetivo es prestar servicios a los usuarios sin importar los recursos que tienen. El sistema requiere un protocolo de gestión de red que permita obtener información de los sistemas y de los agentes.

Es por ello que en este trabajo se busca construir un protocolo de comunicaciones para el sistema TLÖN que permita la inclusión y monitoreo de dispositivos físicos y virtualizados con limitaciones de gestión en un marco de referencia de administración de red. Para ello se define la especificación del servicio y el protocolo, se construye una MIB para representar las principales características de los dispositivos bajo gestión, se realiza programación de su arquitectura y se valida su funcionamiento en diferentes escenarios de operación.

La gestión de red ofrece multiples opciones de implementación pero el desafío se encuentra en integrar dispositivos con limitaciones de recursos físicos o lógicos o que no soporten protocolos de gestión, por ejemplo pequeños sensores en una red Ad Hoc o sistemas heredados en una habitual red centralizada, los cuales, en su mayoría pueden ser controlados mediante comandos de tipo serial. Es aqui, donde el trabajo toma mayor relevancia ya que los resultados de esta investigación pueden ser usados para mejorar los procesos de operación, mantenimiento y aprovisionamiento en redes de telecomunicaciones que dispongan de este tipo de dispositivos, ampliando el tiempo de utilidad de componentes o reduciendo los costos de despliegue de un sistema de monitoreo.

El diseño de la solución considera un proceso de obtención de los requisitos de usuario para transformarlos en requisitos del sistema y del software. Se indaga acerca de las necesidades de los interesados, con lo cual se establecen relaciones y se hace una caracterización que permite definir riesgos y la manera en que son tratados, lo cual finalmente permite realizar cierta planificación para alcanzar el objetivo principal del desarrollo del protocolo.

Los desafíos de gestión de dispositivos heredados o en redes Ad Hoc pueden ser resumidos en un problema de incompatibilidad de procesos de comunicación que puede ser solucionado mediante la conversión de protocolos. A través de un modelo de conversores basado en máquinas de estado finitas se diseña una imagen de protocolo que es común para SNMP y

serial, la cual se toma como base para establecer la especificación del servicio y del protocolo, sus funciones, primitivas de servicio y formato de PDU. Dado que parte del servicio que ofrece el protocolo es SNMP, se construye una MIB para representar los objetos a gestionar, la cual se usa como sistema de almacenamiento temporal de información de gestión.

Con toda esa información se construyó una arquitectura de software del protocolo que además afronta algunos de los desafíos que imponen las redes dinámicas distribuidas. Su implementación se realizo en dos escenarios evaluando su desempeño tanto en redes con topología Ad Hoc como Infraestructura.

# 2. Objetivos

## 2.1. Objetivo General

Construir un protocolo de comunicaciones para el sistema TLÖN que permita la inclusión y monitoreo de dispositivos físicos y virtualizados con limitaciones de gestión en un marco de referencia de administración de red

## 2.2. Objetivos específicos

- Definir la representación de los aspectos del servicio, el protocolo y el mapeo mediante una técnica de descripción formal
- Diseñar la base de información de administración (MIB) para caracterizar las variables de cada uno de los dispositivos a gestionar, siguiendo los lineamientos de las recomendaciones RFC del protocolo SNMP u otro similar
- Construir el módulo funcional del protocolo conversor para la inclusión de dispositivos físicos y virtualizados en un marco de referencia de administración de red
- Validar en dos escenarios la implementación del protocolo de comunicación de gestión sobre la infraestructura del Sistema TLÖN

# 3. Estado del Arte

## 3.1. Sistema TLÖN

El grupo de investigación en redes de Telecomunicaciones Dinámicas y Lenguajes de Programación Distribuidos - TLÖN, propone un esquema de computación inspirado en modelos sociales, basado en conceptos de justicia, inmanencia, paradigma, Estado, existencia y esencia. Para implementar dichos modelos en sistemas computacionales, se usa un esquema de virtualización que opera en una tolología de red Ad Hoc con todas sus condiciones dinámicas, estocásticas e inalámbricas. Así, el sistema TLÖN se basa en el modelo de capas mostrado en la figura **3-1**, el cual está compuesto por la red Ad Hoc, la virtualización, el sistema multiagente y las aplicaciones específicas. Además, de forma transversal existe un lenguaje de programación del sistema, generando las interacciones en todas las capas [TLÖN, 2017].

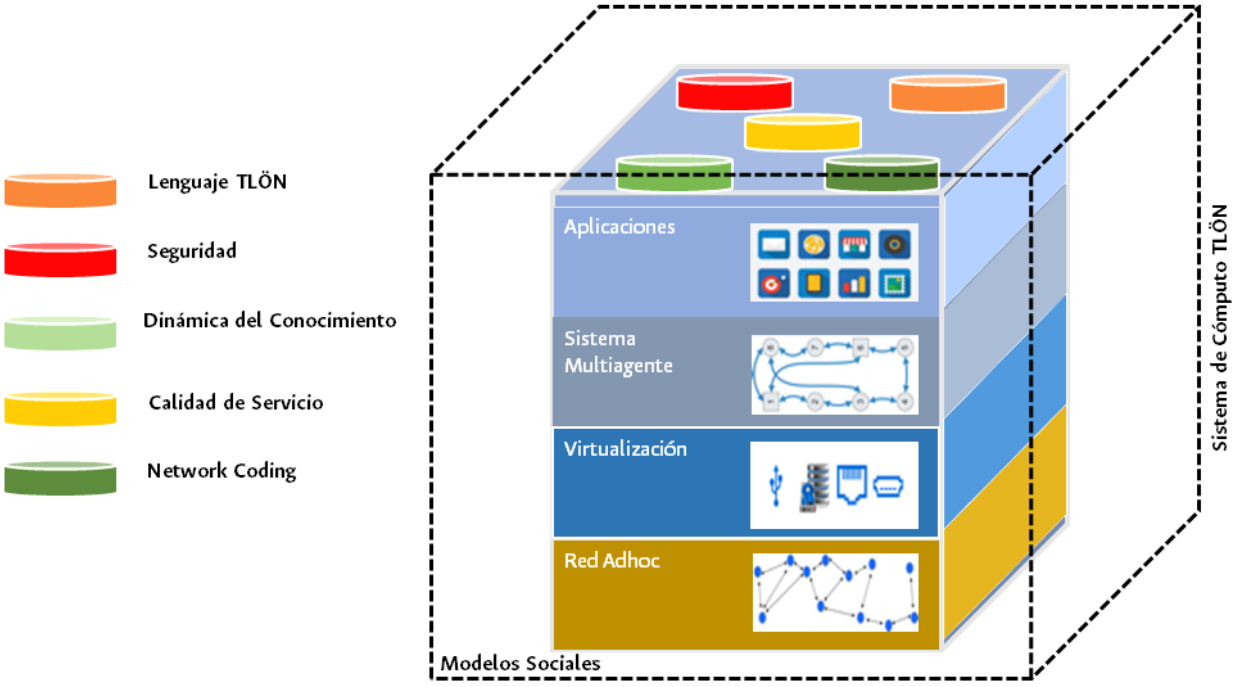


**Figura 3-1**.: Componentes del sistema TLÖN
Fuente [TLÖN, 2017]

El objetivo del sistema TLÖN es prestar servicios a los usuarios con el mismo grado de calidad sin importar los recursos que tienen. Además, al ser un sistema social inspirado, se debe tener un pseudo Estado que regule el comportamiento y las interacciones de los agentes, generando el concepto de nación [Zárate-Ceballos et al., 2015]. Esta analogía se puede ver en la figura **3-2**, en donde los componentes de un Estado son representados por elementos del sistema TLÖN.

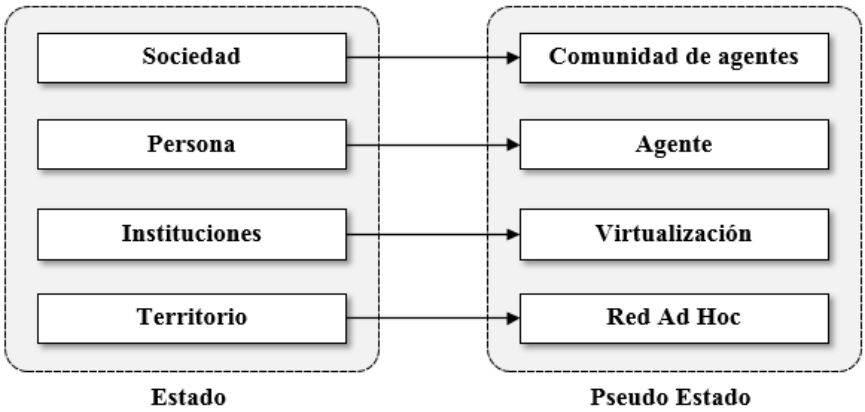


**Figura 3-2**.: Analogía de un Estado en el sistema TLÖN
Fuente [TLÖN, 2017]

El sistema TLÖN requiere un protocolo de gestión de red que permita obtener información de los sistemas y de los agentes. Este protocolo es análogo a una institución del Estado que ejerce control o verifica el comportamiento de una persona (agente) dentro de cierta sociedad (comunidad de agentes). Con la información capturada se pueden manejar los recursos (que son públicos para todos los agentes) de manera eficiente incluyendo la participación de cada nodo (ciudadanía).

## 3.2. Arquitectura de gestión de red

Los sistemas de gestión en una infraestructura de telecomunicaciones hacen referencia a las actividades, métodos, procedimientos y herramientas que permiten la operación, administración, mantenimiento y aprovisionamiento [Clemm, 2006] de la red. Estos sistemas permiten modelar ambientes de administración de red (haciendo referencia al modelo de gestión OSI) tales como organizacional, de información, funcional y de comunicaciones [Miller, 1999], cada uno cumpliendo un rol diferente pero complementario, en donde se controlan y monitorean diferentes variables de operación de los equipos y servicios que disponen la red, haciendo seguimiento y registrando el funcionamiento de los componentes del sistema, para detectar comportamientos inesperados que permitan acciones que prevengan fallas o mejoren el

desempeño de los servicios, controlar las desviaciones y administrar los recursos.

Un sistema de gestión de red se encuentra basado en el modelo agente/gestor que consiste un gestor, un sistema gestionado, una base de datos de información de gestión y un protocolo de gestión [Miller, 1999] donde se captura, controla y registra la información de los parámetros que se están monitoreando en el dispositivo, por ejemplo, el porcentaje de uso del procesador, la temperatura, el nivel de tráfico de una interfaz, entre otros. En la figura **3-3** se muestra este modelo de operación.

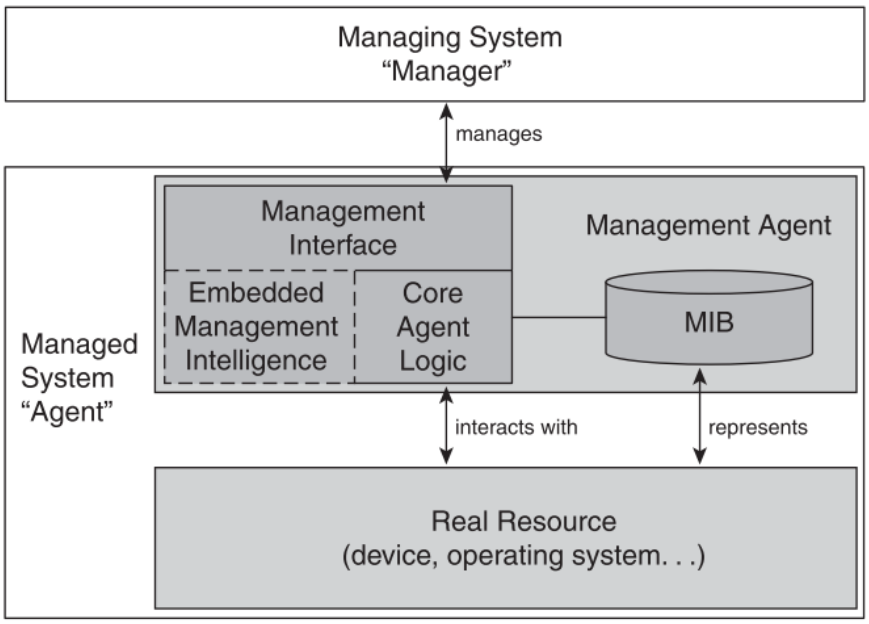


**Figura 3-3**.: Anatomía de un agente de gestión
Fuente [Clemm, 2006]

### 3.2.1. Protocolos de gestión

Para gestionar una red de telecomunicaciones se usan aplicaciones sobre las cuales se reúne toda la información proveniente de los dispositivos o servicios de la red a través de protocolos de gestión como WMI (Windows Management Instrumentation), CMIP (Common Management Information Protocol) o SNMP (Simple Network Management Protocol). Este último es el más utilizado, tanto en IPv4 como en IPv6, y hace parte del paquete de protocolos de Internet definidos por el IETF (Internet Engineering Task Force) bajo recomendaciones RFC (Request for comments) que describen su definición, estructura, arquitectura, sintaxis, tipos de aplicación, transporte de mensajes, coexistencia entre versiones y algunos más, que han sido actualizadas conforme el protocolo ha ido evolucionando.

Los parámetros de medición o variables de administración son representados mediante objetos independientes al protocolo SNMP, siendo vistos como una colección de objetos residentes en una base de gestión de información MIB (Management Information Base), la cual se encuentra definida bajo una estructura de gestión de información SMI (Structure Management Information) y con formato ASN.1 (Abstract Syntax Notation One) [Case et al., 1993].

Los objetos administrados tienen un identificador denominado OID (Object Identifier), que se convierte en un elemento único y detallado, el cual sigue un formato de árbol jerárquico SMI (ver figura **3-4**), el cual es escrito como una secuencia de números enteros separados por puntos [Miller, 1999], por ejemplo [1.3.6.1.2.1.1.1.0], que también tienen una representación con nombres [iso.org.dod.internet.mgmt.mib-2.system.sysDescr]. El OID hace parte de la información que constituye la MIB del dispositivo y que es utilizada por el gestor SNMP al realizar las consultas necesarias para obtener los datos deseados. Estos son registrados en la aplicación de gestión para su monitoreo, control y administración.

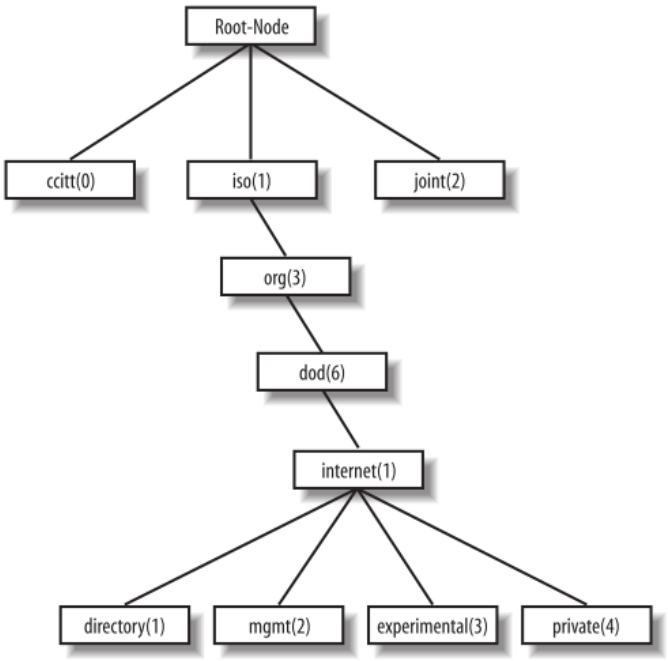


**Figura 3-4**.: Árbol de objetos SMI
Fuente [Mauro and Schmidt, 2005]

### 3.2.2. Arquitectura SNMP

La implementación de la arquitectura SNMP en su tercera versión se denomina entidad SNMP, la cual consta de un motor SNMP y una o más aplicaciones asociadas. El motor provee todos los servicios de envío y recepción de mensajes a través del despachador, con el modelo de procesamiento da soporte y tratamiento a los mensajes SNMP en todas sus versiones, ejecuta autenticación y codificación con el modelo de seguridad y control de acceso a los objetos gestionados. Las aplicaciones SNMP, que son definidas en el RFC3413 [Levi et al., 2002], hacen uso de los servicios provistos por el motor SNMP. Incluyen el generador de comandos (monitorea y manipula datos de gestión), respondedor de comandos (provee acceso a datos de gestión), originador de notificaciones (inicia mensajes asíncronos), receptor de notificaciones (procesa los mensajes asíncronos) y reenviador proxy (reenvío de mensajes entre entidades) En la figura **3-5**, se muestra la arquitectura descrita la cual es tenida en cuenta para la solución propuesta y descrita en la sección 5.5.

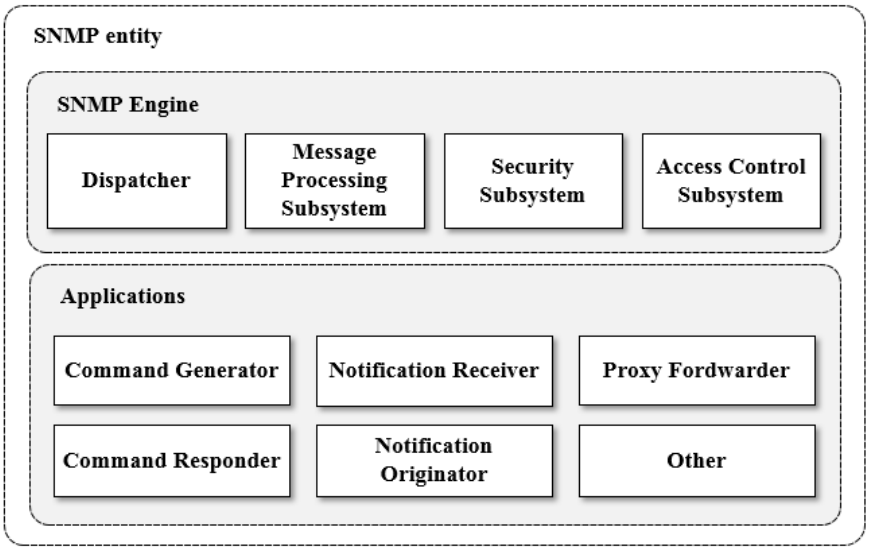


**Figura 3-5**.: Arquitectura de una entidad SNMP
Fuente [Harrington et al., 2002]

## 3.3. Comunicaciones seriales asíncronas

La gestión de red se hace a través de cualquier interfaz de comunicación instalada en el equipo a administar, como Ethernet, WiFi, fibra óptica o puerto serial. Los puertos seriales son interfaces que transmiten información un bit a la vez y generalmente usan protocolos asíncronos como RS232, RS485 o USB, ideales para comunicaciones entre sistemas embebidos, aunque también existen protocolos síncronos como SPI (Serial Periferical Interface), IIC (Inter-Integrated Circuit) o CAN (Controller Area Network) los cuales utilizan una señal de

reloj al transferir datos [Jiménez et al., 2014]. En los equipos de telecomunicaciones su utilización se ha extendido como método alternativo de control y diagnóstico. Algunos equipos, como los heredados (legacy sysmtems), solo pueden ser operados a través de este tipo de interfaces, con lo cual se generan algunas limitaciones en su gestión como se menciona en la siguiente sección.

Algunas de las ventajas de los puertos seriales asíncronos son el intercambio de cualquier tipo de información (a menudo se encuentran en sensores, switches, control de motores, relays, displays, entre otros), el hardware es económico, su formato no es complejo, soportan largos tramos de cable (a excepción de USB) y la tecnología inalámbrica lo soporta [Axelson, 2007]. Su principal limitación es la velocidad de transferencia de información.

## 3.4. El desafío

¿Qué ocurre si un dispositivo en una red de telecomunicaciones no soporta ningún protocolo de gestión? bien sea porque es un equipo heredado o debido a que es un sensor o un dispositivo pequeño con recursos limitados. La realidad es que funcionará y prestará sus servicios, sin embargo, cuando ocurran problemas en la red, no se podrá determinar su origen ni tomar medidas contingentes para solucionar o evitar que vuelvan a ocurrir los incidentes. De igual forma, dentro del sistema TLÖN tampoco se podrá determinar como un agente usa sus recursos y servicios

Se sabe que la información de desempeño de un equipo puede ser consultada mediante la ejecución de comandos a través de una interfaz de usuario que puede ser propietaria o por medio de aplicaciones de control remoto como telnet, SSH (Secure Shell) o acceso web, cuyos resultados muestran valores en tiempo real, pero con limitaciones de visualización histórica. También hay dispositivos de red operados por consola mediante conexiones seriales RS232 o RS485, incluso algunos de ellos, solo pueden ser controlados por este medio, haciéndolos incompatibles con cualquier protocolo de gestión. Ciertos desarrollos se enfocan en la instalación de software que trabaja en bajas capas de operación para ejecutar algunas funciones de administración como el descubrimiento de una topología de red [Nowicki and Malinowski, 2016]

Así, cuando se quiere llevar registro histórico de los atributos de operación los equipos, agentes o sistemas, solo puede hacerse de forma manual, con periodicidad indefinida y con las limitaciones que ello conlleva: datos insuficientes, registros poco confiables, posibles errores de digitación, imposibilidad de integrar la información capturada con los sistemas de gestión, nulidad en correlación de eventos, olvido en la captura de información por parte del operador, etcétera. Con esto, no se tiene la información de desempeño de los sistemas, la cual es vital para tomar decisiones sobre la administración y operación de la red de telecomunicaciones

de forma adecuada.

De esta forma, se observa que también existen retos en la integración de las diferentes aplicaciones que se usan tanto para gestionar y controlar, como para recopilar los datos de operación de los equipos en una infraestructura de comunicaciones, que de acuerdo a Clemm (2006), es una de las razones más importantes del por qué la administración de red puede ser difícil y, además, es usual que una red encontremos equipos de diferentes vendedores, ejecutando aplicaciones de gestión sobre cada uno de ellos (ver figura **3-6**). Estas consideraciones añaden factores de riesgo que hacen que la operación de los sistemas se torne más compleja de lo que debería ser.

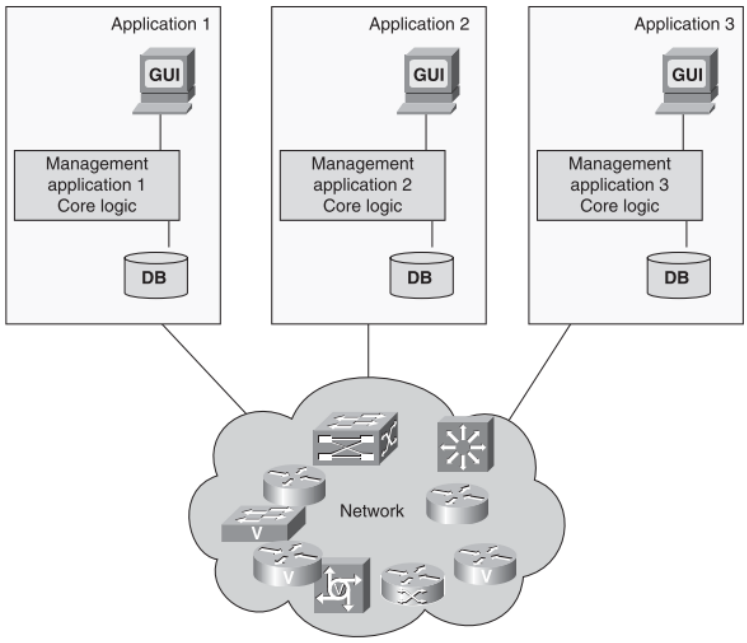


**Figura 3-6**.: Diferentes aplicaciones para gestionar una red
Fuente [Clemm, 2006]

Los anteriores desafíos pueden presentarse en escenarios tan distintos como en sistemas heredados o en redes Ad Hoc con procesamiento limitado, tales como sensores o dispositivos de Internet de las cosas (IoT).

### 3.4.1. Sistemas Herdados

Los sistemas heredados son aquellos dispositivos, aplicaciones o servicios que se encuentran en una etapa de "obsolescencia" o están siendo remplazados por nuevas tecnologías, pero que las empresas siguen utilizando porque han invertido mucho dinero en su compra, sus servicios

continúan activos, ya se ha retornado la inversión o porque es muy difícil su reemplazo por ser críticos en la operación [Sommerville, 2011]

La mayoría de investigaciones que se hacen para incluir, mantener o migrar los sistemas heredados están enfocadas en software y no tanto en hardware, quizás por el hecho que remplazar un dispositivo puede ser más sencillo que una aplicación específica, aunque no menos costoso. Sin embargo, se pueden asociar los resultados de dichas investigaciones para entender, por ejemplo, las cosas que contribuyen a un proceso exitoso de migración como costo, duración, defectos o capacidades [Huijgens et al., 2016]. En adición, la integración de los sistemas heredados tienen otros desafíos además de técnicos que podrían ser considerados como elementos de aceptación de los usuarios como la cultura, la calidad de la información, utilidad, facilidad de uso, compatibilidad, entre otros [Mathule and Kalema, 2016], y en donde los factores organizacionales son muy influyentes si la adquisición de nueva tecnología se da solo por tener las ventajas de la gestión mientras el resto de funciones de operación son similares a las tecnologías anteriores.

Un ejemplo de este reto se da en las estaciones terrenas satelitales, cuya topología se muestra en la figura **3-7**, que se encuentran en operación desde hace algunos años, cuyos componentes como transmisores, amplificadores, controladores de potencia, conversores y controladores de conmutación de señales de radiofrecuencia (RF) pueden carecer de monitoreo y no se encuentran dentro de la red de administración. Dichos dispositivos utilizan comunicaciones seriales para su control debido a que generalmente las antenas están alejadas de los equipos de banda base, superando las distancias permitidas por otro tipo de conexiones como Ethernet.

Una estación terrena puede ser operada local o remotamente, siempre que cuente con un diseño adecuado de monitoreo y control, lo cual permite a los administradores detectar, diagnosticar y resolver problemas técnicos, a su vez que facilita la ejecución de cambios o configuraciones [Elbert, 2001]. Las facilidades para hacer esto están incluidas dentro de los equipos, pero no son aprovechadas cuando hay limitaciones de integración a la infraestructura de gestión.

### 3.4.2. Redes Ad Hoc

Otro ejemplo de los desafíos de administración de dispositivos con limitaciones en gestión son las redes de sensores inalámbricas que hacen parte del Internet de las cosas, cuyos componentes tienen características de heterogeneidad, consumo reducido de energía y particularidades de enlace inalámbrico [Irastorza et al., 2006] que deben ser tenidas en cuenta para el desarrollo de cualquier aplicación. Un sensor dentro de una red Ad Hoc está diseñado para realizar una función específica y debido a sus factores de forma compactos tienen res-

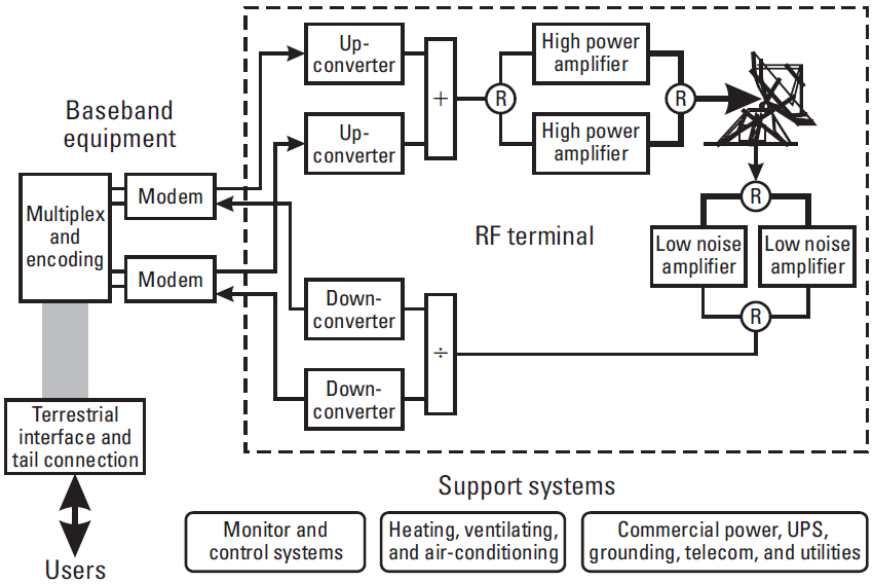


**Figura 3-7**.: Elementos que componen la mayoría de estaciones terrenas
Fuente [Elbert, 2001]

tricción de energía que limita su tiempo de vida o el de sus baterías. Favorablemente existen orientaciones de trabajo [Bhardwaj and Chandrakasan, 2002] que buscan solucionar esta limitación diseñando el nodo y su enlace inalámbrico tan eficiente como sea posible y usando una estrategia de colaboración entre nodos.

Integrando estos conceptos a nivel de la administración de red, se puede inferir que el enfoque de asignar óptimos roles a los sensores para hacerlos más eficientes puede dejar por fuera los procesos de gestión pero que puede existir un método colaborativo con un nodo auxiliar que apoye y se encargue de este servicio, para que los sensores procesen únicamente la información relevante para su aplicación.

Existen investigaciones enfocadas en el desarrollo de protocolos o sistemas de gestión para las redes Ad Hoc, como ANMP [Chen et al., 1999], GUERRILLA [Shen et al., 2002] o LiveNCM [Jacquot et al., 2010] que buscan mitigar el impacto de las condiciones propias de la operación de estas redes como el uso limitado de energía, las restricciones del medio inalámbrico, la heterogeneidad de los nodos, la autoconfiguración y autogestión. Además, los continuos desarrollos tecnológicos y científicos nos dan herramientas para atender los desafíos mencionados a su vez que aparecen nuevos retos y conceptos como el manejo de una arquitectura matemática denominada STEPS (Step Rate Storage) para ejecutar la gestión de almacenamiento colaborativo y control de rata de datos en WSN (Wireless Sensor Networks) [Kabashi and Elmirghani, 2010], el uso de blockchain para el desarrollo de sistemas de gestión distribuidos en MANET (Mobile Ad Hoc Networks) [Goka and Shigeno, 2018], o

el desarrollo de arquitecturas basadas en SDN (Software Defined Networks) y NFV (Network Function Virtualization) para monitorear las telemetrías de vehículos aéreos no tripulados (UAV) [White et al., 2016]. Al encontrar trabajos tan variados acerca de la gestión de la información en diferentes ámbitos significa que dichos datos son de vital importancia para el funcionamiento de la red y el servicio que presta.

### 3.4.3. Soluciones Existentes

En el mercado encontramos respuesta a los retos de administración y monitoreo de equipos de telecomunicaciones sin compatibilidad con los protocolos de gestión, desarrollando drivers para cada dispositivo y poniéndolos a funcionar sobre aplicaciones propietarias, tal es el caso de Compass[1], Dataminer[2], NetBoss XT[3] o NetVue[4], software cuyo costo es elevado ya que dependiendo de la cantidad de equipos en la red, los requisitos del desarrollo y las funciones de administración, su costo puede oscilar entre 100.000 y 250.000 dólares[5] y genera la dificultad de utilizar múltiples aplicaciones en los sistemas de gestión.

También existen soluciones con base en la conversión de diferentes protocolos a SNMP empleando conversores físicos de serial a Ethernet como ipConv[6], productos de Red Lion[7] o desarrollos en sistemas embebidos [Daogang et al., 2010], pero usados, en su mayoría, en redes de automatización industrial como SCADA (Supervisory Control And Data Acquisition), en las cuales se usan protocolos como Modbus, DNP3 (Distributed Network Protocol version 3), PROFIBUS (PROcess FIeld BUS), entre otros. El uso de estas soluciones permite migrar sistemas heredados seriales a redes Ethernet, pero tienen obstáculos de escalabilidad a causa de la utilización de un elemento de hardware por cada componente que se desee incluir en la red y como su uso no se ha extendido a equipos de telecomunicaciones, los proveedores deben hacer desarrollos para que den administración completa de los aparatos y sin las limitaciones que ofrecen los protocolos mencionados.

Estos son los motivos por los cuales la inclusión de los sistemas heredados o equipos con limitaciones de hardware, a una red de gestión es costosa, restringida y a menudo se evade.

[1] http://www.kratosnetworks.com/products/network-management/compass
[2] http://www.skyline.be/dataminer
[3] http://netboss.com/page/netboss-xt
[4] https://www.comtechefdata.com/products/network-bandwidth-management/netvue
[5] Información basada en cotizaciones reales con cada proveedor
[6] http://ipcomm.de/product/ipConv/en/sheet.html
[7] http://www.redlion.net/products/industrial-networking/communication-converters/

## 3.5. Conversión de Protocolos

Los desafíos señalados anteriormente, visto desde la conectividad lógica, pueden ser resumidos en un problema de incompatibilidad de procesos de comunicación y si añadimos las limitaciones de las soluciones actuales, logramos inferir que el problema puede ser tratado con un enfoque en software basado en la conversión de protocolos.

Existe una variedad de ejemplos de conversión de protocolos, pero no hay una teoría general que resuma dicho procedimiento, sin embargo, hay un importante desarrollo de modelos formales que pueden [Green, 1986] ser usados para la especificación y exactitud de la conversión, viendo la incompatibilidad de protocolos como un problema de sintaxis y semántica de los mensajes que son intercambiados por cada protocolo [Lam, 1988]. Ya que la conversión de protocolos puede darse en una enorme diversidad de ambientes, hay muchos métodos que dan diferentes enfoques de solución como la proyección del protocolo, el enfoque de Okumura, el enfoque cociente [Calvert and Lam, 1990] o basado en multi-capas del modelo OSI mediante la normalización del protocolo y sus requerimientos [Sinha, 2015]. Una opción específica para el protocolo SNMP es un agente SNMP proxy cuyo rol es traducir las solicitudes, respuestas y notificaciones de información de gestión y ejecutar el reenvío de dichos mensajes hacia el gestor, usando un formato adecuado [**?**].

## 3.6. Identificación del problema

Si bien, tanto la mayoría de equipos de telecomunicaciones como servidores de aplicaciones son desarrollados con soporte SNMP para su administración, se pueden encontrar algunos dispositivos que tienen acceso limitado a las variables de su sistema a través de este protocolo o en ocasiones ni siquiera lo soportan. Esto puede darse porque no fue considerado en el diseño inicial, debido a que los fabricantes desarrollaron aplicaciones propias para la gestión, son sistemas heredados cuyo reemplazo puede ser muy costoso a nivel financiero y operativo para una empresa o simplemente porque son sensores con sistemas operativos que no soportan ningún protocolo de gestión . Hay soluciones para la administración de este tipo de equipos como usar un software independiente o registrar la información de forma manual. Esto genera inconvenientes de compatibilidad, integración, operación con múltiples aplicaciones, limitaciones en los procesos de operación de una compañía. En adición, los datos pueden ser insuficientes, con periodicidad indefinida, registros poco confiables, posibles errores de digitación, imposibilidad de integrar la información capturada con los sistemas de gestión, nulidad en correlación de eventos, olvido en la captura de información por parte del operador, etcétera. De esta forma, no se tiene la información de desempeño de los sistemas, la cual es vital para tomar decisiones sobre la administración y operación de una red de telecomunicaciones de forma adecuada. La ventaja es que tanto los equipos de telecomuni-

caciones como los sensores, tienen en su mayoría, una interfaz serial para conexión local, la cual puede ser aprovechada para el desarrollo de aplicaciones de diferente índole.

## 3.7. Enfoque de la investigación

Entendiendo la necesidad de administrar aquellos equipos de telecomunicaciones que solo pueden ser operados mediante conexión serial o dispositivos que tienen limitación para soportar algún protocolo de gestión, esta investigación se enfoca en saber si es posible desarrollar un protocolo de comunicaciones conversor serial a SNMP basado en software para integrar dichos elementos a la red de gestión, cuál sería la forma adecuada de implementación, cómo se resolverían los desafíos de diversidad, si alcanzaría a ser visto como una actualización a los sistemas heredados con una implementación de bajo costo y si podría ser escalable a otro tipo de interfaces o elementos heterogéneos en una red.

Para resolver estas preguntas y alcanzar los objetivos propuestos, el desarrollo ha de ser aplicado en dispositivos de distintos fabricantes y sensores inalámbricos para incluirlos en aplicaciones de gestión que existen en el mercado; el protocolo conversor deberá tener la suficiencia de relacionar los comandos de administración y operación del equipo, con una MIB donde se asigne un OID único para cada variable, y por supuesto; esta solución deberá dar la capacidad de capturar la información de desempeño, reportar eventos y modificar los parámetros de funcionamiento a través del protocolo SNMP.

La programación del protocolo conversor se hará mediante un lenguaje que debe ser seleccionado de acuerdo a las facilidades que se encuentran en términos de libreráas para conexiones serial, Ethernet y SNMP. Se evaluará el lenguaje formal de descripciones (FDL) para protocolos de red que se acomode mejor a las especificaciones del protocolo.

Para su implementación, se montará un laboratorio, usando equipos de operación y control de una estación terrena satelital, los cuales son sistemas heredados, de diferentes fabricantes y no tienen conexión IP ni gestión SNMP, cumpliendo con todos los desafíos que componen el problema que se busca resolver. Estos se conectarán a un concentrador serial y este a su vez a la red de datos IP. También se incluirá una red de sensores inalámbricos para validar el funcionamiento de la escalabilidad del protocolo sobre otros ambientes. Las pruebas se realizarán con varias aplicaciones de gestión SNMP para comprobar que cumple con los requisitos del estándar.

De igual forma, el producto de esta tesis apoyará el desarrollo del proyecto del grupo de investigación en redes de telecomunicaciones dinámicas y lenguajes de programación distribuidos – TLÖN, incorporando el protocolo al sistema de cómputo TLÖN, el cual carece de

elementos de gestión y administración de red y que dentro de su modelo (ver figura **3-1**) servirán de soporte a los componentes Ad Hoc y a los dispositivos virtualizados.

# 4. Definición de Requerimientos

Con el análisis de requerimientos se busca definir las características y contexto de uso del protocolo y su servicio así como sus limitaciones, conociendo las necesidades de los usuarios y haciendo trazabilidad y validación. Para ello, se ha definido un proceso apoyado en el estándar ISO/IEC/IEEE 29148:2011 que consiste en la obtención de requisitos de usuario para luego transformarlos en requisitos del sistema y del software.

## 4.1. Requerimientos de usuario

En esta etapa del proceso se definen los requisitos para que el sistema pueda proporcionar los servicios que necesitan los usuarios y otras partes interesadas [ISO et al., 2011], mediante entrevistas informales, observación del ambiente operativo y revisión de documentación técnica de diferentes dispositivos.

La identificación de los interesados se realiza sobre dos ambientes diferentes: una empresa de telecomunicaciones satelitales y el grupo de investigación TLÖN, los cuales tienen las necesidades mencionadas en los numerales 3.4.1 y 3.4.2 respectivamente. En la tabla **4-1** se muestra el listado de interesados proveyendo su influencia o importancia y enmarcado en el proceso dentro de un ciclo de vida conjunto del servicio donde cada uno pertenece. Los interesados son integrados y vistos como parte de un solo escenario, a pesar que tienen presencia en campos muy distintos.

**Tabla 4-1**.: Listado de interesados

| **Interesado** | **Influencia** | **Proceso** |
|---|---|---|
| Director de infraestructura | Media | Estrategia |
| Ingeniero de infraestructura | Alta | Transición/Operación |
| Ingeniero de NOC | Baja | Operación |
| Investigador TLÖN | Media | Diseño |

A través de una entrevista informal con los interesados se indaga acerca de lo que esperan de un sistema de gestión que ofrezca solución a los desafíos que se pretenden resolver. A cada necesidad, deseo o expectativa se le añaden atributos descriptivos para soportar su posterior

análisis. Con esto se construyó la tabla **A-1** (ver anexo A) de requerimientos de interesados (Stackholders requirements specifications, StRS), en la cual se añade:

- una identificación única que sirve para reconocer el requisito en otras etapas del proyecto como en el análisis de riesgos
- una prioridad para determinar la importancia para el interesado y que sirve para tomar decisiones acerca de modificaciones o alternativas
- su dependencia ya que algunos requerimientos tienen relación de origen y si uno de ellos no es tenido en cuenta puede afectar la consecución de otro, a pesar que su prioridad sea baja
- grupos comunes de origen, que es lo mismo que los procesos del ciclo de vida del servicio donde tienen impacto
- un nivel de dificultad para tener un contexto de asequibilidad y asignación de costo
- su tipo para agrupar por clase de propiedad que representa

Con la lista de necesidades de los usuarios, se realiza un proceso de gestión de riesgos en un contexto del desarrollo del protocolo, su uso e implementación, que de acuerdo a [ISO et al., 2006] permite identificar amenazas, peligros o situaciones que generan riesgo, estimar su probabilidad de ocurrencia y consecuencias.

El procedimiento sigue ciertas actividades mencionadas en el estándar ISO/IEC/IEEE 16085 que inicia con la identificación de riesgos haciendo cuestionamientos hipotéticos acerca de lo que podría ocurrir si el requerimiento no es llevado a cabo o pensando en las limitaciones que existen para su ejecución. Luego se realiza el análisis cualitativo de los riesgos considerando su origen, probabilidad de ocurrencia y consecuencias, con lo que se llega a definir un nivel de riesgo apoyado en la matriz mostrada en la tabla **4-2**

Toda esa información es usada para generar alternativas en el desarrollo haciendo posible eliminar o aceptar el riesgo o minimizar su impacto. Así, por ejemplo, un riesgo detectado es que la MIB SNMP diseñada tenga problemas de compatibilidad con el gestor, lo cual, en términos generales, es posible y al ser un elemento funcionalmente importante para el protocolo SNMP, su impacto llega a ser severo en caso que se materialice. Bajo estas condiciones, el riesgo es catalogado con una prioridad alta, por ello es vital darle el tratamiento adecuado, que en este caso será validar la sintaxis de la MIB por medio de un software especializado. La misma evaluación se realizó con cada riesgo identificado y los resultados son mostrados en la tabla **A-2** del anexo A.

**Tabla 4-2**.: Prioridad del riesgo

| **Probabilidad** | **Impacto** | | |
|---|---|---|---|
| | **Severo** | **Moderado** | **Menor** |
| **Probable** | Alto | Alto | Medio |
| **Posible** | Alto | Medio | Bajo |
| **Improbable** | Medio | Bajo | Bajo |

## 4.2. Requerimientos del sistema

Se transforman los requerimientos de los interesados en elementos técnicos del protocolo conversor que entregue un servicio para satisfacer cierta necesidad. Como se menciona en el estándar ISO/IEC/IEEE 29148 [ISO et al., 2011] en este proceso se obtienen características, atributos, funciones y limitaciones del sistema. En la tabla **4-3** se muestra el resultado del análisis de los requerimientos de los interesados y como se convierte cada uno en una característica técnica del protocolo o su servicio que debe ser tenido en cuenta para el desarrollo. Por ejemplo, el requerimiento StRS01 que hace referencia al uso del protocolo con cualquier gestor SNMP tiene como característica del sistema que responde a solicitudes a través del puerto UDP 161 como origen y como destino al puerto UDP del paquete de solicitud recibido.

**Tabla 4-3**.: Especificación de requerimientos del sistema

| **StRS** | **Característica** |
|---|---|
| 01 | Responde a solicitudes a través del puerto UDP 161 como origen y como destino al puerto UDP del paquete de solicitud recibido |
| 01 | Cada dispositivo que se gestiona se encuentra en la MIB |
| 01 | La MIB es estandarizada y utilizable en múltiples ambientes |
| 01 | Las consultas de información se ejecutan de acuerdo a las solicitudes recibidas o la programación de notificaciones |
| 01, 02 | Permite modificación del nombre de la comunidad |
| 01, 03 | Soporta los tres tipos de comunidades: lectura, escritura y notificación |
| 01, 03 | Relaciona un OID SNMP con un comando de consulta en la sintaxis del dispositivo a gestionar |
| 01, 04 | Envia notificaciones al puerto UDP 162 como destino, usando como origen un puerto aleatorio |
| 01, 09 | Utiliza MIB-II RFC1213 |
| 04 | Sondea periódicamente el estado de uno o más objetos de cada dispositivo de red y cuando el umbral es alcanzado, informa al gestor para que este envíe una notificación SNMP notificando el evento |
| 04 | Algunos objetos tienen definido un umbral de operación que se usa para el envío de notificaciones |

**Tabla 4-3**.: Especificación de requerimientos del sistema

| StRS | Característica |
|---|---|
| 04 | Selección de notificaciones es configurable por el usuario |
| 05 | Registra eventos y errores en un archivo de logs |
| 05 | Importa módulo time para registrar fecha y hora de eventos o errores |
| 05 | Genera un nuevo archivo de log cada 100 KB |
| 05 | Las notificaciones automáticas realizan consultas automáticas cada 30 segundos |
| 05, 15 | Sobrescribe la nueva información de gestión recibida del agente |
| 05, 15 | No guarda histórico de información de gestión que ha capturado |
| 07 | Periodicidad con que envía solicitudes al dispositivo de red configurable con un valor en múltiplos de 10 segundos |
| 07 | Opera en múltiples puertos TTY de forma simultanea |
| 07 | Soporta comunicaciones seriales RS232 y RS485 |
| 07 | Recibe respuestas de consultas y extrae información de gestión |
| 07 | Utiliza un diccionario de Python en lugar de una base datos |
| 07, 12 | Selecciona la interfaz por la cual enviar las consultas |
| 07, 12 | Permite seleccionar la interfaz por la cual realizar las comunicaciones de consulta y respuesta |
| 07, 15 | Es autoconfigurable en dispositivos Ad Hoc |
| 11 | Se evita la retrasnmisión de paquetes cuando no recibe respuesta de una solicitud (tipo UDP) |
| 11 | Importa modulos pysnmp y pyserial |
| 11 | Se ejecuta como un script o servicio |
| 13 | Ejecuta sus procesos sobre GNU/Linux |

## 4.3. Requerimientos de software

Tomando las necesidades de usuario y los requerimientos del sistema, se construye la especificación de requerimientos de software (SRS) mostrada en la tabla **4-4**, en la cual las características son segmentadas de acuerdo a su uso o limitación y son aprovechadas para desarrollar módulos funcionales del protocolo conversor. Por ejemplo el tiempo de espera para recibir respuesta de solicitudes es una variable configurable que hace parte de las características de desempeño y es una condición que se tiene en cuenta en el algoritmo programable.

**Tabla 4-4**.: Especificación de requerimientos de software

| **Tipo** | **Característica** |
|---|---|
| Interfaz | El protocolo se ejecuta como un script o un demonio |
| Funciones | Recibir solicitudes SNMP, usar un OID para identificar parámetros de comunicación serial, enviar datos de consulta en sintaxis particular a través de una interfaz o puerto determinado, recibir las respuestas de las consultas, dar formato a información de gestión, responder a solicitudes SNMP, enviar notificaciones SNMP, guardar eventos en un archivo |
| Usabilidad | Importar-Exportar MIB, verificar logs de eventos, modificar comunidad SNMP |
| Desempeño | Tiempo de espera para recibir respuesta de solicitudes, cantidad de servidores de gestión SNMP que pueden hacer consultas, cantidad de nodos Ad Hoc o dispositivos con conexión serial que puede soportar en simultaneo |
| Base de datos | Se usa la estructura de datos nativa de Python a través de diccionarios, no se requiere implementar algún tipo de base de datos relacional o no relacional dado que la cantidad de datos que se deben consultar es pequeña |
| Restricciones | Seguridad básica basada en comunidad, movilidad entre nodos no diseñada, sin interfaz de usuario GUI, nuevas MIB se crean de forma externa |

# 5. Diseño y construcción del protocolo

## 5.1. Diseño del protocolo

Como se mencionó en el numeral 3.5 los desafíos de gestión de dispositivos heredados o en redes Ad Hoc pueden ser resumidos en un problema de incompatibilidad de procesos de comunicación que puede ser solucionado mediante la conversión de protocolos. Lam [Lam, 1988] sugiere un modelo de conversores de estados finitos mediante la construcción de una imagen de protocolo común agregando funcionalidades mediante una máquina de estados finita. Dicho concepto es usado para diseñar el servicio que facilita la interoperabilidad entre los dos protocolos SNMP y serial, aún cuando la solución no implica una conversión directa de los mensajes de cada uno de ellos como se explica más adelante.

Considerando la figura **5-1**, donde el protocolo SNMP es denominado $J$ y el protocolo serial como $S$, los cuales manejan sintaxis y semántica diferentes, para que los mensajes $M$ y $N$ puedan ser entendidos por ambas partes, se debe usar un elemento intermedio $I$ que entienda ambas partes para que pueda existir interoperabilidad. Así, la parte de la red denominada $J_I$ puede ser vista como un proceso que interactua con $J$, cuyos estados se encuentran definidos por una tupla ($s_1$, $s_2$, $m_1$, $m_2$), donde $s_1$ y $s_2$ son estados de $I$ y $S$ respectivamente, mientras que $m_1$ y $m_2$ representan una secuencia de mensajes en los tramos $J - I$ e $I - S$. La misma lógica se maneja para $S_I$.

Visto a nivel de mensajes de protocolo, el resumen de mapeo de primitivas es mostrado en la figura **5-2** la cual toma como base la explicación del modelo de conversión de protocolos de LAM. En el modelo se desglosa cada protocolo a nivel de diagrama de flujo para entender la forma en que los mensajes entran y salen de cada nodo conforme a los eventos internos que aparecen en cada uno de ellos. Una vez se tienen estos detalles, se construyen las máquinas de estado para relacionarlos.

Para construir la imagen del protocolo mediante las máquinas de estado finitas de los protocolos SNMP y serial, cuyo resultado constituye la descripción formal del protocolo mostrado en la figura **5-7**, se analiza el flujo de las primitivas de servicio que tienen relevancia en los procesos de comunicación y relacionando los estados donde son ejecutadas las funciones de consulta de información de gestión usando:

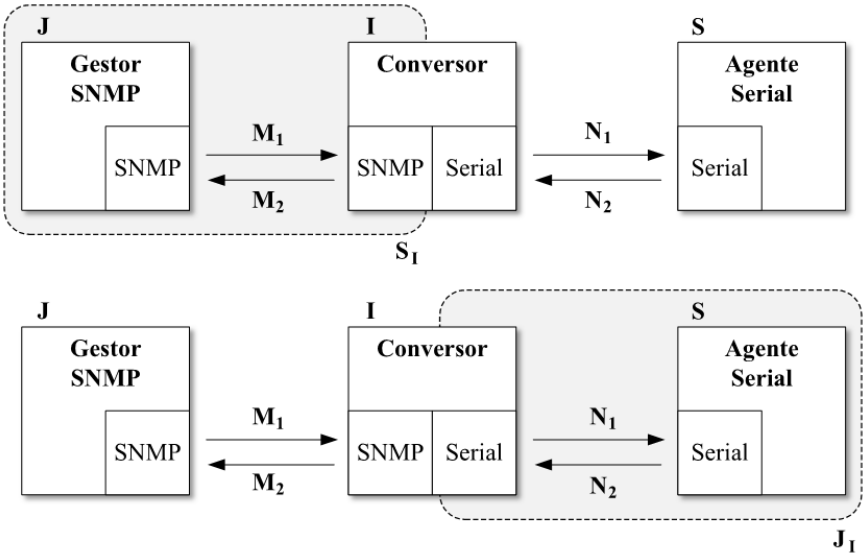


**Figura 5-1**.: Modelo de conversión de protocolos
Fuente propia

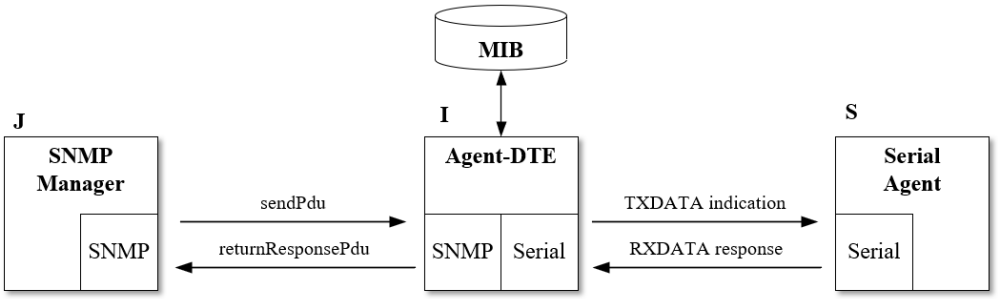


**Figura 5-2**.: Resumen de mapeo de mensajes
Fuente propia

- algunas primitivas de servicio que definen las comunicaciones del despachador en el motor SNMP y que son especificadas en el RFC3411 [Harrington et al., 2002] para un agente SNMP. Existen otras primitivas para comunicaciones internas entre otros subsistemas, las cuales son utilizadas por cualquier entidad SNMP pero que no ayudan a explicar este diseño y son consideradas como eventos internos que existen para que la parte SNMP funcione.

- el intercambio de mensajes de comunicación usados en el control de flujo acorde a la especificación RS-232 [Jiménez et al., 2014], aplicando la configuración más habitual con mensajes TXD y RXD [1] y combinada con primitivas de servicios no orientados a

[1] Otras señales de control, como RTS (resquest to send), CTS (clear to send), DTR (data terminal ready) o DSR (data set ready), no son consideradas debido a que son aplicadas sobre comunicaciones con módem

conexión [Hura and Singhal, 2001].

## 5.2. Especificación del Servicio

El protocolo de comunicaciones diseñado ofrece un servicio de traslación de información de gestión asíncrona y no orientada a conexión entre dos protocolos que tienen sintáxis y semántica diferente. En la figura **5-3** se pone en vista el modelo de comunicación de servicio, el cual está definido en el modelo OSI [ITU-T, 1993] [ISO and IEC, 1995], en donde se distinguen los servicios de usuario *LOCAL-FOREIGN MANAGER*, *AGENT-DTE* y *DCE*, los cuales solicitan los servicios que presta el proveedor de servicio *JIS Process* mediante las primitivas que utiliza cada protocolo para realizar las invocaciones a través de puntos de acceso del servicio (Service Access Point, SAP).

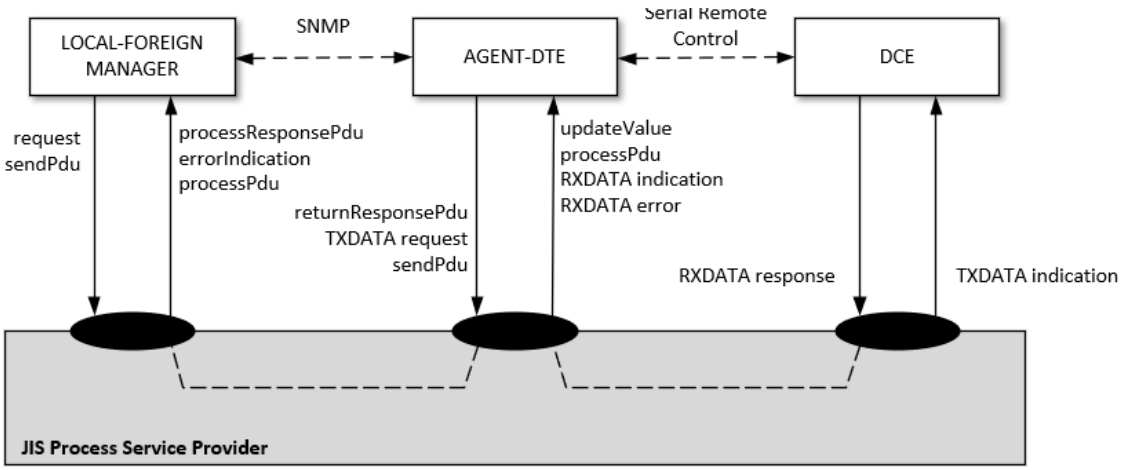


**Figura 5-3**.: Comunicación del servicio y protocolo
Fuente propia

### 5.2.1. Primitivas de servicio y secuencias de tiempo

En el modelo se detallan los procesos para los dos protocolos SNMP y serial, los cuales son tratados de manera independiente aún cuando están relacionados. El primero puede iniciar a través de solicitudes locales o externas con las primitivas *request* o *sendPdu* correspondientemente, originadas por el LOCAL-FOREIGN MANAGER, la cual contiene la información necesaria para solicitar información de gestión. Si el inicio de la transacción es local[2], el receptor recibe una indicación de dicha solicitud mediante la primitiva *updateValue* con la cual

y esa clase de conexiones no hacen parte del enfoque de esta investigación

[2]Existen algunos procesos internos que solicitan recursos propios como la recolección de datos por adelantado. Ver sección 5.5.2

se invocan los servicios de comunicación serial a través de la primitiva *TXDATA request*. Si la solicitud proviene de un servicio de usuario externo como un gestor SNMP, el servicio es invocado por medio de la primitiva *sendPdu*. En cualquiera de los dos casos (local o externo), el proceso sigue con el envío de al primitivas de servicio *processResponsepDU* para leer o escribir en la tabla SNMP y con *returnResponsePdu* para dar la respuesta correspondiente [Harrington et al., 2002]. En la figura **5-4** se muestra el diagrama de secuencia de tiempos del proceso SNMP. Se observan las primitivas de servicio tanto si se recibe una respuesta como si es superado el tiempo de espera, en cuyo caso la entidad SNMP genera la primitiva *errorIndication*

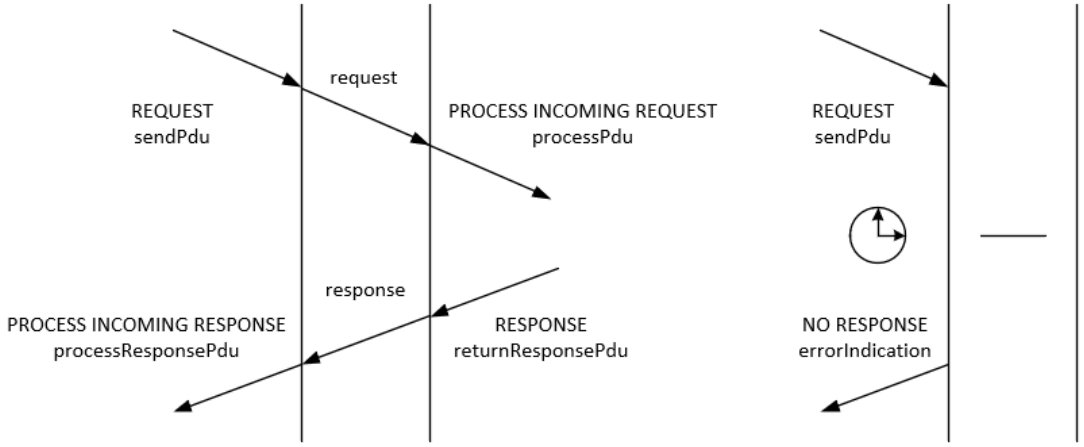


**Figura 5-4**.: Diagrama de secuencia de tiempo del servicio lado SNMP

Por su parte, del lado serial, se uso el método no orientado a conexión donde el AGENT-DTE realiza consultas periódicas al DCE mediante la primitiva *TXDATA request*, la cual es indicada al receptor mediante la primitiva *TXDATA indication*, similar a los mecanismos publicados en [ISO and IEC, 1996]. Una vez el servicio de usuario DCE recibe la indicación, se activan los procesos para obtener los datos de gestión, los cuales son puestos en el mensaje y retornados mediante las primitivas *RXDATA indication* y *RXDATA response*. El proceso es representado en el diagrama de secuencia de tiempos de la figura **5-5** donde se muestran las primitivas de servicio que siguen una respuesta exitosa o un error (*RXDATA error*) por superar el tiempo de espera

Por último, cada vez que se realiza una modificación a las tablas SNMP, bien sea por actualización de los valores de los objectos o por la creación o destrucción de filas de forma dinámica (por el ingreso o salida de un dispositivo), el sistema genera una notificación SNMP a través de la primitiva de servicio *sendPdu* que puede ser recibida por un gestor de traps externo, al cual se le indica con la primitiva *processPdu*. Este es un proceso netamente SNMP pero que depende de la actualización realizada durante los procesos de comunicación serial.

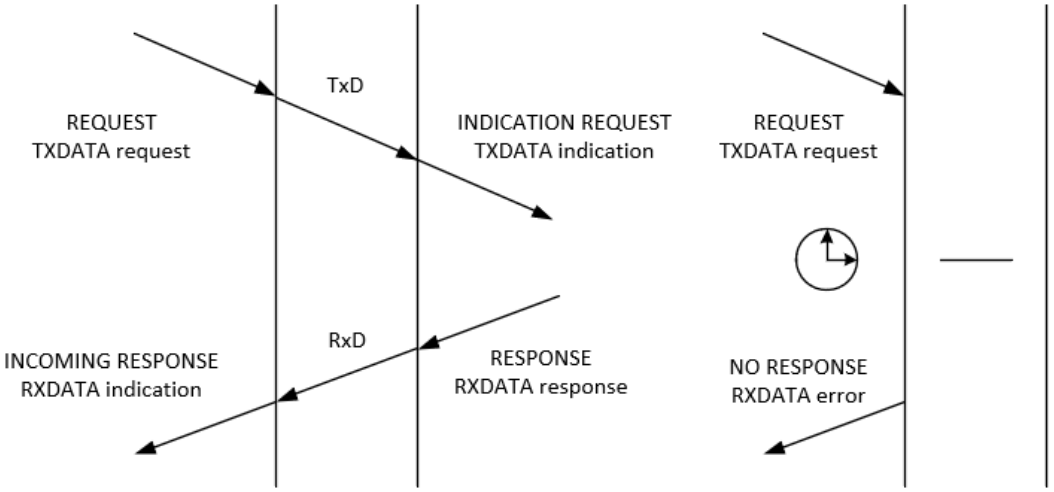


**Figura 5-5**.: Diagrama de secuencia de tiempo del servicio lado Serial

El diagrama de secuencia de tiempos de la figura **5-6**, muestra el funcionamiento de este servicio en el que no se requiere ninguna respuesta de confirmación de recepción, es decir, que es en una sola vía.

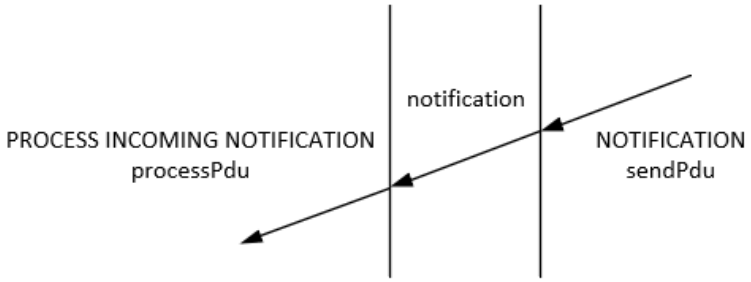


**Figura 5-6**.: Diagrama de secuencia de tiempo de notificaciones SNMP

### 5.2.2. Descripción formal

Visto en el contexto global del *servicio JIS*, los dos protocolos SNMP y serial prestan sus servicios de forma asimétrica e independiente, sin embargo se construyo un solo diagrama de estados del proveedor de servicios usando la imagen del protocolo común descrito en la sección 5.1 y mostrada en la figura **5-7**, el cual relaciona los estados de los dos protocolos SNMP y serial, abordando solo las primitivas necesarias para tal fin y que fueron explicados en la sección anterior. Apartes del código que especifica los servicios SNMP y serial, son mostrados en la sección 5.5.

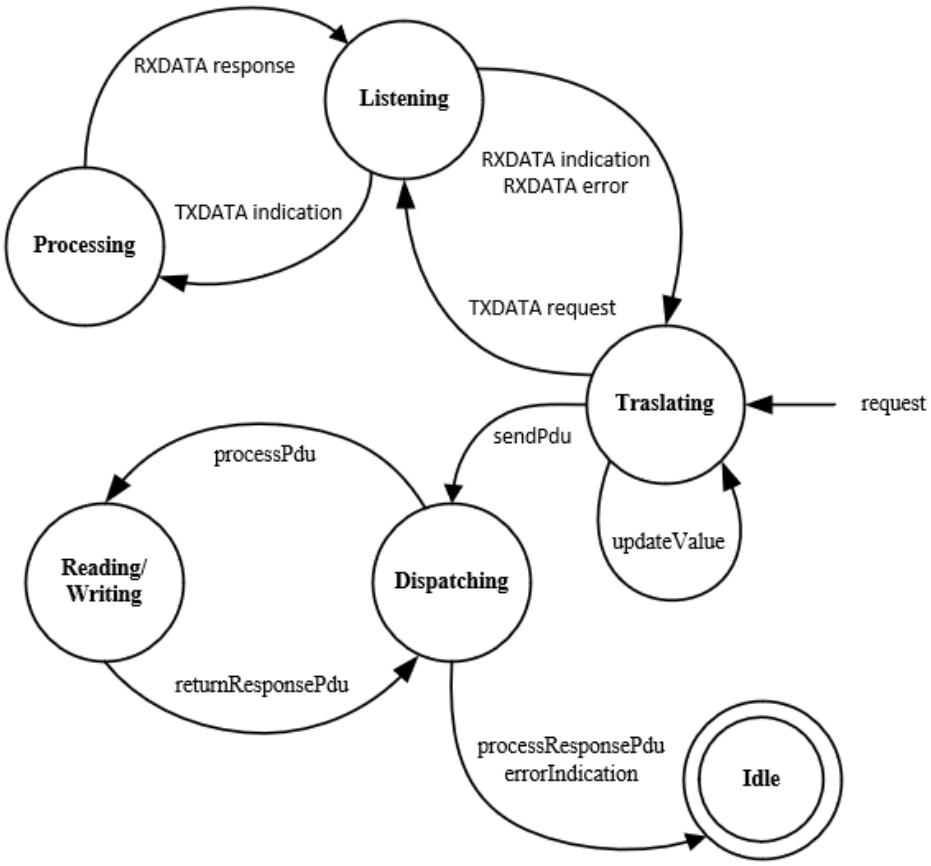


**Figura 5-7**.: Diagrama de estado del proveedor de servicios JIS
Fuente propia

## 5.3. Especificación del Protocolo

El protocolo provee un método de gestión de variables de operación de dispositivos no-SNMP mediante el intercambio de PDU de forma asimétrica entre entidades SNMP y SERIAL, las cuales tienen varios procedimientos y mecanismos que se definen en las funciones del protocolo. En la figura **5-8** se muestra el intercambio de PDU entre entidades lo cual detalla la especificación del protocolo.

### 5.3.1. Formato de PDU

Las unidades de datos que son intercambiadas entre las entidades de comunicación de ambos protocolos son implementados de manera separada.

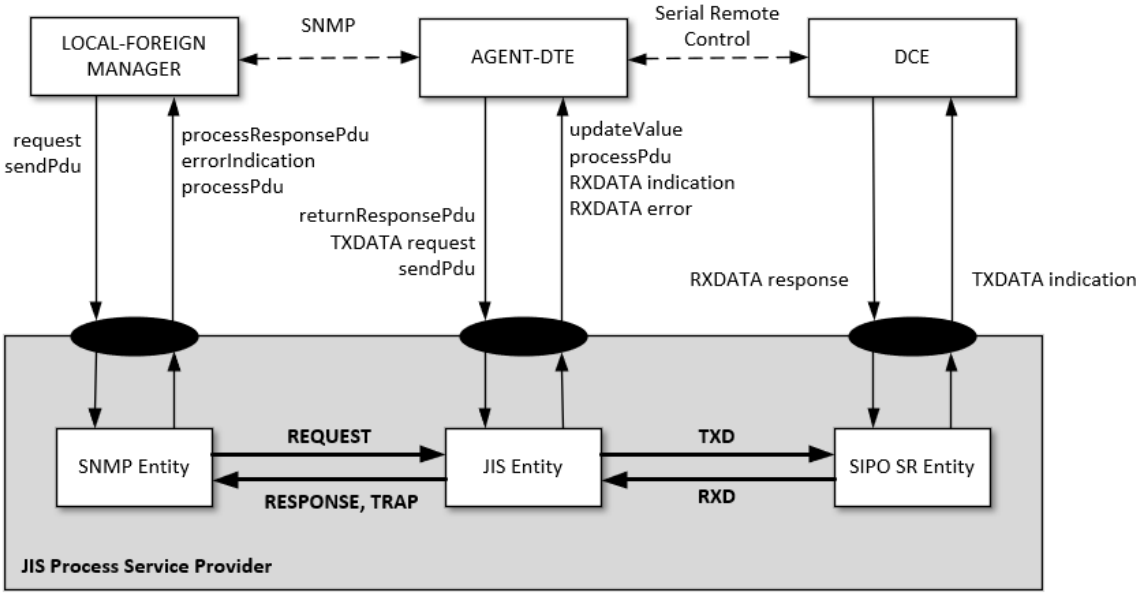


**Figura 5-8**.: Comunicación del servicio y protocolo
Fuente propia

## SNMP

El RFC1157 [Case et al., 1990] indica que es mandatario que todas las implementaciones de SNMP soporten cinco PDU: GetRequest, GetNextRequest, GetResponse, SetRequest y Trap (o Notificación). La definición de cada PDU SNMP depende de la versión implementada, en nuestro caso la versión 2 se encuentra detallada en el RFC3416. En la figura **5-8** se muestran dos entidades que hacen transacciones con PDU tipo SNMP, SNMP Entity y JIS Entity. La primera posee una arquitectura con funciones de generador de comandos SNMP y receptor de notificaciones, mientras que la segunda hace operaciones de respondedor de comandos y generador de notificaciones. Ambos responden a las primitivas de servicio para generar el intercambio de mensajes que es netamente SNMP cuyo orden y condicional es explicada en [Case et al., 2002] donde puede ser consultada. A continuación la definición de las PDU usando notación ASN.1:

```
GetRequest-PDU ::= [0] IMPLICIT PDU
GetNextRequest-PDU ::= [1] IMPLICIT PDU
Response-PDU ::= [2] IMPLICIT PDU
SetRequest-PDU ::= [3] IMPLICIT PDU
GetBulkRequest-PDU ::= [5] IMPLICIT BulkPDU
SNMPv2-Trap-PDU ::= [7] IMPLICIT PDU

PDU ::= SEQUENCE {
        request-id INTEGER (-214783648..214783647),
```

```
        error-status                -- sometimes ignored
            INTEGER {
                noError(0),
                tooBig(1),
                noSuchName(2),      -- for proxy compatibility
                badValue(3),        -- for proxy compatibility
                readOnly(4),        -- for proxy compatibility
                genErr(5),
                noAccess(6),
                wrongType(7),
                wrongLength(8),
                wrongEncoding(9),
                wrongValue(10),
                noCreation(11),
                inconsistentValue(12),
                resourceUnavailable(13),
                commitFailed(14),
                undoFailed(15),
                authorizationError(16),
                notWritable(17),
                inconsistentName(18)
            },

        error-index                 -- sometimes ignored
            INTEGER (0..max-bindings),
        variable-bindings           -- values are sometimes ignored
            VarBindList
    }
BulkPDU ::=                         -- must be identical in
    SEQUENCE {                      -- structure to PDU
        request-id      INTEGER (-214783648..214783647),
        non-repeaters   INTEGER (0..max-bindings),
        max-repetitions INTEGER (0..max-bindings),
        variable-bindings           -- values are ignored
            VarBindList
    }
-- variable binding
VarBind ::= SEQUENCE {
        name ObjectName,
         CHOICE {
            value           ObjectSyntax,
            unSpecified     NULL,   -- in retrieval requests
                                    -- exceptions in responses
            noSuchObject    [0] IMPLICIT NULL,
            noSuchInstance  [1] IMPLICIT NULL,
            endOfMibView    [2] IMPLICIT NULL
        }
    }
```

```
-- variable-binding list
VarBindList ::= SEQUENCE (SIZE (0..max-bindings)) OF VarBind
```

### UDP-SERIAL

Cuando hay una invocación de servicio a través de la primitiva TXDATA request, a la entidad JIS entity se recibe una PDU que puede ser de diferente tipo dependiendo de la conexión que el dispositivo destino maneje, TxdUDP para UDP a través de WiFi O TxdRs232 para RS-232. La PDU TxdUDP posee la información de dirección IP y puerto UDP para crear un socket UDP, a través del cual se envía el mensaje de datos. Entre tanto, la PDU TxdRs232 está constituida por los parámetros de conexión física del puerto serial del dispositivo destino, algunos son obligatorios y otros opcionales, con los que se realiza el establecimiento de la comunicación para enviar el mensaje de datos que ha de ser previamente codificado de acuerdo al formato que entiende el destino. Las PDU mencionadas son entregadas a la entidad par SIPO Entity (Serial-Input Paralell-Output [Jiménez et al., 2014]) que envía al DCE la primitiva TXDATA indication señalando que se ha recibido una solicitud de información de gestión. Posteriormente, SIPO Entity recibe una PDU RxdUDP o RxdRs232 mediante la primitiva RXDATA response, cuya estructura es la misma que la PDU origen TxdUDP o TxdRs232, según sea el caso. La PDU es retornada a la entidad par JIS entity la cual notifica al servicio de usuario AGENT-DTE mediante la primitiva RXDATA indication. Si pasado el tiempo de espera no se recibe una PDU de retorno por parte de SIPO entity, JIS entity enviara una primitiva RXDATA error. A continuación la definición de las PDU usando notación ASN.1:

```
PDUs ::= CHOICE {
      txd-udp       TxdUDP-PDU,
      rxd-udp       RxdUDP-PDU,
      txd-rs232     TxdRs232-PDU,
      rxd-rs232     RxdRs232-PDU}

-- PDUs

TxdUDP-PDU ::= [0] IMPLICIT UDP
RxdUDP-PDU ::= [1] IMPLICIT UDP
TxdRs232-PDU ::= [2] IMPLICIT RS232
RxdRs232-PDU ::= [3] IMPLICIT RS232

UDP ::= SEQUENCE {
         message OCTET STRING,
         address IpAddress,
         port    INTEGER (0..65535)
     }
```

```
RS232 ::= SEQUENCE {
        address  OCTET STRING,
        baudrate INTEGER (0..115200),
        timeout  INTEGER (0..300),
        message  OCTET STRING,
        encode   OCTET STRING,  --set encode/decode scheme,
                                --one of {utf-8, ascii} (def utf-8)
       -- optional parameters
        parity   OCTET STRING,  --set parity, one of {N E O S M} (def N)
        rtscts   OCTET STRING,  --enable RTS/CTS flow control (def off)
        xonxoff  OCTET STRING,  --enable software flow control (def off)
        rts      INTEGER (0,1), --set initial RTS line state (def 0)
        dtr      INTEGER (0,1)  --set initial DTR line state (def 0)
    }
```

Los datos contenidos en el mensaje de las PDU RxdUDP o RxdRs232 son decodificados por el AGENT-DTE y enviados dentro de la PDU SNMP SetRequest para almacenar la información en la tabla SNMP, que posteriormente será consultada. Al mismo tiempo se activa la primitiva de servicio sendPdu indicando el envío de una notificación o trap SNMP ante la actualización de la tabla mencionada.

### 5.3.2. Funciones del protocolo

#### Timers

En diferentes módulos del desarrollo se utilizan tiempos de espera para que la respuesta a una solicitud pueda ser procesada, enviada o recibida. En caso que esto no ocurra la entidad origen genera una primitiva de indicación de error como se muestra en los diagramas de secuencias de tiempo de las figuras **5-4** y **5-5**. Esto también evita que se reciban PDU duplicadas por retraso en el procesamiento o por limitaciones del medio, ya que el tiempo de espera de una respuesta es mucho menor que el tiempo entre solicitudes.

#### Codificación

La codificación de la PDU SNMP se realiza a través del lenguaje ASN.1 siguiendo el estándar del protocolo y usando la librería de python **pyasn**[3]. Esta ya viene implementada dentro de la librería *pysnmp*, por lo cual no se hace necesario una implementación adicional. Dentro de la programación se observa la importación de código como *hlapi*[4], cuyos recursos son los que usan directamente la librería de ASN.1.

```
from pysnmp.hlapi import *
```

[3]http://pyasn1.sourceforge.net/
[4]https://github.com/etingof/pysnmp/tree/master/pysnmp/hlapi

La PDU de las comunicaciones seriales no tiene una sintaxis estándar debido a que cada dispositivo tiene o entiende las conexiones de manera diferente, por lo que el protocolo debe implementar un método por cada una. En general, la mayoría de ellas son basadas en texto usando ASCII o UTF-8, lo cual simplifica el proceso de codificación/decodificación, el cual utiliza los métodos nativos del lenguaje de programación. A continuación se muestra la función base que envía y recibe una cadena de caracteres que representa un mensaje en comunicación serial, que es convertido en Bytes para luego ser transmitido en formato UTF-8 y ejecuta el proceso contrario cuando recibe la respuesta.

```
def loading (self, address, message):
        txd = serial.Serial(address, baudrate=9600, timeout=1)
        txd.write(message.encode())
        rxd = txd.readline().decode()
        txd.close()
        return rxd
```

### Autoconfiguración

Las WSNs sufren cambios de topología dinámica debido al ingreso o salida de un nodo de la red, aunque comparado con las MANETs estos cambios se hacen en menor medida [Benhaddou and Al-Fuqaha, 2015]. Para realizar conexiones dinámicas a través de SNMP, se omite el uso de OID fijos para cada objeto, ya que no se conoce la cantidad de host que van a operar dentro de la red, a su vez que es importante manejar reuso de recursos físicos y lógicos. Por ello los objetos en la MIB (ver sección 5.4) se representan mediante tablas SNMP que tienen filas conceptuales que permiten la creación o eliminación de instancias de objetos [McCloghrie et al., 1999a] a través de un objeto columnar denominado *RowStatus*, cuyo valor representa el estado de su fila en la tabla SNMP. Existen seis posibles valores que definen dicho estado: active (1), notInService (2), notReady (3), createAndGo (4), createAndWait (5) y destroy (6) [McCloghrie et al., 1999b]. El valor de *RowStatus* puede ser cambiado mediante comandos SNMP tipo *setRequest*, bien sea con un valor de cadena de caracteres o un entero.

### Secuencia y Control de Flujo

Debido a que la capa de transporte utilizada por el protocolo es UDP, no hay soporte de confiabilidad, control de flujo o recuperación de errores desde las capas inferiores, por lo que esas funciones deben ser ejecutadas por el protocolo como parte de las acciones para prestar el servicio.

- Los paquetes de gestión son muy pequeños para pensar en el uso de funciones de re-ensamble o aplicación de números de secuencia, lo que se hace es identificar que los paquetes de retorno correspondan a las solicitudes SNMP a través del parámetro *request-id*. En el caso de las comunicaciones seriales, agregar un campo de identificación de paquete tiene limitaciones sobre todo para controlar dispositivos heredados, sobre los cuales no hay manera de modificar su sistema operativo.

- El control de flujo es realizado por aproximación mediante el uso de los timers descritos con anterioridad, ya que en el intercambio de mensajes no existen paquetes de control o mantenimiento de la conexión que permitan pausar el flujo de datos cuando haya congestión en la red.

## 5.4. Construcción de la MIB

La premisa en el diseño y construcción de la MIB fue realizar un desarrollo estándar al protocolo SNMP y siguiendo el formato adecuado. Por ello se solicito la asignación de un número único dentro del árbol SMI. Con esto se realizo el diseño y codificación de dos MIB a través del software MIB Smithy [5], la cual facilitó validar y compilar su sintaxis mientras se hacia su desarrollo.

### 5.4.1. Número Privado de Empresa

En una MIB los identificadores de objeto OID se encuentran implementados en un árbol jerárquico SMI. Su numeración se encuentra administrada por varias organizaciones como ETSI (European Telecommunication Standards Institute), ITU-T (International Telecommunications Union), IANA (Internet Assigned Numbers Authority). Para el desarrollo de la MIB, podrían usarse números de asignación experimentales (1.3.6.1.3), sin embargo para dejar el desarrollo en producción en necesario mover los OID a otras rama del árbol como al mng (1.3.6.1.2), algo que es poco práctico [Heard, 2005]. Por ello y además de facilitar desarrollos futuros, se procuró ante IANA la designación de un número privado de empresa (Private Enterprise Number, PEN) con la intención de hacer visible el nombre de la Universidad Nacional de Colombia frente a organismos internacionales. La solicitud fue registrada con ticket REQUEST-77527 y aprobada con la asignación del PEN 49843[6]. De esta manera, el árbol SMI de OID para todos los desarrollos que se realicen por la Universidad, en los cuales se deba usar un PEN es 1.3.6.1.4.1.49843 o iso.identified-organization.dod.internet.private.enterprise.49843.

[5] Se recibió una licencia temporal gratuita por parte de la empresa desarrolladora Muonics Inc para uso únicamente académico. Más información en http://www.muonics.com/Products/MIBSmithy/

[6] Este registro puede ser consultado en http://www.iana.org/assignments/enterprise-numbers

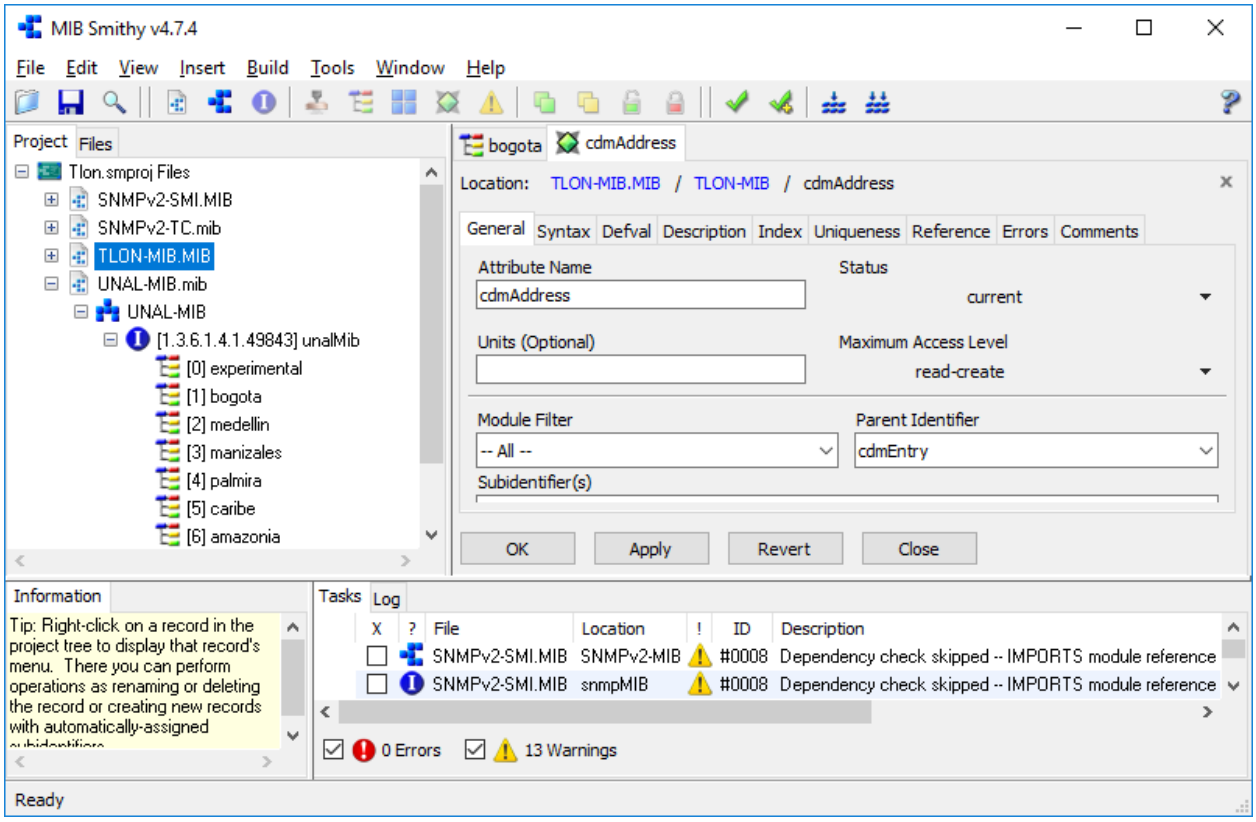


**Figura 5-9**.: Construcción y validación de sintaxis de MIB
Fuente propia

### 5.4.2. MIB UNAL

Se estructuró el árbol SMI asignando una rama a cada sede de la Universidad. Futuros proyectos deberán ubicarse dentro de su rama correspondiente y siguiendo el consecutivo de la numeración establecida. De esta forma, por ejemplo, el proyecto TLÖN de la sede de Bogotá, tiene el OID 1.3.6.1.4.1.49843.1.1

La definición de estos objetos en la MIB se encuentra en formato ASN.1. tal como se muestra a continuación. La MIB UNAL completa se encuentra en el Anexo B

```
experimental OBJECT IDENTIFIER ::= { unalMib 0 }
bogota OBJECT IDENTIFIER ::= { unalMib 1 }
medellin OBJECT IDENTIFIER ::= { unalMib 2 }
manizales OBJECT IDENTIFIER ::= { unalMib 3 }
palmira OBJECT IDENTIFIER ::= { unalMib 4 }
caribe OBJECT IDENTIFIER ::= { unalMib 5 }
amazonia OBJECT IDENTIFIER ::= { unalMib 6 }
tumaco OBJECT IDENTIFIER ::= { unalMib 7 }
orinoquia OBJECT IDENTIFIER ::= { unalMib 8 }
```

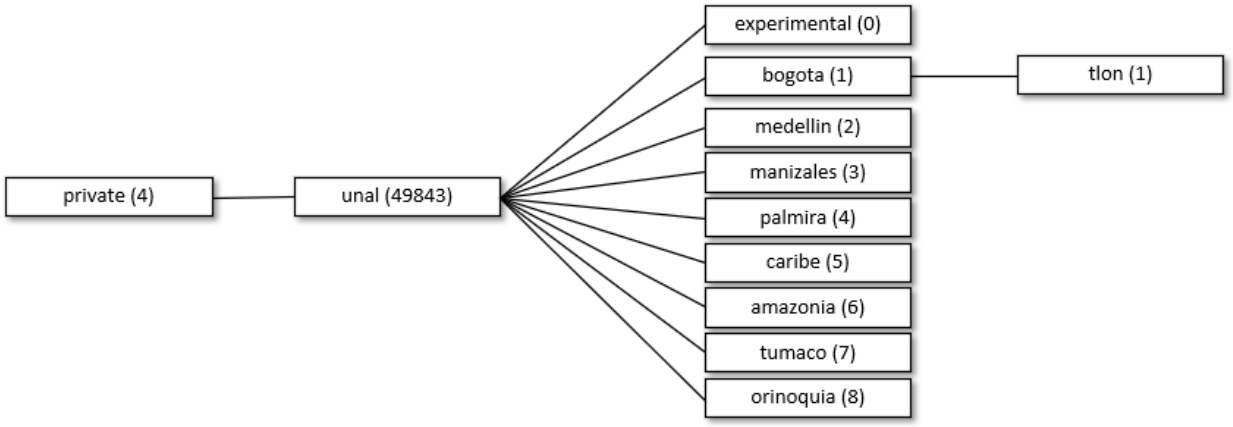


**Figura 5-10**.: Estructura del árbol SMI UNAL-MIB
(Fuente propia)

### 5.4.3. MIB TLÖN

La mib fue desarrollada bajo la estructura SMI versión 2 (SMIv2), ya que es el estanándar más reciente, la cual incluye los elementos mostrados en la figura **5-11**. A continuación se explican los módulos implementados y se muestran apartes de la MIB, la cual puede ser consultada en su totalidad en el Anexo C.

Se importan algunos objetos definidos en otras MIB, necesarios para la creación de los propios objetos en la MIB TLON, entre ellos se destacan *RowStatus* para manejar el estado de las filas en una tabla, *TEXTUAL-CONVENTION* para definir tipos de variables inexistentes en el formato ASN.1 y *bogota* para identificar este proyecto dentro de la rama correspondiente.

```
IMPORTS
    NOTIFICATION-GROUP, OBJECT-GROUP
        FROM SNMPv2-CONF
    Integer32, IpAddress, MODULE-IDENTITY, NOTIFICATION-TYPE,
    OBJECT-IDENTITY, OBJECT-TYPE
        FROM SNMPv2-SMI
    RowStatus, TEXTUAL-CONVENTION
        FROM SNMPv2-TC
    bogota
        FROM UNAL-MIB
```

Los segmentos de definiciones de objetos y en objectos de conformidad especifican las representaciones de elementos, sus atributos y agrupaciones

```
tlonObjects OBJECT IDENTIFIER ::= { tlonMib 1 }
...
```

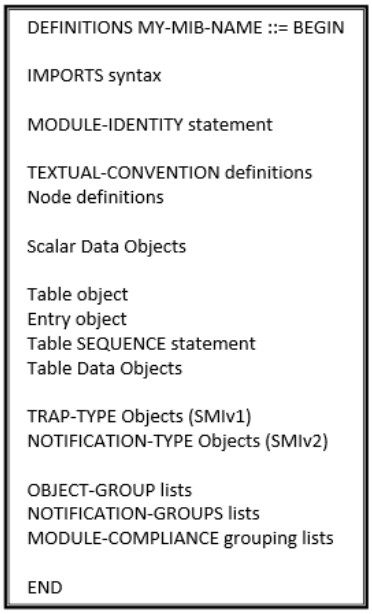


**Figura 5-11**.: Estructura base de la MIB
Fuente [Walsh, 2008]

```
tlonConformance OBJECT IDENTIFIER ::= { tlonMib 2 }
```

### Definiciones de Objectos

En este segmento se encuentran definidos los objetos que representan cada dispositivo y sus variables de operación. Esto se hace mediante una tabla (ver figura **5-12**) donde cada fila es un equipo y las columnas sus partes escalares. Se cuenta con tres tipos de identidades de objeto:

- notificaciones: tiene un único objecto llamado *alterTable* que se utiliza para el envío de notificaciones a un servidor receptor de traps, las cuales se activan cuando una tabla ha sido alterada bien sea por la creación o destrucción de una fila o por la actualización de algún valor dentro de la MIB

```
tlonNotification OBJECT IDENTIFIER ::= { tlonObjects 0 }

alterTable NOTIFICATION-TYPE
    OBJECTS         { bmpPres, bmpTemp }
    STATUS          current
    DESCRIPTION     "An alterTable trap signifies that the SNMP table
                    has been altered and it must be notified to a trap
                    server."
    ::= { tlonNotification 1 }
```

- legacy: dentro de esta definición se crean los objetos de dispositivos heredados, por ejemplo en la tabla *cdmTable* cada fila representa un modem Comtech CDM600[7] y cada columna es imagen de variables como frecuencia *cdmTxFrec*, modulación *cdmTxMod*, potencia *cdmTxPower*, entre otros. La definición de estos objetos sigue el mismo formato que se muestra como ejemplo en el siguiente ítem.

- ad hoc: esta definición es usada para introducir las representaciones de sensores en operación ad hoc como objetos en tablas, por ejemplo el *bmpTable* que se refiere a dispositivos BMP180 [8] cuyas columnas representan las mediciones de temperatura *bmpTemp* y presión *bmpPres* o el *hcsrTable* que representa un sensor de proximidad HC-SR04 con objetos escalares donde se registra la medición de distancia *hcsrDistance*.

```
tlonAdhoc OBJECT IDENTIFIER ::= { tlonObjects 2 }

bmpTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF BmpEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "A list of BMP180 sensors"
    ::= { tlonAdhoc 1 }

bmpEntry OBJECT-TYPE
    SYNTAX          BmpEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "The BMP180 is a basic sensor that is designed
                    specifically for measuring barometric pressure,
                    it also does temperature measurement on the side
                    to help.
    INDEX
        { bmpId }
    ::= { bmpTable 1 }

BmpEntry ::= SEQUENCE
{
    bmpId           Integer32,
    bmpAddress      IpAddress,
    bmpTemp         Integer32,
    bmpPres         Integer32,
    bmpPres2d       Float2d,
    bmpPres32       Float32,
    bmpTableStatus  RowStatus
```

[7] http://www.comtechefdata.com/files/manuals/mn-modems-pdf/mn-cdm600-600L.pdf
[8] https://learn.adafruit.com/bmp085?view=all

```
}

bmpId OBJECT-TYPE
    SYNTAX          Integer32 (0..32000)
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "A number to identify each sensor independently"
    ::= { bmpEntry 1 }

bmpAddress OBJECT-TYPE
    SYNTAX          IpAddress
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "IP Address of the device"
    ::= { bmpEntry 2 }

bmpTemp OBJECT-TYPE
    SYNTAX          Integer32
    UNITS           "deg C"
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "temperature sensing, -40 to + 85C operational
                    range, +-2 C temperature accuracy. The manager
                    should divide the data by 100, to obtain the
                    real value in decimal."
    ::= { bmpEntry 3 }

(...)

bmpPres32 OBJECT-TYPE
    SYNTAX          Float32
    UNITS           "hPa"
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "The same bmpPres object but with another kind
                    of type to demonstrate the use of
                    TEXTUAL-CONVENTION base on Octect String"
    ::= { bmpEntry 6 }

bmpTableStatus OBJECT-TYPE
    SYNTAX          RowStatus
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION
        "To create or drop an entire row in one shot (RFC 2579)"
    ::= { bmpEntry 7 }
```

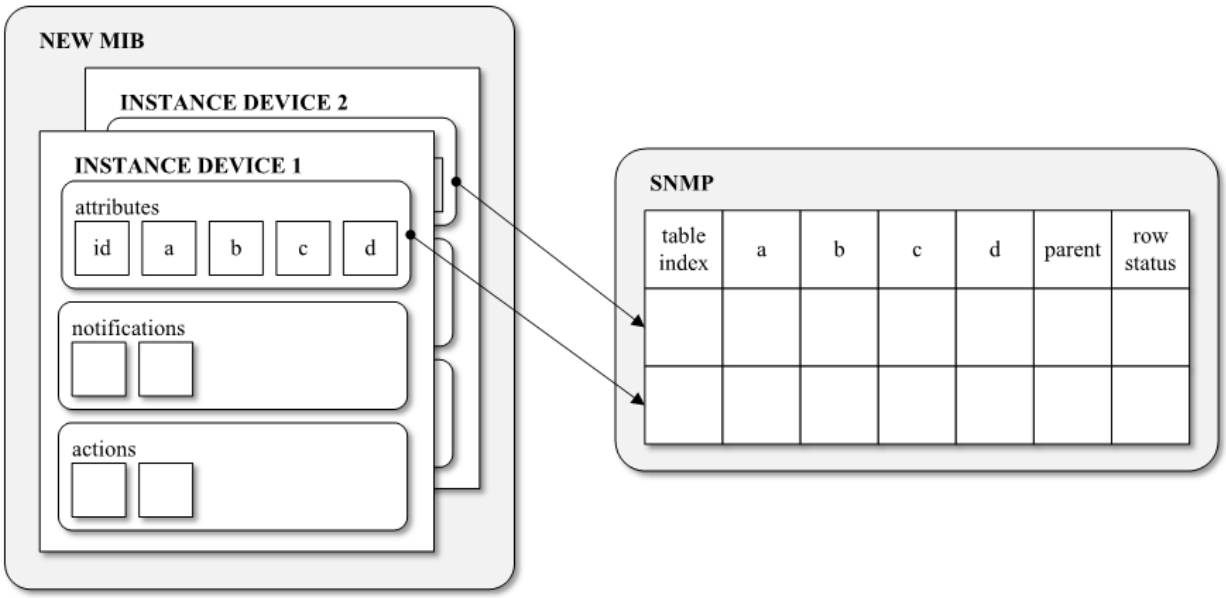


**Figura 5-12**.: Enfoque de traslación de atributos a SNMP
Fuente [Koerner, 1997]

### Objectos de Conformidad

Es una agrupación de objectos de datos y notificaciones. A cada elemento le es asignado un OID pero estos únicamente sirven para que el lector entienda un poco mejor la estructura de la MIB, más no se utiliza como objectos en procesos de gestión *get*, *set* o *request*.

```
tlonConformance OBJECT IDENTIFIER ::= { tlonMib 2 }

tlonNotifications OBJECT IDENTIFIER ::= { tlonConformance 0 }

notificationsGroup NOTIFICATION-GROUP
    NOTIFICATIONS
        { alterTable }
    STATUS          current
    DESCRIPTION     "Some notifications implemented over the SNMP agent
                    supporting ommand responder applications."
    ::= { tlonNotifications 1 }

tlonCompliances OBJECT IDENTIFIER ::= { tlonConformance 1 }

tlonGroups OBJECT IDENTIFIER ::= { tlonConformance 2 }

(...)

sensorObjectGroup OBJECT-GROUP
```

```
    OBJECTS
        { bmpPres, bmpPres2d, bmpPres32, bmpTemp, hcsrDistance }
    STATUS          current
    DESCRIPTION     "This group contains the principal objects of ad hoc
                    devices"
    ::= { tlonGroups 3 }

(...)

END
```

## 5.5. Arquitectura

La arquitectura del protocolo desarrollado se muestra en la figura **5-13**, cuyos módulos se programaron con lenguaje Python utilizando librerías ya existentes como pysnmp [Etingof, 2017] y pyserial [Liechti, 2015] que facilitan los desarrollos en servicios SNMP y comunicaciones seriales respectivamente.

### 5.5.1. Módulos

A continuación se explica el funcionamiento de cada módulo y se muestran apartes el código fuente, el cual puede ser consultado en su totalidad en el anexo D

#### Agente SNMP

El agente SNMP esta construido siguiendo la arquitectura del protocolo mostrada en la sección 3.2.2 con la implementación de solo dos aplicaciones, respondedor de comandos y orginador de notificaciones. El agente posee el software necesario para construir las tablas SNMP base, definidas en la MIB. El código fuente utiliza como base el desarrollo realizado por [Charbonneau, 2013] al cual se agregan funciones de la librería pysnmp [Etingof, 2017]. En el siguiente código se muestra la creación del motor SNMP, el socket UDP, los parámetros de configuración del agente, la importación de MIB, la creación de objetos escalares y las funciones para responder ciertos tiempos de PDU SNMP a través del despachador.

```
class SNMPAgent(object):
    def __init__(self, mibObjects):
        # Create a SNMP engine
        self._snmpEngine = engine.SnmpEngine()

        # Open a UDP socket to listen for snmp requests over IPv4
        config.addTransport(self._snmpEngine, udp.domainName,
                                udp.UdpTransport().openServerMode(
                                (ipAddress, 161)))
```

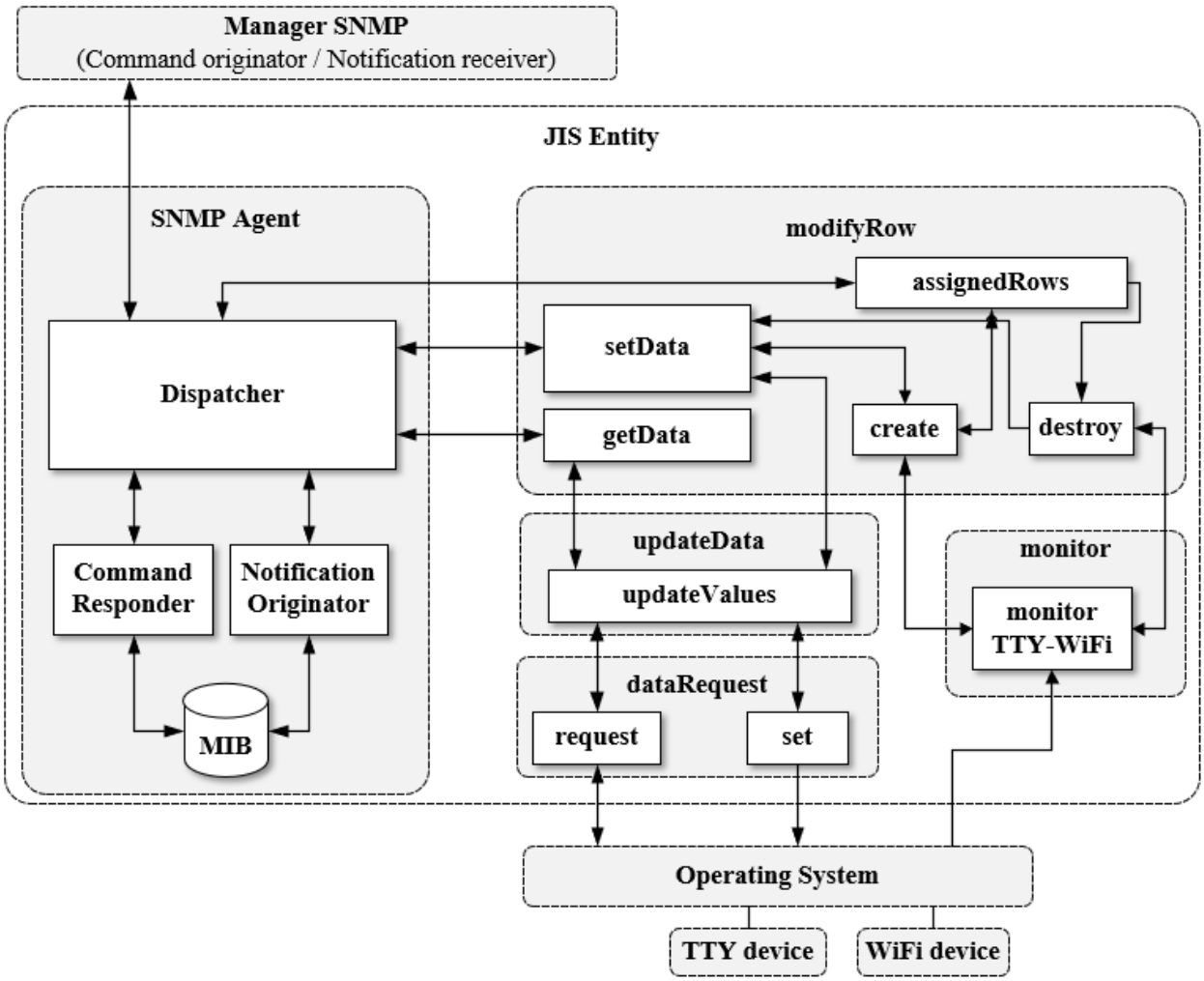


**Figura 5-13**.: Arquitectura del protocolo diseñado
Fuente: Propia

```
# SNMPv2c setup
config.addV1System(self._snmpEngine, "agent", "public")

# Allow read MIB access for this user / securityModels at VACM
config.addVacmUser(self._snmpEngine, 2, "agent", "noAuthNoPriv",
                   (1,3,6,1,4,1), (1,3,6,1,4,1))

# Create an SNMP context
self._snmpContext = context.SnmpContext(self._snmpEngine)

#the builder is used to load mibs
mibBuilder = self._snmpContext.getMibInstrum().getMibBuilder()
mibSources = mibBuilder.getMibSources() +
             (builder.DirMibSource('.'),)
```

```
        mibBuilder.setMibSources(*mibSources)

        #variables will subclass this since we only have scalar types
        #can't load this type directly, need to import it
        MibScalarInstance, =
            mibBuilder.importSymbols('SNMPv2-SMI', 'MibScalarInstance')
        #export our custom mib
        for mibObject in mibObjects:
            nextVar, = mibBuilder.importSymbols(mibObject.mibName,
                                                mibObject.objectType)
            instance = createVariable(MibScalarInstance,
                                      mibObject.valueFunc,
                                      nextVar.name, (0,),
                                      nextVar.syntax)
            #need to export as <var name>Instance
            instanceDict = {str(nextVar.name)+"Instance":instance}
            mibBuilder.exportSymbols(mibObject.mibName,
                                     **instanceDict)

        cmdrsp.GetCommandResponder(self._snmpEngine, self._snmpContext)
        cmdrsp.SetCommandResponder(self._snmpEngine, self._snmpContext)
        cmdrsp.NextCommandResponder(self._snmpEngine, self._snmpContext)
        cmdrsp.BulkCommandResponder(self._snmpEngine, self._snmpContext)

    def dispatcher(self):
        self._snmpEngine.transportDispatcher.jobStarted(1)
        try:
            self._snmpEngine.transportDispatcher.runDispatcher()
        except:
            self._snmpEngine.transportDispatcher.closeDispatcher()
            raise
```

### monitor

A pesar que no es una función particular del protocolo, se diseño un modulo encargado de monitorear los cambios en las interfaces activas TTY del dispositivo donde corre este código usando la librería pyudev [9]. Si un nuevo dispositivo es detectado, se crea un nuevo dispositivo en la tabla SNMP mediante la clase *Create()*, en el cual se indica el tipo de dispositivo y su dirección física TTY. Cuando un dispositivo es desconectado, se elimina de la tabla SNMP usando la clase *Destroy()*. Ambas clases son importadas del modulo *modifyRow*.

```
import pyudev
from pyudev import Context, Monitor, MonitorObserver
from modifyRow import *
```

[9] https://pyudev.readthedocs.io

```
context = pyudev.Context()
monitor = Monitor.from_netlink(context)
monitor.filter_by(subsystem='tty')

for action, device in monitor:
    if action == 'add':
        print (action, device.get('DEVNAME'))
        newDevice = Create()
        newDevice.create('hcsrId', device.get('DEVNAME'))
    else:
        print (action, device.get('DEVNAME'))
        newDevice = Destroy()
        newDevice.destroy(device.get('DEVNAME'))
```

### updateData

La función principal de este modulo es actualizar la información de gestión de todos los dispositivos en las tablas SNMP, siendo la entidad que interactua con los protocolos SNMP y serial. Para ello ejecuta un ciclo infinito de verificación sobre la tabla SNMP, indagando los dispositivos que allí se encuentran creados a través de la función *assignedRows* importada desde el método *modifyRow*, la cual retorna una lista con los números de instancia asignados a cada dispositivo.

```
def assignedRows (mibObjectIndex):
    # Calculates how many instances there are in a table and returns
    # the index of each one
    rowInstanceId = 1
    statusTableIndex = []
    print('Checking the assigned rows in the TLON-MIB...')

    while rowInstanceId < 32:
    # this number is the limit of sensor of each table
        errorIndication, errorStatus, errorIndex, varBinds = next(
            getCmd(SnmpEngine(),
                   CommunityData(community),
                   UdpTransportTarget((server, port)),
                   ContextData(),
                   ObjectType(ObjectIdentity(mibName,
                                             mibObjectIndex,
                                             rowInstanceId)))
        )
        if errorIndication:
            print(errorIndication)
        elif errorStatus:
            print('%s at %s' %(errorStatus.prettyPrint(),
```

```
                                errorIndex and
                                varBinds[int(errorIndex) - 1][0] or '?'))
        else:
            for varBind in varBinds:
                x = varBind[1]
                if x :
                    statusTableIndex.append(rowInstanceId)
        rowInstanceId += 1
    return statusTableIndex
```

Posteriormente con la información de la tabla, los objetos (representados en columnas) e instancias asignadas (números de fila) se genera un ciclo de lecto-escritura a través de la función *readValue*, en donde se crea un objeto de clase *Request()* importada del método *dataRequest*, el cual permite realizar las solicitudes de lectura de cada uno de los objetos cuyas respuestas son escritas en la tabla SNMP mediante un comando *setData*. Durante cada transacción se verifica si funciones de creación o destrucción de instancias en la tabla están siendo ejecutadas para evitar errores al intentar consultar o escribir información sobre filas que están siendo modificadas. Esto se realiza mediante la consulta de un registro en un archivo de texto plano.

```
def readValue (device, mibObjectData, cycles):
# Puts the data in all columns of each object
    data = Request()      # Request is a class from dataRequest
    for rowInstanceId in cycles:
        # Checks if the SNMP table is blocked by 'Destroy' function
        register = open('/home/pi/code/register.txt', 'r')
        validation = register.read()
        register.close()
        if validation == '0':
            # Applies the data on the ObjectType indicated
            address = getData(device+'Address', rowInstanceId)
            setData(mibObjectData,
                    data.sensorData(mibObjectData, address, ['read', 0]),
                    rowInstanceId)
        else:
            print('The SNMP database is busy it will restart')
            break
```

### modifyRow

Este método contiene tanto las funciones necesarias para operar sobre las tablas SNMP como para enviar notificaciones o traps indicando algún tipo de modificación.
La creación de un nuevo objeto en la tabla se realiza mediante la función *create* de la clase *Create*, la cual invoca otras operaciones de dicha clase para asignar la identificación de

instancia con *newSensorId*, colocar valores por defecto en cada uno de los nuevos objetos escalares para mantener la consistencia de la tabla SNMP con *valuesList* y configurando los valores en la tabla SNMP mediante un comando de *setData* enviado al agente respondedor de comandos.

```
class Create:
(...)
    def create (self, indexTable, addressId):
    # Main method of this class, it verifies the actual data into SNMP
    # table and use it to know the information to create the row
        print ('Creating a new row for the sensor', indexTable)
        sensorId = self.newSensorId(indexTable)
        objects = self.objectsList(indexTable)
        values = self.valuesList(indexTable, sensorId, addressId)

        for iteration in range (0, len(objects)):
            setData(objects[iteration],
                    values[iteration],
                    sensorId
                   )
        trapMessage = 'A new sensor was created'
        sendingNotification(trapMessage)
        print ('The sensor ', indexTable, 'was created with ID', sensorId)
```

Cuando un nuevo dispositivo ingresa a la red, se busca una fila disponible para ser asignada mediante un comando *setRequest*, modificando el valor de *RowStatus* en *4* o *createAndGo*. Esta nueva fila asigna un nuevo OID dentro de la rama de la tabla, por ejemplo el dispositivo BMP180 tiene OID 1.3.6.1.4.1.49843.1.1.1.2.1, un nuevo dispositivo de este tipo se le asignará 1.3.6.1.4.1.49843.1.1.1.2.1.1 (agrega una rama) y si ingresa un adicional quedará con 1.3.6.1.4.1.49843.1.1.1.2.1.2. Cuando el dispositivo sale de la red se hace el proceso de búsqueda de su fila y se coloca el valor de *6* o *destroy* en el *RowStatus*, liberando el recurso que podrá ser usado con nuevos accesos a la red.

La destrucción de un objeto en la tabla SNMP se realiza cuando el dispositivo sale de la red usando la clase *Destroy*. Lo primero que se hace es bloquear la tabla con la función *blockTable* para evitar modificaciones generadas por el proceso *updateData* que puedan generar inconsistencias y errores en el servicio, mediante un registro en archivo de texto plano. La función *destroy* se apoya en otras funciones previamente descritas para realizar la búsqueda de la dirección del dispositivo en la tabla SNMP y así conocer su número de instancia. Una vez obtenida la identificación, se elimina de la tabla SNMP mediante la función *setData*, donde se asigna el valor de 6 en el objeto tipo *rowStatus*, con lo que la fila completa es eliminada de la tabla, liberando el recurso que podrá ser usado con nuevos accesos a la red.

```
class Destroy:
(...)
    def destroy (self, addressToDelete):
        print ('It will be remove from the dinamyc SNMP table...')
        assignedRowsList = []

        for indexTable in self.mibObjectList:
        # Search in all tables (indexTable) created in the mib
            # Verify the rows assigned in each table

            assignedRowsList = assignedRows(indexTable+'Id')
            if len(assignedRowsList) > 0:
                for indexRow in assignedRowsList:
                    addressAsigned = getData(indexTable+'Address',
                                             indexRow)
                    if addressAsigned == addressToDelete:
                        self.blockTable('1')
                        setData(indexTable+'TableStatus', 6, indexRow)
                        self.blockTable('0')
                        trapMessage = 'A sensor was deleted'
                        sendingNotification(trapMessage)
                        print('The row of the sensor was deleted')
                        break
```

Cada vez que se realiza una actualización en la tabla SNMP, bien sea por la creación o destrucción de un objeto, o por la actualización de la información de los objetos, se llama la función *sendingNotification*, la cual envía una notificación o trap SNMP a un servidor de traps externo a a través del puerto UDP 162, indicando el tipo de modificación realizado.

```
def sendingNotification (trapMessage):
    errorIndication, errorStatus, errorIndex, varBinds = next(
        sendNotification(
            SnmpEngine(),
            CommunityData('public'),
            UdpTransportTarget(('192.168.184.1', 162)),
            ContextData(),
            'trap',
            NotificationType(
                ObjectIdentity(mibName, 'alterTable'),
                                objects={('TLON-MIB', 'anyUpdate'):
                                         trapMessage,
                                        }
                                )
            )
        )
    if errorIndication:
```

```
        print(errorIndication)
```

### dataRequest

Este es el método encargado de la comunicación directa con los dispositivos no-SNMP, usando su medio, protocolo y formato específico, serializando la información para enviarla por un puerto TTY o creando un socket UDP para comunicaciones inalámbricas con lo que se ejecutan solicitudes de información cuyas respuestas son recolectadas y formateadas para que puedan ser colocadas dentro de la PDU SNMP.

Todo se realiza mediante la clase *Request* el cual posee una función llamada sensorData, la cual verifica el objeto que debe ser gestionado y busca la función correspondiente dentro de la estructura de la clase.

```
class Request:

    def sensorData (self, mibObjectData, address, dataReq):
        method = getattr(self, mibObjectData, lambda: "Invalid MIB Object Data")
        return method (address, dataReq)
```

Como cada dispositivo tiene un método de conexión particular con formatos y mensajes específicos, la estructura de la clase contiene un listado amplio de funciones, cada una corresponde a un objeto a gestionar, cada una llama a una función principal que contiene las instrucciones de conexión correspondiente. Por ejemplo, si se recibe una solicitud de consulta de un sensor de temperatura, la función *bmpTemp* es solicitada. Esta le da formato al mensaje y se apoya en una función de transporte llamada *bmp*, la cual crea un socket UDP para enviar el mensaje a la dirección IP del dispositivo. Una vez el mensaje es respondido, se extrae la información requerida y se retorna, luego de dar el formato requerido.

```
def bmp (self, address, message):
        data = ""
        port = 23050

        # Creates a socket, sends the message and waits until the total
        # answer is received, next the connection is closed
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sent = sock.sendto(message, (address, port))
            data, server = sock.recvfrom(1024)
            sock.close()
            print (type(data), data)
            if data == b"error 100":
                data = self.bmp(address, message)
        except:
```

```
            print('It was not able to stablish communication with the peer\n')
            data = '0'

        return data

    def bmpTemp (self, address, dataReq):
        message = b'temperature' # keyword to send in the message
        data = self.bmp(address, message)
        return float(data)
```

### 5.5.2. Recolección de datos por adelantado

La detección automática de un nuevo dispositivo genera una nueva fila dentro de la tabla de acuerdo al tipo de dispositivo. Un proceso encargado de validar las máquinas registradas en las tablas, hace consultas periódicas de cada objeto de gestión en cada dispositivo, actualizando cada uno de los campos en la tabla para mantener su consistencia con la información de gestión. Así cuando el gestor SNMP realiza alguna consulta mediante un OID determinado, el agente respondedor estará en capacidad de responder de inmediato. A esto denominamos consulta de información de gestión adelantada, puede que la información almacenada se use o puede que no, pero el proceso permite incrementar la efectividad en la entrega de la información de gestión así como su tiempo de respuesta. Si bien, hay algo de generación de tráfico para obtener la información previamente, esta modalidad permite el desarrollo de procesos de gestión distribuidos y detección automática de cambios de topología (entrada/salida de host en la red) que podrían ser ejecutados por diferentes dispositivos dentro de la red Ad Hoc, por ejemplo un dispositivo puede tomar el rol de agente respondedor y otro puede ser el colector adelantado, y en caso que el primero mencionado salga de la red, otro podrá asumir su funciones, incluso el mismo colector.

# 6. Implementación y resultados

Se organizó un plan de pruebas de aceptación para realizar la verificación del protocolo desarrollado, cuyas actividades se contemplan de igual forma en dos escenarios para el despliegue del servicio, usando equipos físicos, elementos virtualizados y aplicaciones de gestión de red, con el fin de comprobar la escalabilidad del protocolo aplicándolo sobre dos interfaces de comunicación diferentes.

- Conectividad entre nodos
- Levantamientos de procesos entre host
- Configuración en software de gestión
- Generación de poll SNMP y toma de gráficas
- Recepción de traps
- Captura de paquetes con sniffer

## 6.1. Topología Ad Hoc

Se implementó una red en topología Ad Hoc tipo WSN, la cual se muestra en figura **6-1**) cuyos enlaces muestran la comunicación directa que se dispuso para la prueba, pero que en realidad fue una red mesh completa. En este escenario se distribuyeron las funciones del protocolo en diferentes dispositivos, dejando el AGENTE-DTE en un host y una de las funciones de actualización de datos sobre el primero mencionado, mostrado en la figura . Aquí se usaron los siguientes componentes:

- Raspberry Pi 3B con sistema operativo Raspian
- Raspberry Pi Zero W con sistema operativo Raspian
- Sensores BMP180
- Portátil con software de gestión SNMPc y MIB Browser

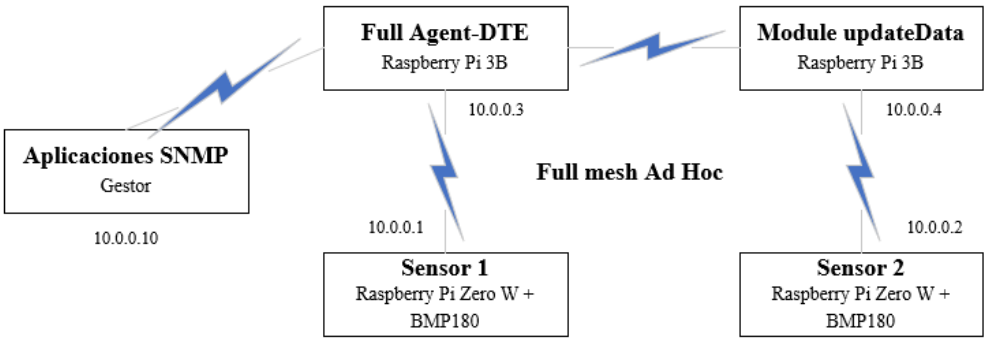


**Figura 6-1**.: Escenario de pruebas sobre red Ad Hoc
Fuente propia

Los sensores BMP180 quedaron conectados a las Raspberry Pi Zero W, cuya comunicación se realiza por WiFi mediante protocolo UDP en puerto 23050. La configuración de estos dispositivos se muestra en el anexo E aunque está fuera del alcance de este proyecto, pero fue necesaria su implementación para realizar las pruebas correspondientes. Una vez ingresan a la red, se crearon de forma dinámica las filas en la tabla SNMP para dicho dispositivo, ampliando la capacidad de registros almacenados en la misma. Cuando ocurre esto, se genera una notificación SNMP, la cual es recibida por una aplicación receptora de traps (ver anexo F.2).

En una máquina se dispuso de dos aplicaciones de gestión SNMP para realizar las consultas sobre dicho protocolo. Por un lado de uso SNMPc el cual permitió capturar la información de la tabla *bmpTable* donde quedaron registrados los datos de los dos bmp180 (ver figura **6-4**) los cuales pudieron ser usados para generar gráficas de gestión históricas, por ejemplo comparando las dos mediciones de temperatura (ver figura **6-5**). Por otro lado la aplicación MIB Browser se uso para realizar consultas SNMP mediante diferentes comandos como walk o bulk

Se realizaron pruebas exitosas bajando el proceso del agente SNMP (jisAgent), sin que otros procesos se detuvieran o generarán errores que interrumpieran la comunicación, como se ve en la figura **F-1**. Esto permite que a futuro los procesos de movilidad o intercambio de roles no se vea afectado.

## 6.2. Topología Infraestructura

En este escenario se combino el uso de equipos heredados con el de sensores, conectados en una topología centralizada de infraestructura explicada en la figura **6-6**, usando conexión

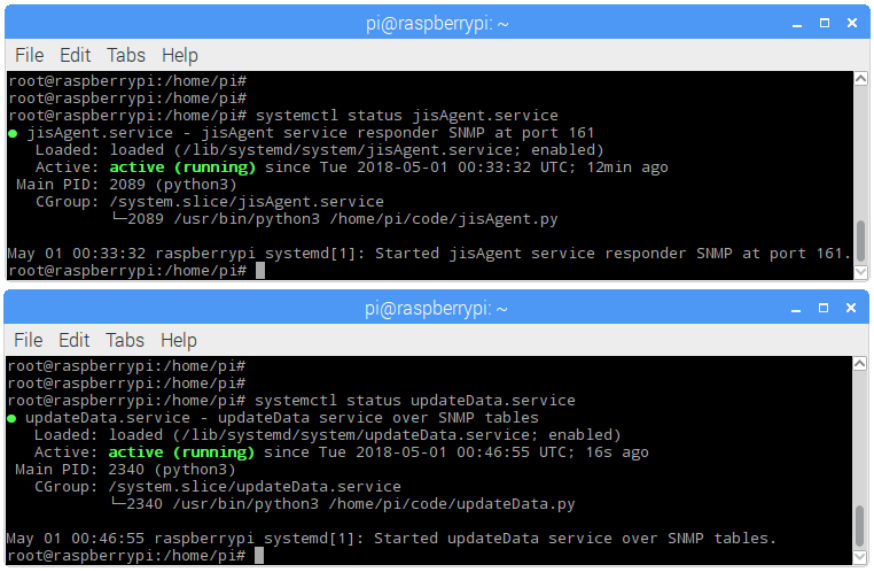


**Figura 6-2**.: Servicios de gestión jisAgent y updateData
Fuente propia

serial RS232 y serial USB. Aquí se usaron los siguientes componentes:

- Módem satelital Comtech CDM600
- Switch de terminales Moxa NPort 6650
- Raspberry Pi 3B con sistema operativo Raspian
- Arduino Uno
- Sensor de proximidad HC-SR04
- Portátil con software de gestión SNMPc y MIB Browser, así como el programa de virtualización VMWare

En el servidor encargado de las conexiones RS232, se configuró un servicio de virtualización, donde se ejecutan las operaciones del agente-DTE. Para que este pueda comunicarse con los dispositivos heredados. Se mapean los puertos de comunicaciones físicos y virtualizados. La conexión implementada, hace uso de un switch de terminales, el cual amplia la capacidad de la infraestructura y permite conectar hasta 16 equipos RS232, RS422 o RS485 (ver figura **6-7**).
Dado que los equipos heredados operan en redes centralizadas sin condiciones de movilidad como una red Ad Hoc, se crea el dispositivo a gestionar manualmente, en donde el agente asigna una instancia en la tabla SNMP con lo que el módulo de actualización de datos *updateData*, realiza la conexión serial y se establece el intercambio de mensajes de gestión,

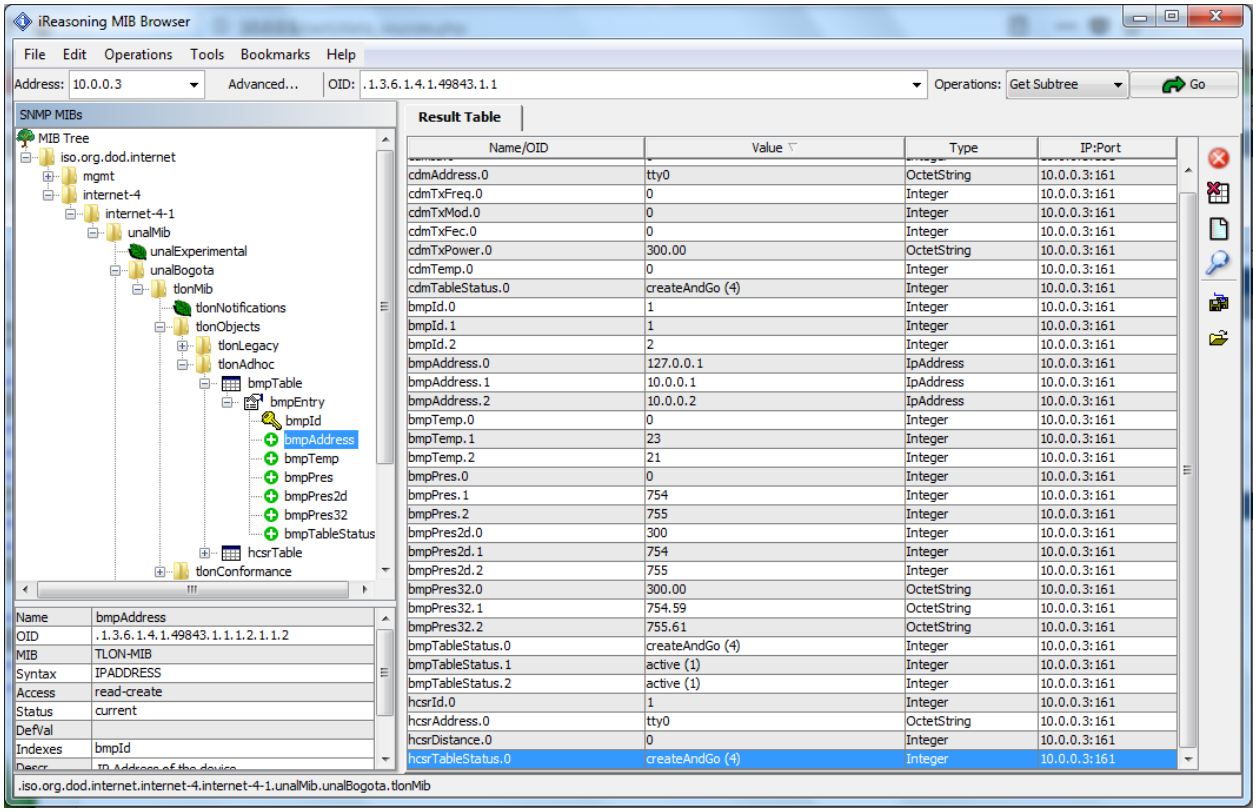


**Figura 6-3**.: Comando walk SNMP para consultar datos en toda la MIB
Fuente propia

algunos de ellos son mostrados en la figura **F-2** , cuya información es actualizada en la tabla utilizando el OID correspondiente para cada objeto. Cada vez ocurre esto, un mensaje de notificación tipo TRAP es enviado al servidor de traps, al igual que en el escenario anterior. Entre tanto, una vez es detectado el sensor de proximidad, se crea el objeto de forma automática en la tabla SNMP usando sus OID correspondientes y al ser desconectado, el sistema elimina la información, liberando el recurso asignado incialmente. Una captura de los mensajes de este proceso es mostrada en la figura **6-8**

Se verificó la respuesta del sistema ante solicitudes de cambio en uno de los objetos a través de mensajes *setRequest*. No todos los objetos son susceptibles de ser modificados, sobre todo aquellos que están solo sensando algún tipo de variable. Por ello, las pruebas se realizaron sobre el módem satelital cambiando la configuración del modcod cuyo objeto *cdmTxMod* es un entero entre 0 y 5, según se define en la MIB. En la figura **6-9** se evidencia el resultado de dos consultas al OID del objeto indicado, previo y posterior al cambio a través del mensaje SNMP.

Mientras se realizaba el proceso de gestión se tomaron capturas con un sniffer de tráfico lo cual permitió ver el flujo de mensajes request, response y traps, cuyas imágenes se muestran en el anexo F.3

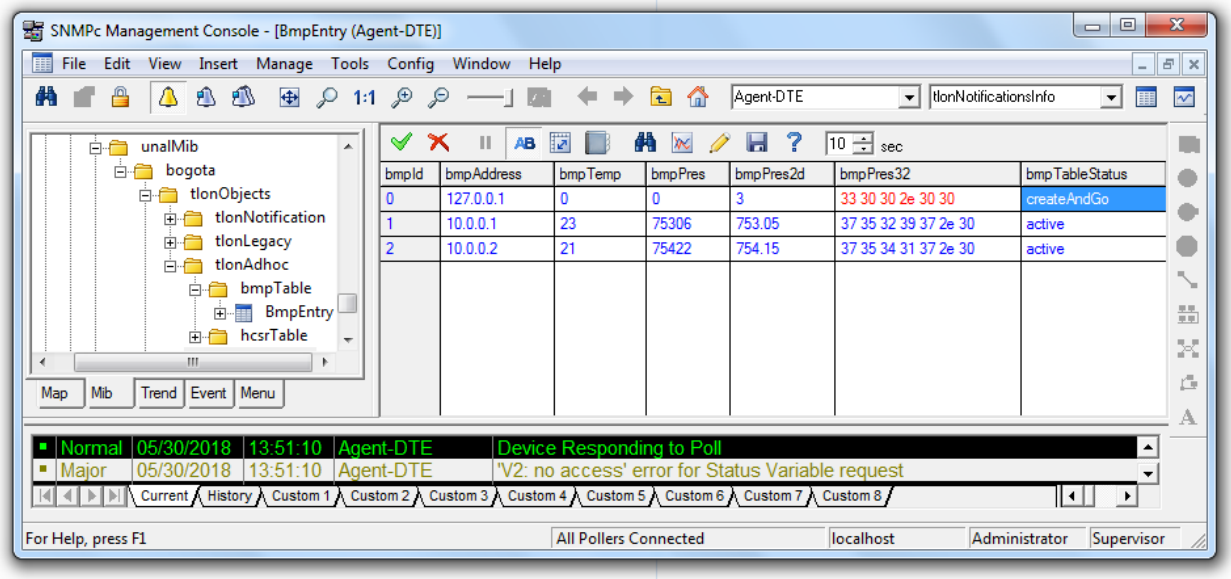


**Figura 6-4**.: Gestión de tabla bmpTable
Fuente propia

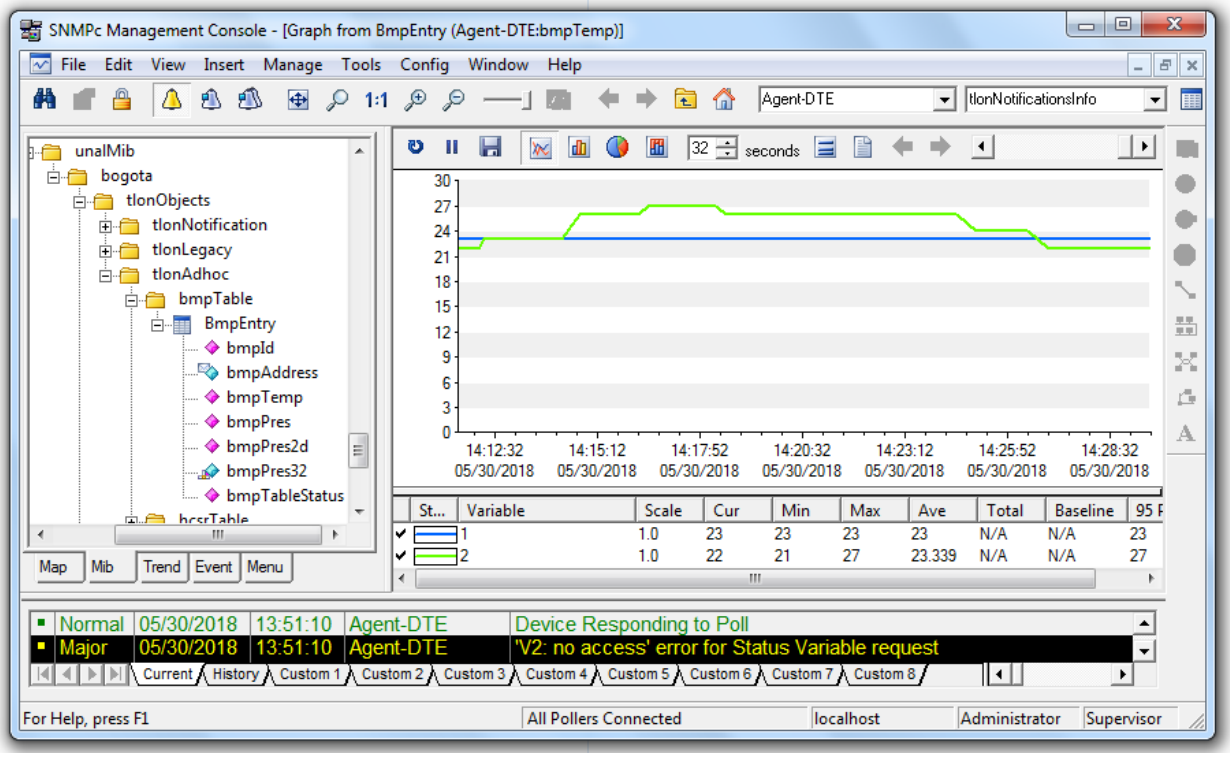


**Figura 6-5**.: Gráfico de monitoreo histórico de temperatura
Fuente propia

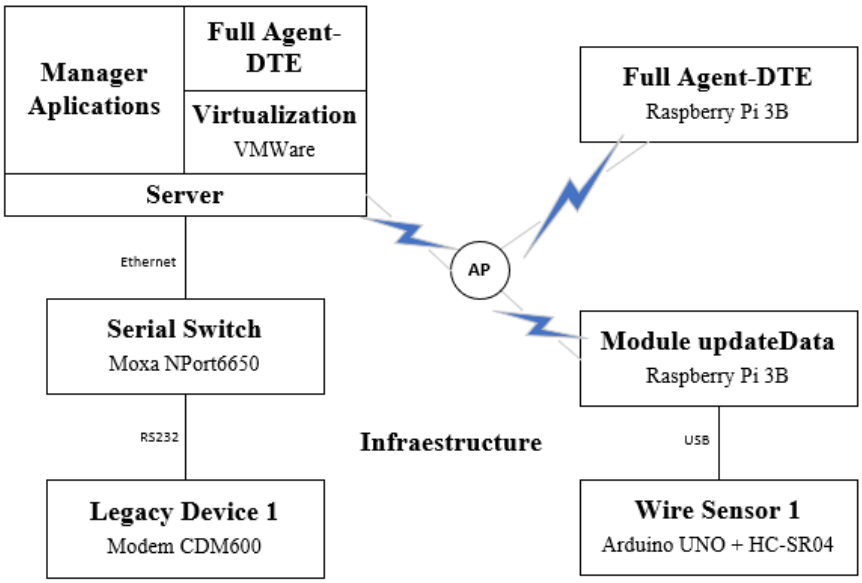


**Figura 6-6**.: Escenario en topología Infraestructura
Fuente propia


**Figura 6-7**.: Montaje de equipos heredados
Fuente propia

```
root@raspberrypi:/home/pi/code# python3 monitorTTY.py
add /dev/ttyACM0
------------------
Creating a new row in the table for the sensor hcsrId
Checking the assigned rows in the TLON-MIB...
Updating data into the SNMP table
Updating data into the SNMP table
Updating data into the SNMP table
Updating data into the SNMP table
The sensor  hcsrId was created with ID 2

remove /dev/ttyACM0
------------------
The device with address /dev/ttyACM0 has gone out of the network.
It will be remove from the dinamyc SNMP table...
Checking the assigned rows in the TLON-MIB...
Checking the assigned rows in the TLON-MIB...
Updating data into the SNMP table
The row of the sensor hcsr with index 2 was deleted
Checking the assigned rows in the TLON-MIB...
```

**Figura 6-8**.: Detección de ingreso y salida de dispositivos TTY
Fuente propia

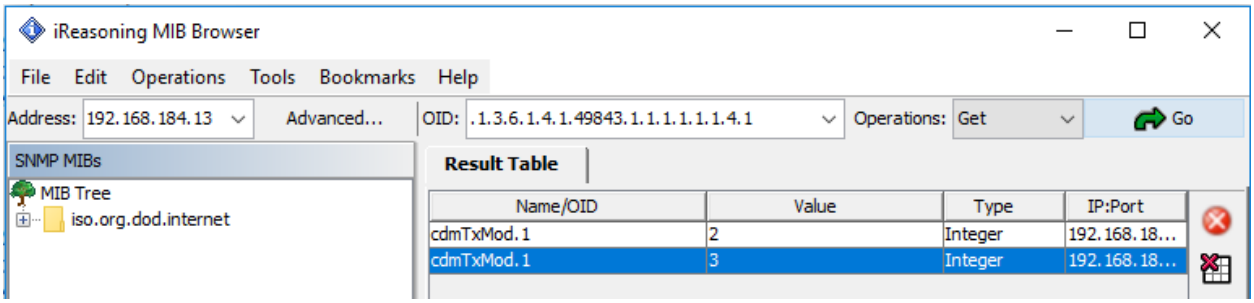


**Figura 6-9**.: Consulta en MIB previo y posterior
Fuente propia

# 7. Conclusiones y recomendaciones

## 7.1. Conclusiones

La metodología de conversión de protocolos permitió definir la estructura del servicio y el protocolo, los métodos de descripción formal y especificaciones del servicio y del protocolo, que combinados con el levantamiento de requerimientos de usuario, se logro hacer énfasis en el desarrollo de funciones que afrontan los desafíos que imponen las redes dinámicas distribuidas. La arquitectura del servicio diseñado permite:

- La conversión de múltiples protocolos a SNMP debido a que el intercambio de mensajes entre ellos no es realizado directamente sino como procesos independientes, que a su vez, facilitan el despliegue en redes distribuidas

- El concepto de recolección de datos por adelantado usa las tablas SNMP de representadas en la MIB como base de datos del servicio, por lo que la utilización de motores de base de datos, dependiendo de su contexto de uso, para almacenamiento temporal de información de gestión no es necesario

- El método propuesto para capturar la información de gestión es ejecutado de manera adecuada ya que se presta un servicio en nivel de aplicación que no interviene en el establecimiento o mantenimiento de la conexión entre los host y solo se requiere de la inclusión de funciones en la clase que realiza la comunicación entre los agentes usando el formato especifico

Durante el desarrollo de este protocolo de gestión se identificaron algunos desafíos sobre los cuales se deberían enfocar desarrollos de este tipo para que el servicio de gestión basado en SNMP pueda ser utilizado en una red Ad Hoc:

- Heterogeneidad y auto-configuración: La base de información de gestión es diseñada y construida conociendo el equipo, lo que hace y lo que se puede medir. Cuando una red es heterogénea, lo complicado no es administrar gran diversidad de dispositivos, lo difícil es abarcar la gestión de equipos que no son conocidos ya que no están incluidos en la MIB. El reto futuro es crear y/o modificar de manera autónoma las MIB acorde a las características de cada host.

- Dinamismo: El protocolo SNMP permite realizar conexiones dinámicas, omitiendo el uso de OID fijos para cada objeto, ya que no se conoce la cantidad de host que van a operar dentro de la red, a su vez que facilita el reuso de recursos físicos y lógicos utilizando tablas dinámicas con objetos de tipo *rowStatus*

- Distribución: El diseño permite que la colección de datos de gestión sean en las cabeceras de los cluster, quienes a su vez podrán compartir la información con otros nodos en el mismo cluster o en otro, permitiendo que la cantidad de datos de gestión que circulan a través de la red sea menor. Se evidencia que el desarrollo permite integrar la gestión de una red con sistemas de diferente naturaleza o incluso dispositivos no-SNMP sin importar el tipo de topología

El resultado de este trabajo de investigación podrá servir como herramienta de mejora en los procesos de administración y operación de la infraestructura de las empresas de telecomunicaciones que usen sistemas heredados, haciendo más eficiente el monitoreo de las variables de operación de los equipos debido a que:

- Se facilitará la inclusión de los dispositivos en los sistemas de gestión que ya se disponen por el uso de protocolos estandarizados

- Mejorará la calidad del servicio en temas de disponibilidad ya que el registro en línea de la información servirá para la detección temprana de incidentes o diagnostico reactivo de los mismos, eliminando o reduciendo los tiempos de afectación

- Ayudará en el control de cambios debido a que se podrá tener acceso remoto a los equipos evitando el desplazamiento de personal técnico hasta los sitios donde estos se encuentren

- Se abrirán oportunidades para eliminar, mitigar o trasladar nuevos riesgos que podrían aparecer y que de otra forma no lograrían serían detectados

- Reducirá los costos de operación y mantenimiento debido a la automatización de los procesos de administración y evitando la compra de nuevos equipos o sistemas de gestión independientes

## 7.2. Recomendaciones

Los módulos funcionales de la arquitectura del protocolo podrán ser puestos en contenedores de software para que operen en diversos ambientes de desarrollo o producción que requieran simplificar los procesos distribuidos. También se requiere explorar las opciones de movilidad del código para que se pueda compartir la información de gestión o realizar procesos de asignación de roles dentro de la red, así como explorar las opciones de auto-configuración de las MIB.

El uso de TEXTUAL-CONVENTION en las MIB puede facilitar la creación de tipos de datos que requieran los agentes dentro del proyecto TLÖN, para describir parámetros propios o particulares del sistema. Por ejemplo, se podrían representar los sentimientos que un agente pueda tener dentro de un ambiente o entorno especifico.

Durante la implementación de los escenarios de pruebas se usaron diferentes tipos de hardware basados en el chip ESP8266[1] para montar la topología Ad Hoc, encontrando limitación en el funcionamiento de sus interfaces de red ya que solo permiten conexiones de tipo infraestructura como *access point*, *estación* y *soft-ap* (operación dual, como AP y estación simultánea). Este último permite crear conexiones mesh pero no puramente Ad Hoc y con restricción de cantidad de enlaces establecidos. Dichos equipos son una alternativa de bajo costo en las redes de sensores, pero dada la limitación mencionada, se debe evitar su uso para implementaciones en el proyecto TLÖN o en las que se requiera conectividad completamente Ad Hoc. Se recomienda el uso dispositivos como Raspberry Pi Zero W, que también son de bajo costo y con facilidades de configuración, administración, desempeño, costo y versatilidad.

[1] http://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

# A. Anexo: Requerimientos de usuario

## A.1. Tabla de especificación de requerimientos de interesados

**Tabla A-1**.: Especificación de requerimientos de interesados

| ID | Definición | Prioridad | Dependencia | Origen | Dificultad | Tipo |
|---|---|---|---|---|---|---|
| 01 | El usuario podrá usar cualquier gestor de red SNMP para integrar el protocolo y monitorear las variables de operación de los equipos | Alta | Ninguna | Contexto de uso | Difícil | Interfaz |
| 02 | El protocolo desarrollado debe ser configurable para que el usuario pueda agregar nuevos dispositivos o ampliar las variables de operación de un equipo gestionado mediante este sistema | Alta | 01 | Efectividad | Difícil | Funcional |
| 03 | El usuario debe tener la facilidad de modificar los parámetros de configuración del equipo gestionado | Alta | 01,02 | Ciclo de vida operacional | Nominal | No funcional - factor humano |
| 04 | El sistema deberá enviar notificaciones de los sensores que el usuario crea convenientes | Baja | 01 | Escenario operacional | Nominal | Funcional |
| 05 | Los eventos de envío de notificaciones, errores del sistema, modificación de parámetros de configuración de los equipos deberán ser registrados en un log de eventos | Baja | 01,02,03,04 | Desempeño | Fácil | Funcional |
| 06 | La solución debe tener documentación técnica | Baja | Todas | Características de usuario | Fácil | Req. de proceso |

**Tabla A-1**.: Especificación de requerimientos de interesados (continuación)

| ID | Definición | Prioridad | Dependencia | Origen | Dificultad | Tipo |
|---|---|---|---|---|---|---|
| 07 | El protocolo debe gestionar los dispositivos a través de dos interfaces para mostrar adaptabilidad, una de ellas debe ser puerto serial RS232 o RS485, la otra puede ser WiFi | Media | Ninguna | Efectividad | Difícil | Limitación de diseño |
| 08 | La implementación del protocolo debe ser segura para no afectar servicios que puedan estar operando | Media | 10 | Ciclo de vida operacional | Nominal | Requerimiento de proceso |
| 09 | El protocolo desarrollado debe operar con la versión 1 del SNMP. La operación en versiones 2 y 3, son opcionales | Alta | 01 | Desarrollo operacional | Nominal | Limitación de diseño |
| 10 | Las pruebas de funcionamiento deben hacerse en un ambiente controlado | Media | Ninguna | Contexto de uso | Fácil | No funcional - factor humano |
| 11 | El desarrollo debe ser hecho en Python dado que los ambientes y agentes que se encuentran diseñados en el sistema TLÖN fueron construidos en dicho lenguaje de programación | Alta | 13 | Desarrollo operacional | Nominal | Limitación de diseño |
| 12 | El protocolo desarrollado debe correr sobre el protocolo de capa inferior BATMAN | Baja | 13 | Desarrollo operacional | Nominal | Funcional |
| 13 | El protocolo debe operar sobre el sistema operativo Linux | Alta | Ninguna | Escenario operacional | Nominal | Limitación de diseño |
| 14 | El usuario debe recibir información de instaladores para complementar su sistema de virtualización y así crear el contenedor Docker (python, pip, comandos de red, etc) | Media | 06 | Características de usuario | Fácil | Requerimiento de proceso |
| 15 | El protocolo desarrollado debe ser ligero para que pueda operar con bajos recursos de desempeño y ser fácilmente transportable | Baja | 11 | Características de usuario | Nominal | No funcional - calidad |
| 16 | Debe funcionar en una red Ad Hoc, con características de autoconfiguración y distribución | Alta | 7, 11 | Desarrollo operacional | Nominal | Funcional |

## A.2. Tabla análisis de riesgos de requisitos de usuario

**Tabla A-2**.: Análisis de riesgos de requerimientos de interesados

| StRS | Riesgo | Probabilidad | Impacto | Prioridad | Tratamiento |
|---|---|---|---|---|---|
| 01 | Desarrollo no estándar al protocolo SNMP por lo que algunas consultas no se ejecuten apropiadamente o no se reciba la información solicitada | Improbable | Severo | Medio | Uso de librerías y métodos pysnmp que han sido desarrollados bajo el estándar del protocolo SNMP |
| 01 | La MIB diseñada tenga problemas de compatibilidad con el gestor | Posible | Severo | Alto | Probar la sintaxis de la MIB diseñada mediante un software especializado como MIB Smithy o smilint |
| 01 | Posible incompatibilidad entre sistemas operativos | Improbable | Menor | Bajo | Usar protocolos estándar de comunicación en capas inferiores para garantizar que el flujo de información del protocolo SNMP se traslade del gestor al agente |
| 02 | El usuario realice cambios de bajo nivel que afecten el desempeño del servicio | Posible | Moderado | Medio | Se debe permitir únicamente realizar modificaciones en la gestión de dispositivos y sus variables (es decir, desde el gestor más no en el agente), sin modificar ni alterar el módulo raíz del protocolo. En ciertos dispositivos se podría dejar autoconfigurable |
| 02 | Configuración de nuevos dispositivos no es controlado por lo que no hay garantía que funcione adecuadamente | Posible | Severo | Alto | El desarrollo debe permitir el uso de ciertos parámetros ya creados en la MIB y aquellos que estén fuera de ese estándar no se deben permitir. En caso de ser necesario, se debería diseñar una nueva MIB o modificar la existente, por lo que el usuario deberá tener las competencias para dicho desarrollo |

**Tabla A-2.**: Análisis de riesgos de requerimientos de interesados (continuación)

| StRS | Riesgo | Probabilidad | Impacto | Prioridad | Tratamiento |
|---|---|---|---|---|---|
| 02 | Ejecutar modificaciones sobre el protocolo podrían ser complicadas para el usuario | Probable | Menor | Bajo | Facilidades de modificación en un entorno amigable esta fuera del alcance de la investigación, sin embargo se buscaría la forma de modificar ciertos parámetros con archivos de texto o con la ayuda de la documentación |
| 03 | Generación comandos erróneos que pueden afectar el desempeño de la máquina gestionada | Improbable | Menor | Bajo | Desarrollar el protocolo soportado en la documentación técnica de los componentes y realizar pruebas de aceptación antes de su liberación |
| 03 | Pueden abrirse huecos de seguridad | Posible | Moderado | Medio | Usar los elementos básicos de configuración del protocolo SNMP como versión y comunidad. La seguridad es una característica que se encuentra fuera del alcance del proyecto |
| 04 | No todos los gestores SNMP son receptores de trap por lo que esta opción podría no usarse | Probable | Menor | Medio | El envío de notificaciones será opcional y seleccionable por el usuario |
| 04 | Colapsar memoria del dispositivo conversor por sobrecarga en buffers o gestión de muchos equipos o sensores simultáneamente | Improbable | Moderado | Bajo | Diseñar el algoritmo de tal forma que permita la rápida liberación de memoria utilizada para almacenar información temporal y en casos particulares ni siquiera guarde datos |
| 05 | El log de eventos podría tener una sintaxis difícil de entender | Probable | Menor | Medio | El módulo de registro de datos en un log será diseñado de tal forma que permita conocer fecha, hora y descripción del evento que puede ser técnica, pero de entendimiento para un ingeniero encargado de soportar la solución |

**Tabla A-2**.: Análisis de riesgos de requerimientos de interesados (continuación)

| StRS | Riesgo | Probabilidad | Impacto | Prioridad | Tratamiento |
|---|---|---|---|---|---|
| 05 | Información histórica de eventos en ocasiones no es consultada | Posible | Menor | Bajo | Los eventos son consultados cuando las fallas se presentan, así que a pesar que las consultas no se hagan regularmente, se dejará registro de los eventos más importantes que aparezcan en la operación del protocolo |
| 05 | Archivo de eventos puede ser muy grande con el pasar del tiempo | Posible | Moderado | Medio | Los log podrían guardarse en archivos separados por cantidad de datos o por periodicidad |
| 06 | Mala documentación no permitirá operar adecuadamente la solución | Posible | Moderado | Medio | Se estable una estructura de documentación de la investigación que contempla diseño, desarrollo e implementación |
| 06 | Perdida de información técnica o del desarrollo por la forma o el momento de documentar | Probable | Moderado | Alto | A medida que el desarrollo se va ejecutando se irá registrando la información tanto de diseño y construcción como de desempeño |
| 07 | El dispositivo donde corra el protocolo no tenga suficientes puertos TTY abiertos para las múltiples conexiones seriales | Probable | Severo | Alto | Es una limitación intrínseca del dispositivo que opera como gestor, en tanto se acepta este riesgo. Así mismo, se espera que las conexiones seriales estén limitadas por el tamaño del concentrador serial donde llegan todas las conexiones |
| 07 | Se genere mucha información de gestión que tarde en ser transportada por el tipo de conexión | Improbable | Menor | Bajo | La información de gestión es de tamaño reducido y la periodicidad de consultas varia de 30 segundos a 5 minutos, por lo cual el intervalo entre comunicaciones es bastante alto |
| 08 | La implementación puede generar efectos inesperados durante su homologación en ambientes productivos | Posible | Moderado | Medio | Se organiza un plan de pruebas de aceptación (ATP) a ejecutar en el laboratorio para que cualquier falla del sistema pueda ser corregido antes de pasar al ambiente de producción |

**Tabla A-2.**: Análisis de riesgos de requerimientos de interesados (continuación)

| StRS | Riesgo | Probabilidad | Impacto | Prioridad | Tratamiento |
|---|---|---|---|---|---|
| 09 | El desarrollo se torna complejo si se implementa version 3 debido a los métodos de encripción | Posible | Menor | Bajo | Todo el desarrollo se realizará con soporte en versión 2 ya que es la más utilizada |
| 09 | Se obvien elementos de configuración importantes por el desarrollo de una MIB por cada dispositivo o una sola que los agrupe | Improbable | Menor | Bajo | Se diseñarán varias MIB por cada equipo, con previo análisis de la cantidad de sensores, ya que si son pocos por cada uno, se podrá relacionar en una sola MIB por tipo de conexión (infraestructura o Ad Hoc). En cada una de ellas se tendrán en cuenta los diferentes módulos estándar para que no falten funcionalidades |
| 10 | Se requieren equipos para implementar un laboratorio completo para probar varios escenarios, es posible que no se tengan todos los que se deban o quieran cubrir | Posible | Moderado | Medio | Se contempla el uso de equipos previamente alquilados (de forma gratuita) y se considera la adquisición de algunos componentes de tipo IoT. Si bien el desarrollo busca integrar agentes de diferente índole, el protocolo final será probado en algunos equipos disponibles. Otros que no queden dentro de este esquema, deberá ser desarrollado el código y las MIB para su implementación |
| 11 | Construir programas que no sean eficientes | Posible | Moderado | Medio | Se seguirán las fases de desarrollo de un protocolo de comunicaciones de acuerdo al modelo planteado por Hartmut König |
| 11 | El protocolo tenga muchos errores (bug) y no se identifiquen | Posible | Severo | Alto | Se incluirán los procedimientos de manejo e identificación de errores y aplicación de excepciones recomendados en la documentación de Python |
| 11 | No tenga integración con el sistema TLÖN | Posible | Menor | Bajo | Incluir los requisitos del grupo de investigación TLÖN que tenga mayor prioridad |

**Tabla A-2**.: Análisis de riesgos de requerimientos de interesados (continuación)

| StRS | Riesgo | Probabilidad | Impacto | Prioridad | Tratamiento |
|---|---|---|---|---|---|
| 11 | No se logre programar en este lenguaje por las limitaciones de conocimiento en el mismo. | Improbable | Severo | Medio | Realizar cursos o tutoriales acerca del lenguaje de programación o en su defecto recibir asesoría de estudiantes con mayor experiencia en este ámbito |
| 11 | Se haga un desarrollo que dependa que el dispositivo donde corra el protocolo deba tener Python instalado | Posible | Severo | Alto | Hacer el desarrollo para que se ejecute como un script o un servicio |
| 12 | El protocolo BATMAN se encuentra en fase de desarrollo por lo que pueden aparecer resultados inesperados | Improbable | Moderado | Bajo | La teoría indica que el protocolo BATMAN en capa 2 puede soportar protocolos de capas superiores. Se podrían ejecutar pruebas de desempeño y en caso de ver situaciones inesperadas recopilar la información y socializarla para mejoras del protocolo de enrutamiento o futuras versiones. Se acepta el riesgo y no se manipulará el protocolo de enrutamiento |
| 12 | Se deban hacer modificaciones en interfaces de red para soportar el protocolo BATMAN | Improbable | Moderado | Bajo | El desarrollo podría incluir la elección de interfaces de red diferentes a la WiFi , Ethernet o TTY |
| 13 | Aparezcan problemas de incompatibilidad con otros sistemas operativos ya que en ambientes reales de producción se encuentran sistemas hetereogeneos | Probable | Menor | Medio | El desarrollo se contempla únicamente en Linux, en los casos en que el protocolo deba operar en otros sistemas operativos, sale del esquema que soporta el protocolo, por lo que no se garantiza su óptimo funcionamiento. Desarrollos en otras plataformas están fuera del alcance del proyecto y su riesgo es aceptado |

**Tabla A-2.**: Análisis de riesgos de requerimientos de interesados (continuación)

| StRS | Riesgo | Probabilidad | Impacto | Prioridad | Tratamiento |
|---|---|---|---|---|---|
| 13 | Ciertas tareas no puedan ejecutarse o programarse por desconocimiento de la operación o configuración avanzada de los sistemas basados en Linux | Posible | Menor | Bajo | En los casos en que esto ocurra se recurrirá a la asesoría de estudiantes con experiencia en Linux o consultas con expertos en la materia |
| 14 | Entregar información incompleta | Improbable | Menor | Bajo | Documentar todos los módulos que se utilicen en cada etapa del desarrollo para que se entregue información veraz y completa |
| 14 | El contenedor Docker no soporte alguno de los instaladores | Improbable | Menor | Bajo | Usar módulos estandar y reconocidos |
| 15 | El protocolo sea desarrollado en archivos muy grandes | Posible | Moderado | Medio | Intentar reducir la cantidad de líneas de código del programa, reutilizando métodos y en dado caso, fragmentar cierta información para que se usen múltiples archivos, de los cuales solo se transporten aquellos que requiera el host |
| 15 | Se pierdan conexiones entre archivos cuando se mueva de nodo | Probable | Severo | Alto | Usar una ruta específica de alojamiento local o remoto de archivos con verificación de la existencia de los mismos |
| 15 | Incompatibilidad con el nodo donde se aloje el protocolo | Posible | Severo | Alto | El host donde se aloje el protocolo debe cumplir con ciertos requisitos de hardware y software, lo cual debe estar documentado para que en la integración se contemple cual nodo es elegible como dispositivo principal dentro del cluster Ad Hoc |
| 15 | Se generen problemas de seguridad ya que ciertos nodos tienen información de gestión de otros dispositivos y también podrían ser controlados | Posible | Severo | Alto | Es un riesgo aceptado y debe ser valorado en futuros trabajos |

**Tabla A-2**.: Análisis de riesgos de requerimientos de interesados (continuación)

| StRS | Riesgo | Probabilidad | Impacto | Prioridad | Tratamiento |
|---|---|---|---|---|---|
| 15 | No se soporten ciertas características de movilidad del protocolo ya que esta última no se encuentra contemplada dentro del alcance del proyecto | Probable | Menor | Medio | Desarrollo del protocolo ligero para que en un futuro, el proyecto de movilidad tenga facilidad de transportar los archivos de operación entre nodos |

# B. Anexo: MIB UNAL

A continuación se muestra la estructura completa de la MIB base, la cual debe ser usada por todos los proyectos que en adelante requieran la asignación de un identificador de objeto que permita describir cierto componente o servicio

```
UNAL-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY
        FROM SNMPv2-SMI
    ;

unalMib MODULE-IDENTITY
    LAST-UPDATED  "201805141203Z" -- May 14, 2018
    ORGANIZATION
        "Universidad Nacional de Colombia"
    CONTACT-INFO
        "Mauricio Tamayo
         Universidad Nacional de Colombia
         Av. Carrera 30 No. 45-03 Edificio 608
         Bogota, 111321
         Colombia
         Tel: (571) 3165000
         E-Mail: emtamayog@unal.edu.co
         Website: www.unal.edu.co"
    DESCRIPTION
        "Top-level hierarchy of the SNMP MIB tree for the UNAL
         project"
    REVISION      "201702281045Z" -- February 28, 2017
    DESCRIPTION
        "First draft"
    ::= { 1 3 6 1 4 1 49843 }

-- managed objects divided for university headquarters

experimental OBJECT IDENTIFIER ::= { unalMib 0 }
bogota OBJECT IDENTIFIER ::= { unalMib 1 }
medellin OBJECT IDENTIFIER ::= { unalMib 2 }
manizales OBJECT IDENTIFIER ::= { unalMib 3 }
```

```
palmira OBJECT IDENTIFIER ::= { unalMib 4 }
caribe OBJECT IDENTIFIER ::= { unalMib 5 }
amazonia OBJECT IDENTIFIER ::= { unalMib 6 }
tumaco OBJECT IDENTIFIER ::= { unalMib 7 }
orinoquia OBJECT IDENTIFIER ::= { unalMib 8 }

-- Notifications
--
-- Please define notifications objects on each MIB project from each
-- headquarter using the next hierarchy:
-- unalMib headquarterID projectName projectNotification (with id 0)
--
-- projectNameNotificationPrefix OBJECT IDENTIFIER ::= {projectName 0}
-- projectNameNotifications  OBJECT IDENTIFIER ::=
-- {projectNameNotificationPrefix 0}
-- projectNameNotificationObjects OBJECT IDENTIFIER ::=
-- {projectNameNotificationPrefix 1}
--
-- Conformance
--
-- Please define notifications objects on each MIB project from each
-- headquarter using the next hierarchy:
-- unalMib headquarterID projectName projectConformance (with id 2)
--
-- projectNameConformance OBJECT IDENTIFIER ::= {projectName 2}
-- projectNameCompliances OBJECT IDENTIFIER ::= {projectNameConformance 1}
-- projectNameGroups  OBJECT IDENTIFIER ::= {projectNameConformance 2}

END
```

# C. Anexo: MIB TLON

A continuación se muestra la estructura completa de la MIB diseñada para la gestión de dispositivos con limitaciones de gestión. En el aparecen los objectos para equipos heredados y pequeños sensores

```
-- MIB objects for agent modules on devices with management protocol
-- limitations as ad-hoc sensors

TLON-MIB DEFINITIONS ::= BEGIN

IMPORTS
    NOTIFICATION-GROUP, OBJECT-GROUP
        FROM SNMPv2-CONF
    Integer32, IpAddress, MODULE-IDENTITY, NOTIFICATION-TYPE,
    OBJECT-IDENTITY, OBJECT-TYPE
        FROM SNMPv2-SMI
    RowStatus, TEXTUAL-CONVENTION
        FROM SNMPv2-TC
    bogota
        FROM UNAL-MIB
    ;

tlonMib MODULE-IDENTITY
    LAST-UPDATED  "201709250426Z" -- September 25, 2017
    ORGANIZATION  "Grupo de Investigacion en Redes de Telecomunicaciones
                   Dinamicas y Lenguajes de Programacion Distribuidos - TLON
                   Ingenieria de Sistemas y Computacion - Sede Bogota
                   http://tlon.unal.edu.co/"
    CONTACT-INFO
                   "Mauricio Tamayo
                   Universidad Nacional de Colombia
                   Av. Carrera 30 No. 45-03 Edificio 608
                   Bogota, 111321
                   Colombia
                   Tel: (571) 3165000
                   E-Mail: emtamayog@unal.edu.co
                   Website: www.unal.edu.co"
    DESCRIPTION   "This MIB contains objects of legacy devices and ad hoc
                   sensors with processing and resources restrictions. It
                   serves to the protocol conversion SNMP to serial, which
```

```
                project is named JIS."
    ::= { bogota 1 }

Float2d ::= TEXTUAL-CONVENTION
    DISPLAY-HINT  "d-2"
    STATUS        current
    DESCRIPTION   "This type represents a
                   floating-point number with two decimals."
    SYNTAX        Integer32

Float32 ::= TEXTUAL-CONVENTION
    STATUS        current
    DESCRIPTION   "This type represents a 32-bit (4-octet) IEEE floating-point
                   number in binary interchange format."
    REFERENCE     "IEEE Standard for Floating-Point Arithmetic, Standard
                   754-2008"
    SYNTAX        OCTET STRING

tlonObjects OBJECT IDENTIFIER ::= { tlonMib 1 }

tlonNotification OBJECT IDENTIFIER ::= { tlonObjects 0 }

alterTable NOTIFICATION-TYPE
    OBJECTS
        { anyUpdate }
    STATUS        current
    DESCRIPTION
        "An alterTable trap signifies that the SNMP table has been
         altered and it must be notified to a trap server."
    ::= { tlonNotification 1 }

anyUpdate OBJECT-TYPE
    SYNTAX        OCTET STRING
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Object to report any change in all tables of the MIB"
    ::= { tlonNotification 2 }

tlonLegacy OBJECT IDENTIFIER ::= { tlonObjects 1 }

cdmTable OBJECT-TYPE
    SYNTAX        SEQUENCE OF CdmEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION   "A list of CDM-600 devices."
    ::= { tlonLegacy 1 }
```

```
cdmEntry OBJECT-TYPE
    SYNTAX          CdmEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "The CDM-600 is a modem Comtech Ef Data. It operates on the
                    IF-Band frequency (70/140 MHz). More info:
                    https://www.comtechefdata.com/files/manuals/mn-modems-pdf/
                    mn-cdm600-600L.pdf"
    INDEX           { cdmId }
    ::= { cdmTable 1 }

CdmEntry ::= SEQUENCE
{
    cdmId             Integer32,
    cdmAddress        OCTET STRING,
    cdmTxFreq         Integer32,
    cdmTxMod          Integer32,
    cdmTxFec          Integer32,
    cdmTxPower        Float32,
    cdmTemp           Integer32,
    cdmTableStatus    RowStatus
}

cdmId OBJECT-TYPE
    SYNTAX          Integer32
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "Up to 9,999 devices can be uniquely addressed. In
                    RS-232 applications this value is set to 0. In RS-485
                    applications, the permissible range of values is 1 to
                    9999. It is programmed into a Target unit using the
                    front panel keypad. The format has four digits '0000'"
    ::= { cdmEntry 1 }

cdmAddress OBJECT-TYPE
    SYNTAX          OCTET STRING
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "TTY port (serial) where the device is connected"
    ::= { cdmEntry 2 }

cdmTxFreq OBJECT-TYPE
    SYNTAX          Integer32
    UNITS           "Hz"
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "With hardware revision xx.0, standard frequency
                    range: 52 MHz to 88 MHz, or 104 MHz to 176 MHz.
```

```
                    With hardware revision xx.1, extended frequency
                    range: 50 MHz to 90 MHz, or 100 MHz to 180 MHz.
                    (Use HRV? to obtain the hardware revision level)
                    Resolution=100Hz. Example: TFQ=072.9872
                    Note: The CDM-600 supports 70 and 140 MHz bands."
    ::= { cdmEntry 3 }

cdmTxMod OBJECT-TYPE
    SYNTAX          Integer32
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "Tx Modulation type, where:
                    0=BPSK
                    1=QPSK
                    2=OQPSK
                    3=8PSK
                    4=16QAM (Turbo or Viterbi+RS only)
                    5=8-QAM (LDPC only)
                    (Requires TPC/LDPC codec and FAST option) Depending
                    on FEC type, not all of these selections will be
                    valid. Example: TMD=2 (OQPSK)"
    ::= { cdmEntry 4 }

cdmTxFec OBJECT-TYPE
    SYNTAX          Integer32
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "Tx FEC Code Rate, where:
                    0 = Rate 1/2
                    1 = Rate 3/4
                    2 = Rate 7/8
                    3 = Rate 2/3 (8-PSK TCM or LDPC only)
                    4 = Rate 1/1 (Uncoded or No FEC)
                    5 = Rate 21/44 (Turbo Only)
                    6 = Rate 5/16 (Turbo Only)
                    7 = Rate 0.95 (Turbo Only) Depending on FEC type
                    of these selections will be valid.
                    Example: TCR=1 (Rate 3/4)"
    ::= { cdmEntry 5 }

cdmTxPower OBJECT-TYPE
    SYNTAX          Float32
    UNITS           "dBm"
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "Tx Output power level between 0 and -20 dBm
                    (minus sign assumed). Example: TPL=13.4"
    ::= { cdmEntry 6 }
```

```
cdmTemp OBJECT-TYPE
    SYNTAX          Integer32
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "Unit returns the value of the internal temperature, in
                    the form xxx (degrees C). Example: TMP=+26"
    ::= { cdmEntry 7 }

cdmTableStatus OBJECT-TYPE
    SYNTAX          RowStatus
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "To create or drop an entire row in one shot (RFC 2579)"
    ::= { cdmEntry 8 }

tlonAdhoc OBJECT IDENTIFIER ::= { tlonObjects 2 }

bmpTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF BmpEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "A list of BMP180 sensors"
    ::= { tlonAdhoc 1 }

bmpEntry OBJECT-TYPE
    SYNTAX          BmpEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "The BMP180 is a basic sensor that is designed
                    specifically for  measuring barometric pressure,
                    it also does temperature  measurement on the side
                    to help."
    INDEX           { bmpId }
    ::= { bmpTable 1 }

BmpEntry ::= SEQUENCE
{
    bmpId           Integer32,
    bmpAddress      IpAddress,
    bmpTemp         Integer32,
    bmpPres         Integer32,
    bmpPres2d       Float2d,
    bmpPres32       Float32,
    bmpTableStatus  RowStatus
}
```

```
bmpId OBJECT-TYPE
    SYNTAX          Integer32 (0..32000)
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "A number to identify each sensor independently"
    ::= { bmpEntry 1 }

bmpAddress OBJECT-TYPE
    SYNTAX          IpAddress
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "IP Address of the device"
    ::= { bmpEntry 2 }

bmpTemp OBJECT-TYPE
    SYNTAX          Integer32
    UNITS           "deg C"
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "Temperature sensing, -40 to + 85C operational range,
                    +-2 C temperature accuracy. The manager should divide
                    the data by 100, to obtain the real value in decimal"
    ::= { bmpEntry 3 }

bmpPres OBJECT-TYPE
    SYNTAX          Integer32
    UNITS           "hPa"
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "Pressure sensing range: 300-1100 hPa (9000m to -500m
                    above sea level). The manager should divide the data
                    by 100, to  obtain the real value in decimal."
    ::= { bmpEntry 4 }

bmpPres2d OBJECT-TYPE
    SYNTAX          Float2d
    UNITS           "hPa"
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "The same bmpPres object but with another kind of type
                    to demonstrate the use of TEXTUAL-CONVENTION base on
                    integer32"
    ::= { bmpEntry 5 }

bmpPres32 OBJECT-TYPE
    SYNTAX          Float32
    UNITS           "hPa"
    MAX-ACCESS      read-create
```

```
    STATUS          current
    DESCRIPTION     "The same bmpPres object but with another kind of type
                    to demonstrate the use of TEXTUAL-CONVENTION base on
                    Octect String"
    ::= { bmpEntry 6 }

bmpTableStatus OBJECT-TYPE
    SYNTAX          RowStatus
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION     "To create or drop an entire row in one shot (RFC 2579)"
    ::= { bmpEntry 7 }

hcsrTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF HcsrEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "A list of HC-SR04 sensors"
    ::= { tlonAdhoc 2 }

hcsrEntry OBJECT-TYPE
    SYNTAX          HcsrEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "Ultrasonic ranging module HC-SR04 provides non-contact
                    measurement function. The modules includes ultrasonic
                    transmitters, receiver and control circuit. The basic
                    principle of work:(1) Using IO trigger for at least 10
                    us high level signal, (2) The Module automatically sends
                    eight 40 kHz and detect whether there is a pulse signal
                    back, and (3) IF the signal back, through high level,
                    time of high output IO duration is the time from sending
                    ultrasonic to returning"
    INDEX           { hcsrId }
    ::= { hcsrTable 1 }

HcsrEntry ::= SEQUENCE
{
    hcsrId            Integer32,
    hcsrAddress       OCTET STRING,
    hcsrDistance      Integer32,
    hcsrTableStatus   RowStatus
}

hcsrId OBJECT-TYPE
    SYNTAX          Integer32 (0..32000)
    MAX-ACCESS      read-create
    STATUS          current
```

```
        DESCRIPTION    "A number to identify the sensor"
        ::= { hcsrEntry 1 }

    hcsrAddress OBJECT-TYPE
        SYNTAX         OCTET STRING (SIZE(4))
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION    "TTY port (serial) where the device is connected"
        ::= { hcsrEntry 2 }

    hcsrDistance OBJECT-TYPE
        SYNTAX         Integer32
        UNITS          "cm"
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION    "provides 2cm - 400cm non-contact measurement function,
                       the ranging accuracy can reach to 3mm"
        ::= { hcsrEntry 3 }

    hcsrTableStatus OBJECT-TYPE
        SYNTAX         RowStatus
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION    "To create or drop an entire row in one shot (RFC2579)"
        ::= { hcsrEntry 4 }

    tlonConformance OBJECT IDENTIFIER ::= { tlonMib 2 }

    tlonNotifications OBJECT IDENTIFIER ::= { tlonConformance 0 }

    notificationsGroup NOTIFICATION-GROUP
        NOTIFICATIONS { alterTable }
        STATUS         current
        DESCRIPTION    "Some notifications implemented over the SNMP agent
                       supporting command responder applications."
        ::= { tlonNotifications 1 }

    tlonCompliances OBJECT IDENTIFIER ::= { tlonConformance 1 }

    tlonGroups OBJECT IDENTIFIER ::= { tlonConformance 2 }

    indexTableGroup OBJECT-GROUP
        OBJECTS        { bmpId, hcsrId, cdmId }
        STATUS         current
        DESCRIPTION    "This group contains the index row of each table"
        ::= { tlonGroups 0 }

    identifiersGroup OBJECT-GROUP
```

```
    OBJECTS        { bmpAddress, hcsrAddress, cdmAddress }
    STATUS         current
    DESCRIPTION    "List of identifiers (interface type, ip address,
                   mac address and others) of each object"
    ::= { tlonGroups 1 }

legacyObjectGroup OBJECT-GROUP
    OBJECTS        { cdmTemp, cdmTxFec, cdmTxFreq, cdmTxMod, cdmTxPower }
    STATUS         current
    DESCRIPTION    "This group contains the objects of legacy devices"
    ::= { tlonGroups 2 }

sensorObjectGroup OBJECT-GROUP
    OBJECTS        { bmpPres, bmpPres2d, bmpPres32, bmpTemp, hcsrDistance }
    STATUS         current
    DESCRIPTION    "This group contains the objects of ad hoc devices"
    ::= { tlonGroups 3 }

tableStatusGroup OBJECT-GROUP
    OBJECTS        { bmpTableStatus, hcsrTableStatus, cdmTableStatus }
    STATUS         current
    DESCRIPTION    "This group contains the table status objects"
    ::= { tlonGroups 4 }

END
```

# D. Anexo: Código Fuente

## D.1. jisAgent

```
from pysnmp.entity import engine, config
from pysnmp.entity.rfc3413 import cmdrsp, context, ntforg
from pysnmp.proto import rfc1902
from pysnmp.carrier.asyncore.dgram import udp
from pysnmp.proto.api import v2c
from pysnmp.smi import builder
import threading
import collections
import time
import subprocess
import os

MibObject = collections.namedtuple('MibObject', ['mibName',
                                    'objectType', 'valueFunc'])

ipAddress = subprocess.check_output(
            "hostname -I", shell=True).decode('utf-8')

class Mib(object):
    #Initializes the first row of each table to keep consistency

    def tableId(self):
        return (1)

    def tableIdStr(self):
        return ('0000')

    def address(self):
        a = rfc1902.IpAddress('127.0.0.1')
        return a

    def ttyAddress(self):
        return ('tty0')

    def objectData(self):
        return (0)
```

```
        def objectData2d(self):
            return (300.0)

        def objectData32(self):
            return (b'300.00')

        def tableStatus(self):
            return ('createAndGo')

    def createVariable(SuperClass, getValue, *args):
        """This is going to create an instance variable that we can export.
        getValue is a function to call to retreive the value of the scalar
        """
        class Var(SuperClass):
            def readGet(self, name, *args):
                return name, self.syntax.clone(getValue())
        return Var(*args)


    class SNMPAgent(object):
        """Implements an Agent that serves the custom MIB and
        can send a trap.
        """

        def __init__(self, mibObjects):
            """
            mibObjects - a list of MibObject tuples that this agent
            will serve
            """

            # Create SNMP engine
            self._snmpEngine = engine.SnmpEngine()

            # Transport setup
            #open a UDP socket to listen for snmp requests over IPv4
            config.addTransport(self._snmpEngine, udp.domainName,
                                udp.UdpTransport().openServerMode((ipAddress,


            # SNMPv2c setup
            # SecurityName <-> CommunityName mapping.
            # add a v2 user with the community string public
            config.addV1System(self._snmpEngine, "agent", "public")

            # Allow read MIB access for this user / securityModels at VACM
            # let anyone accessing 'public' read anything in the subtree below,
            # which is the enterprises subtree that we defined our MIB to be in
            config.addVacmUser(self._snmpEngine, 2, "agent", "noAuthNoPriv",
```

```
                                                    (1,3,6,1,4,1), (1,3,6,1,4,1))

        # Create an SNMP context, each app has one or more contexts
        self._snmpContext = context.SnmpContext(self._snmpEngine)

        #the builder is used to load mibs. tell it to look in the
        #current directory for our new MIB. We'll also use it to
        #export our symbols later
        mibBuilder = self._snmpContext.getMibInstrum().getMibBuilder()
        mibSources = mibBuilder.getMibSources() +
                     (builder.DirMibSource('.'),)
        mibBuilder.setMibSources(*mibSources)

        #variables will subclass this since we only have scalar types
        #can't load this type directly, need to import it
        MibScalarInstance, = mibBuilder.importSymbols('SNMPv2-SMI',
                                                      'MibScalarInstance')

        #export our custom mib
        for mibObject in mibObjects:
            nextVar, = mibBuilder.importSymbols(mibObject.mibName,
                                                mibObject.objectType)
            instance = createVariable(MibScalarInstance,
                                      mibObject.valueFunc,
                                      nextVar.name, (0,),
                                      nextVar.syntax)
            #need to export as <var name>Instance
            instanceDict = {str(nextVar.name)+"Instance":instance}
            mibBuilder.exportSymbols(mibObject.mibName,
                                     **instanceDict)

        # tell pysnmp to respotd to get, set, getnext, and getbulk

        cmdrsp.GetCommandResponder(self._snmpEngine, self._snmpContext)
        cmdrsp.SetCommandResponder(self._snmpEngine, self._snmpContext)
        cmdrsp.NextCommandResponder(self._snmpEngine, self._snmpContext)
        cmdrsp.BulkCommandResponder(self._snmpEngine, self._snmpContext)

    def dispatcher(self):
        print ("Starting Agent")
        self._snmpEngine.transportDispatcher.jobStarted(1)
        try:
            self._snmpEngine.transportDispatcher.runDispatcher()
        except:
            self._snmpEngine.transportDispatcher.closeDispatcher()
            print ("\nClose Dispatcher")
            raise
```

```
if __name__ == '__main__':

    mib = Mib()
    objects = [MibObject('TLON-MIB', 'bmpId', mib.tableId),
               MibObject('TLON-MIB', 'bmpAddress', mib.address),
               MibObject('TLON-MIB', 'bmpTemp', mib.objectData),
               MibObject('TLON-MIB', 'bmpPres', mib.objectData),
               MibObject('TLON-MIB', 'bmpPres2d', mib.objectData2d),
               MibObject('TLON-MIB', 'bmpPres32', mib.objectData32),
               MibObject('TLON-MIB', 'bmpTableStatus', mib.tableStatus),
               MibObject('TLON-MIB', 'hcsrId', mib.tableId),
               MibObject('TLON-MIB', 'hcsrAddress', mib.ttyAddress),
               MibObject('TLON-MIB', 'hcsrDistance', mib.objectData),
               MibObject('TLON-MIB', 'hcsrTableStatus', mib.tableStatus),
               MibObject('TLON-MIB', 'cdmId', mib.tableIdStr),
               MibObject('TLON-MIB', 'cdmAddress', mib.ttyAddress),
               MibObject('TLON-MIB', 'cdmTxFreq', mib.objectData),
               MibObject('TLON-MIB', 'cdmTxMod', mib.objectData),
               MibObject('TLON-MIB', 'cdmTxFec', mib.objectData),
               MibObject('TLON-MIB', 'cdmTxPower', mib.objectData32),
               MibObject('TLON-MIB', 'cdmTemp', mib.objectData),
               MibObject('TLON-MIB', 'cdmTableStatus', mib.tableStatus)
              ]

    agent = SNMPAgent(objects)

    try:
        agent.dispatcher()
    except KeyboardInterrupt:
        print ("\nShutting down")
```

## D.2. modifyRow

```
"""
This function modifies a row into a dynamic SNMP table, assigning default
values (in each object to keep its consistency.) if a new device has gone
into the network or deleting a row if the device has gone out
"""

from pysnmp.hlapi import *
import subprocess

# Define the SNMP server's parameters
server = subprocess.check_output("hostname -I", shell=True).decode('utf-8')
port = 161
community = 'public'
```

```
mibName = 'TLON-MIB'

# Define the list of objects congifured in the MIB
mibObjects = {'bmpId': ['bmpAddress', 'bmpTemp', 'bmpPres', 'bmpPres2d',
               'bmpPres32', 'bmpTableStatus'],
               'hcsrId': ['hcsrAddress', 'hcsrDistance', 'hcsrTableStatus'],
               'cdmId':['cdmAddress','cdmTxFreq','cdmTxMod', 'cdmTxFec',
               'cdmTxPower', 'cdmTemp', 'cdmTableStatus']
              }

def assignedRows (mibObjectIndex):
    # Calculates how many instances there are in a table and returns
    # the index of each one
    rowInstanceId = 1
    statusTableIndex = []
    print('Checking the assigned rows in the TLON-MIB...')
    while rowInstanceId < 32:
    # limit of sensor of each table, it can be increased or decreased
        errorIndication, errorStatus, errorIndex, varBinds = next(
            getCmd(SnmpEngine(),
                   CommunityData(community),
                   UdpTransportTarget((server, port)),
                   ContextData(),
                   ObjectType(ObjectIdentity(mibName, mibObjectIndex,
                                             rowInstanceId)))
        )
        if errorIndication:
            print(errorIndication)
        elif errorStatus:
            print('%s at %s' %(errorStatus.prettyPrint(),
                               errorIndex and
                               varBinds[int(errorIndex) - 1][0] or '?'))
        else:
            for varBind in varBinds:
                x = varBind[1]
                if x :
                    statusTableIndex.append(rowInstanceId)
        rowInstanceId += 1
    return statusTableIndex

def sendingNotification (trapMessage):
    errorIndication, errorStatus, errorIndex, varBinds = next(
        sendNotification(
            SnmpEngine(),
            CommunityData('public'),
            UdpTransportTarget(('192.168.184.1', 162)),
            ContextData(),
            'trap',
```

```
                NotificationType(
                    ObjectIdentity(mibName, 'alterTable'),
                                objects={('TLON-MIB', 'anyUpdate'):
                                        trapMessage,
                                    }
                                )
                )
            )
        if errorIndication:
            print(errorIndication)

    def setData (objectName, value, rowInstanceId):
        # Receives the data and sets it into the object in the MIB table
        print ('Updating data into the SNMP table')
        g = setCmd(SnmpEngine(),
                   CommunityData(community),
                   UdpTransportTarget((server, port)),
                   ContextData(),
                   ObjectType(ObjectIdentity(mibName, objectName,
                                             rowInstanceId), value)
                  )
        next(g)

    def getData (mibObjectIndex, rowInstanceId):
    # It verifies the information into the MIB table and returns the value
    # from certain object device
        data = 0
        errorIndication, errorStatus, errorIndex, varBinds = next(
            getCmd(SnmpEngine(),
                   CommunityData(community),
                   UdpTransportTarget((server, port)),
                   ContextData(),
                   ObjectType(ObjectIdentity(mibName, mibObjectIndex,
                                             rowInstanceId)))
        )
        if errorIndication:
            print(errorIndication)
        elif errorStatus:
            print('%s at %s' %(errorStatus.prettyPrint(),
                               errorIndex and
                               varBinds[int(errorIndex) - 1][0] or '?'))
        else:
            for varBind in varBinds:
                x = varBind[1]
                if x :
                    data = x.prettyPrint()
        return data
```

```
class Create:
#This class creates a new device in the dynamic table SNMP

    def newSensorId (self, indexTable):
    # Search for the assigned IDs and returns a new ID
        instancesCreated = assignedRows(indexTable)
        value = 1
        if instancesCreated != []:
            for instance in range (0, len(instancesCreated)):
                if instance+1 != instancesCreated[instance]:
                    value = instance+1
                    break
                else:
                    value = instance+2
        return value

    def valuesList (self, indexTable, sensorId, addressId):
    # Puts the default values into a list according to the quantity of
    # sensor's objects. Define the list of values that must be configured
    # as default in the MIB

        defaultValues = {'bmpId': [0,300,300.00,b'300.00','createAndGo'],
                         'hcsrId': [0,'createAndGo'],
                         'cdmId':[0,0,0,'0.00',0,'createAndGo']
                        }
        values = [sensorId, addressId]
        for sensorType in defaultValues:
            if sensorType == indexTable:
                for sensorValue in defaultValues[sensorType]:
                    values.append(sensorValue)
                break
        return values

    def objectsList (self, indexTable):
    # Puts into a list each object from a kind of sensor and returns it
        objects = [indexTable]
        for indexRow in mibObjects[indexTable]:
            objects.append(indexRow)
        return objects

    def create (self, indexTable, addressId):
    # It verifies the actual data into SNMP table and use it to know
    # the new necessary information to create the row
        print ('--------------------')
        print ('Creating a new row in the table for the sensor', indexTable)
        sensorId = self.newSensorId(indexTable)
```

```
            objects = self.objectsList(indexTable)
            values = self.valuesList(indexTable, sensorId, addressId)

            for iteration in range (0, len(objects)):
                setData(objects[iteration],
                        values[iteration],
                        sensorId
                       )
            trapMessage = 'A new sensor was created'
            sendingNotification(trapMessage)
            print ('The sensor ', indexTable, 'was created with ID', sensorId)

    class Destroy:
    #This class deletes a device from the dynamic table SNMP
        # Define the list of devices created in the MIB
        mibObjectList = ['bmp', 'hcsr', 'cdm']

        def blockTable(self, reg):
        # Uses a file to put 0 (unblock) or 1 (block) to indicate if the
        # system must avoid to write in the SNMP table when an object is
        # being delete
            register = open('/home/pi/code/register.txt', 'w')
            register.write(reg)
            register.close()

        def destroy (self, addressToDelete):
        # Receives the device's address, check its instance and delete it
            print('--------------------')
            print ('The device with address', addressToDelete,
                   'has gone out of the network.')
            print ('It will be remove from the dinamyc SNMP table...')
            assignedRowsList = []

            for indexTable in self.mibObjectList:
            # Search in all tables (indexTable) created in the mib
                # Verify the rows assigned in each table

                assignedRowsList = assignedRows(indexTable+'Id')
                if len(assignedRowsList) > 0:
                    for indexRow in assignedRowsList:
                        addressAsigned = getData(indexTable+'Address',
                                                 indexRow)
                        if addressAsigned == addressToDelete:
                            self.blockTable('1')
                            setData(indexTable+'TableStatus', 6,
                                    indexRow)
                            self.blockTable('0')
                            trapMessage = 'A sensor was deleted'
```

```
                    sendingNotification(trapMessage)
                    print('The row of the sensor', indexTable,
                          'with index', indexRow, 'was deleted')
                    break
```

## D.3. updateData

```
"""
This code verifies the devices created on the SNMP table and requests
the objects's data for each one and update the table.
"""

from pysnmp.hlapi import *
from dataRequest import *
from modifyRow import *
import time

# Define the list of modificable objects congifured in the MIB

mibObjects = {'bmp': ['bmpTemp','bmpPres','bmpPres2d','bmpPres32'],
              'hcsr': ['hcsrDistance'],
              'cdm':['cdmTxFreq','cdmTxMod','cdmTxFec','cdmTxPower',
                     'cdmTemp']
             }

assignedRowsList = []

def readValue (device, mibObjectData, cycles):
# Puts the data in all columns of each object
    data = Request()      # Request is a class from dataRequest
    for rowInstanceId in cycles:
        # Checks if the SNMP table is blocked by 'Destroy' function
        register = open('/home/pi/code/register.txt', 'r')
        validation = register.read()
        register.close()
        if validation == '0':
            # Applies the data on the ObjectType indicated
            address = getData(device+'Address', rowInstanceId)
            setData(mibObjectData,
                    data.sensorData(mibObjectData,address,['read',0]),
                    rowInstanceId)
        else:
            print('The SNMP database is busy, '
                  'the update process will restart')
            break

while True:
#  Runs on an infinity cycle
```

```
        print("----------")
        for indexTable in mibObjects:
        # Search in all tables (indexTable) created in the mib
            # Wait 1 seconds to execute the process in each interval
            time.sleep(1)
            # Verify the rows assigned in each table
            assignedRowsList = assignedRows(indexTable+'Id')
            if len(assignedRowsList) > 0:
                for indexRow in mibObjects[indexTable]:
                # Puts the data in each column of each row of the table
                    readValue(indexTable, indexRow, assignedRowsList)
            else:
                print('There is no rows to update '
                      'in the table of', indexTable)
```

## D.4. dataRequest

```
"""
This code requests the data to certain device that could be connected
via wifi or serial. If it is connected to the wifi network, this code
creates a socket to request data to the sensor but if it is connected
to some TTY interface, this code open the port to request the data
"""

import socket
import serial
import time

class Request:

    def sensorData (self, mibObjectData, address, dataReq):
        method = getattr(self, mibObjectData,
                         lambda: "Invalid MIB Object Data")
        return method (address, dataReq)

    def extractDataReq(self, dataReq):
        if dataReq[0]=='write':
            return '='+dataReq[1]
        else:
            return '?'

    def bmp (self, address, message):
        data = ""
        port = 23050

        # Creates a socket, sends the message and waits until the total
        # answer is received, next the connection is closed
        try:
```

```
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sent = sock.sendto(message, (address, port))
            data, server = sock.recvfrom(1024)
            sock.close()
            print (type(data), data)
            if data == b"error 100":
                data = self.bmp(address, message)
        except:
            print('It was not able to stablish communication'
                  'with the peer\n')
            data = '0'
        return data

    def bmpTemp (self, address, dataReq):
        message = b'temperature' # keyword to send in the message
        data = self.bmp(address, message)
        return float(data)

    def bmpPres (self, address, dataReq):
        message = b'pressure' # keyword to send in the message
        data = self.bmp(address, message)
        return float(data)

    def bmpPres2d (self, address, dataReq):
        message = b'pressure' # keyword to send in the message
        data = self.bmp(address, message)
        return data

    def bmpPres32 (self, address, dataReq):
        message = b'pressure' # keyword to send in the message
        data = self.bmp(address, message)
        return str(data)

    def bmpAddress (self, address, dataReq):
        print ('The device was update')

    def hcsrDistance (self, address, dataReq):
        txd = serial.Serial(address, baudrate=9600, timeout=1)
        rxd = txd.readline(20).decode('ascii')

        """
        The data shown in the console appears like: Distance = 42.55 cm
        The next code manipulates the line captured, but it verifies if
        the data is useful and takes only the wanted value. If the data
        has errors, it tries to request it again until 10 times before
        close the program.
        """
```

```
        if rxd.find('cm') > 0:
            rxd = rxd[rxd.find('=')+2:rxd.find('cm')]
        else:
            for reply in range(10):
                rxd = txd.readline(20).decode('ascii')
                if rxd.find('cm') > 0:
                    rxd = rxd[rxd.find('=')+2:rxd.find('cm')]
                    break
        try:
            rxd = float(rxd)*100
            rxd = int(rxd)
        except ValueError:
            print('The captured information is not useful '
                  'or presents errors\n')
            rxd = 0
        return rxd

    def loading (self, address, message):
        txd = serial.Serial(address, baudrate=9600, timeout=1)
        txd.write(message.encode())
        # Convierte en Bytes para transmitir con formato UTF-8 (defecto)
        rxd = txd.readline().decode()
        txd.close()
        # Convierte en UTF-8 el valor recibido
        # otras opciones como 'ascii' entre parentesis
        return rxd
        # >0000/TFT=7

    def cdmTxFreq (self, address, dataReq):
        extract=self.extractDataReq(dataReq)
        message = '<0000/TFQ'+extract+'\x0D'
        data = self.loading(address, message)
        data = data[data.find('=')+1:data.find('.')]+
                                data[data.find('.')+1:]
        return int(data)

    def cdmTxMod (self, address, dataReq):
        extract=self.extractDataReq(dataReq)
        message = '<0000/TMD'+extract+'\x0D'
        data = self.loading(address, message)
        data = data[data.find('=')+1:]
        return int(data)

    def cdmTxFec (self, address, dataReq):
        extract=self.extractDataReq(dataReq)
        message = '<0000/TFT'+extract+'\x0D'
        data = self.loading(address, message)
        data = data[data.find('=')+1:]
```

```
        return int(data)

    def cdmTxPower (self, address, dataReq):
        extract=self.extractDataReq(dataReq)
        message = '<0000/TPL?'+extract+'x0D'
        data = self.loading(address, message)
        data = data[data.find('=')+1:]
        return str(data)

    def cdmTemp (self, address, dataReq):
        extract=self.extractDataReq(dataReq)
        message = '<0000/TMP'+extract+'\x0D'
        data = self.loading(address, message)
        data = data[data.find('=')+1:]
        return int(data)
```

## D.5. monitor

```
"""
This function monitors the server's hardware for changes in
the active TTY interfaces. If a new device is detected, it
creates a new row in the SNMP dinamyc table or destroy the
row if a device is removed
"""

import pyudev
from pyudev import Context, Monitor, MonitorObserver
from modifyRow import *

context = pyudev.Context()
monitor = Monitor.from_netlink(context)
monitor.filter_by(subsystem='tty')

for action, device in monitor:
    if action == 'add':
        print (action, device.get('DEVNAME'))
        newDevice = Create()
        newDevice.create('hcsrId', device.get('DEVNAME'))
    else:
        print (action, device.get('DEVNAME'))
        newDevice = Destroy()
        newDevice.destroy(device.get('DEVNAME'))
```

# E. Anexo: Sensor BMP180

A continuación se detallan las configuraciones del sensor BMP180 conectado a una Raspberry Pi Zero W.

## E.1. Configuración I2C

Se deben seguir los siguientes pasos[1] para activar el bus de comunicaciones necesario para la operación del sensor con la Raspberry Pi Zero W

- Modificar el archivo */etc/modules* agregando al final:
  i2c-bcm2708
  i2c-dev
- Se requiere la activación del bus I2C en el sistema operativo Raspian. Para ello se da el comando *sudo raspi-config*, se ubica en periféricos la opción para activar el I2C bus.
- Reiniciar la Raspberry
- Instalar las siguientes herramientas con el comando *sudo apt-get install python-smbus i2c-tools git*
- Ejecutar comando de validación *i2cdetect -y 1*

## E.2. Código Servidor UDP

```
#https://gist.github.com/Manouchehri/67b53ecdc767919dddf3ec4ea8098b20"""
import socket
from Adafruit_BMP085 import BMP085

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
address = "10.0.0.1"
port = 23050
server = (address, port)
sock.bind (server)
print("Listening in " + address + ":" + str(port))
```

[1] Mayor información https://tutorials-raspberrypi.com/raspberry-pi-and-i2c-air-pressure-sensor-bmp180/

```
def bmpMeasure (request):
    bmp = BMP085(0x77)
    if request == "temperature":
        value = bmp.readTemperature()
    elif request == "pressure":
        value = bmp.readPressure()
    else:
        value 0 "error 100"
    value = str(value)
    return value

while True:
    request, client = sock.recvfrom(1024)
    request = request.decode()
    response = bmpMeasure(request)
    response = response.encode()
    print("Echoing data back to " + str(client))
    sent = sock.sendto(response, client)
```

# F. Anexo: Pruebas de aceptación

## F.1. Mensajes primitivos de servicio

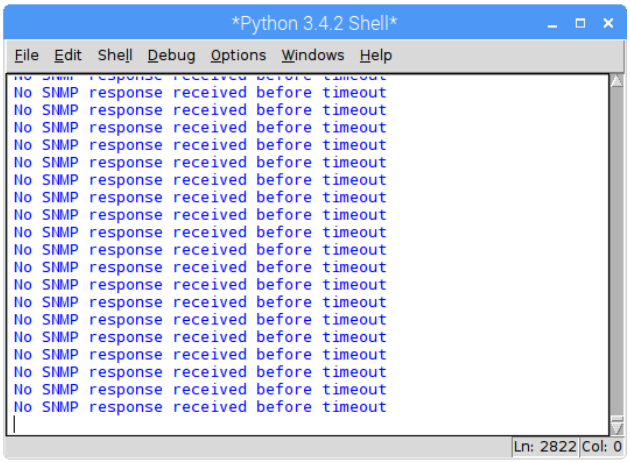


**Figura F-1**.: Mensajes de error ante agente SNMP no disponible

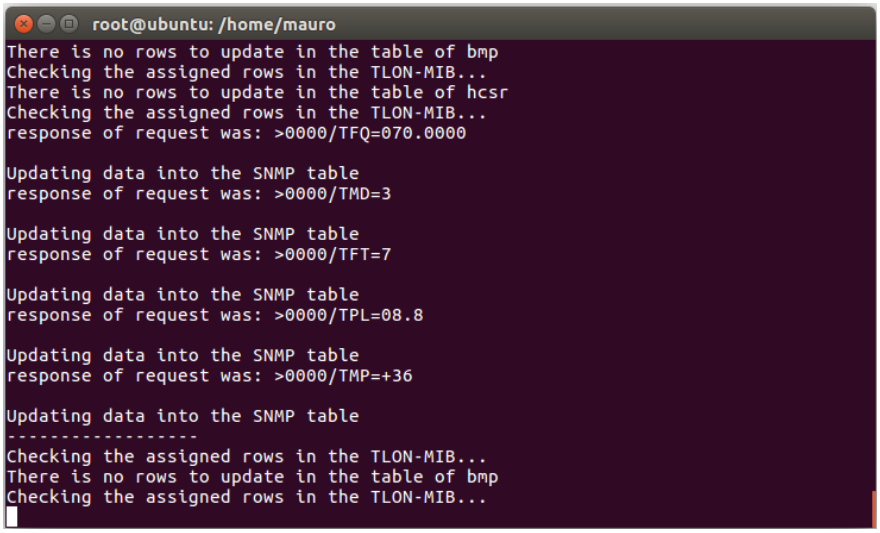


**Figura F-2**.: Mensajes de solicitud y respuesta en conexión serial

## F.2. Mensajes de notificación

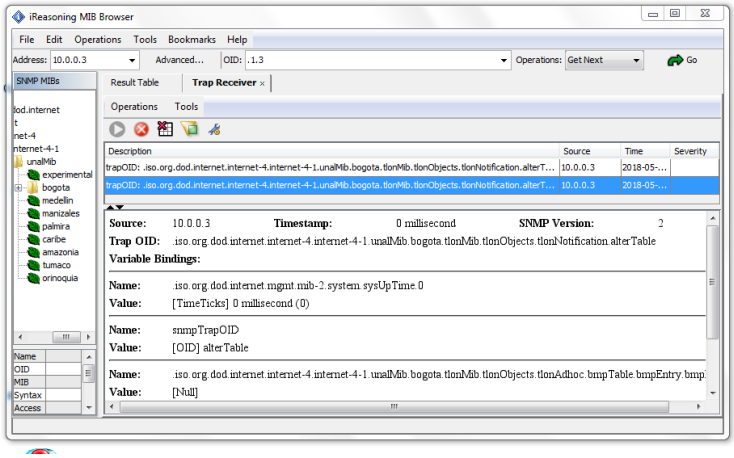


**Figura F-3**.: Notificación SNMP informando cambios en tabla

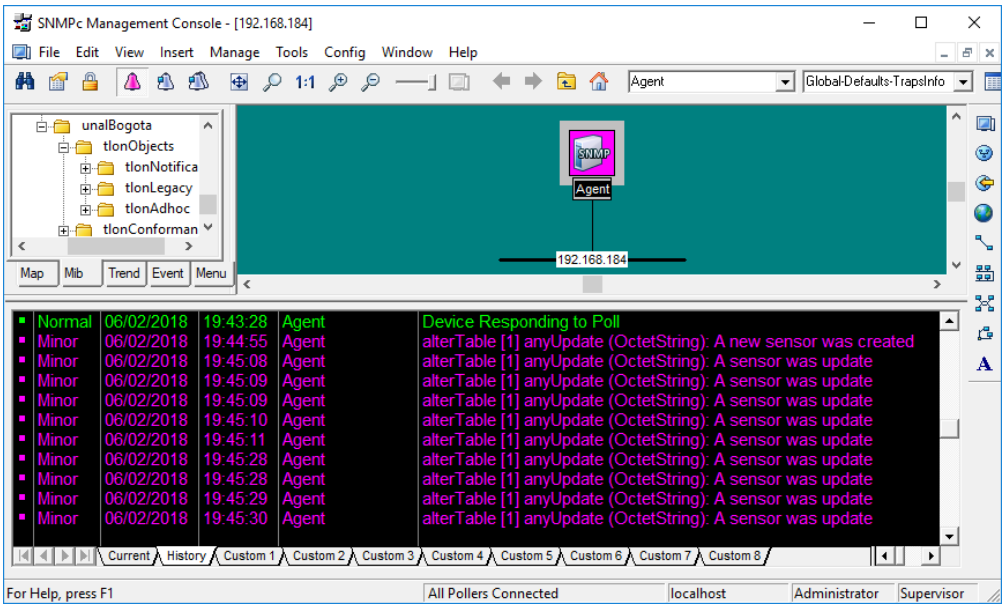


**Figura F-4**.: Traps recibidos por aplicación externa

## F.3. Capturas con sniffer

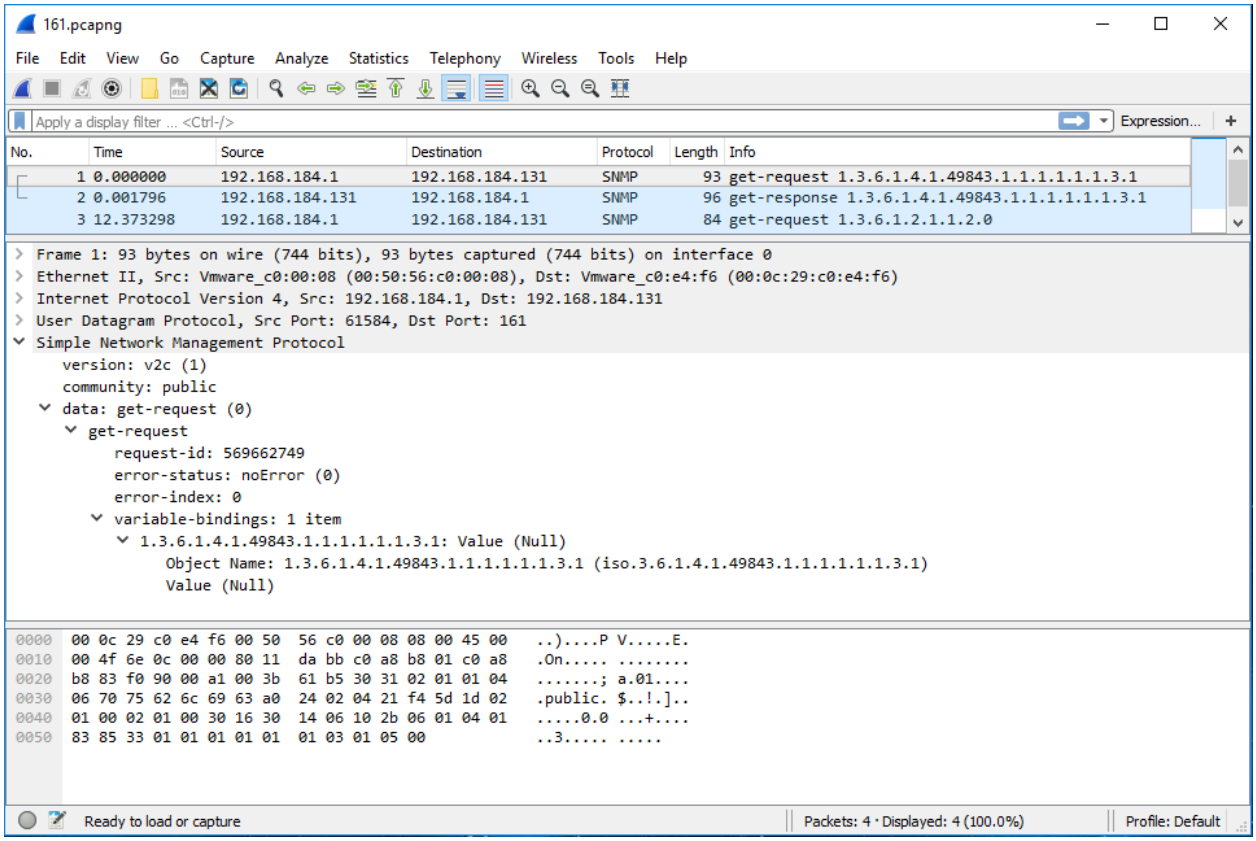


**Figura F-5.**: Mensajes SNMP de solicitud GetRequest
Fuente propia

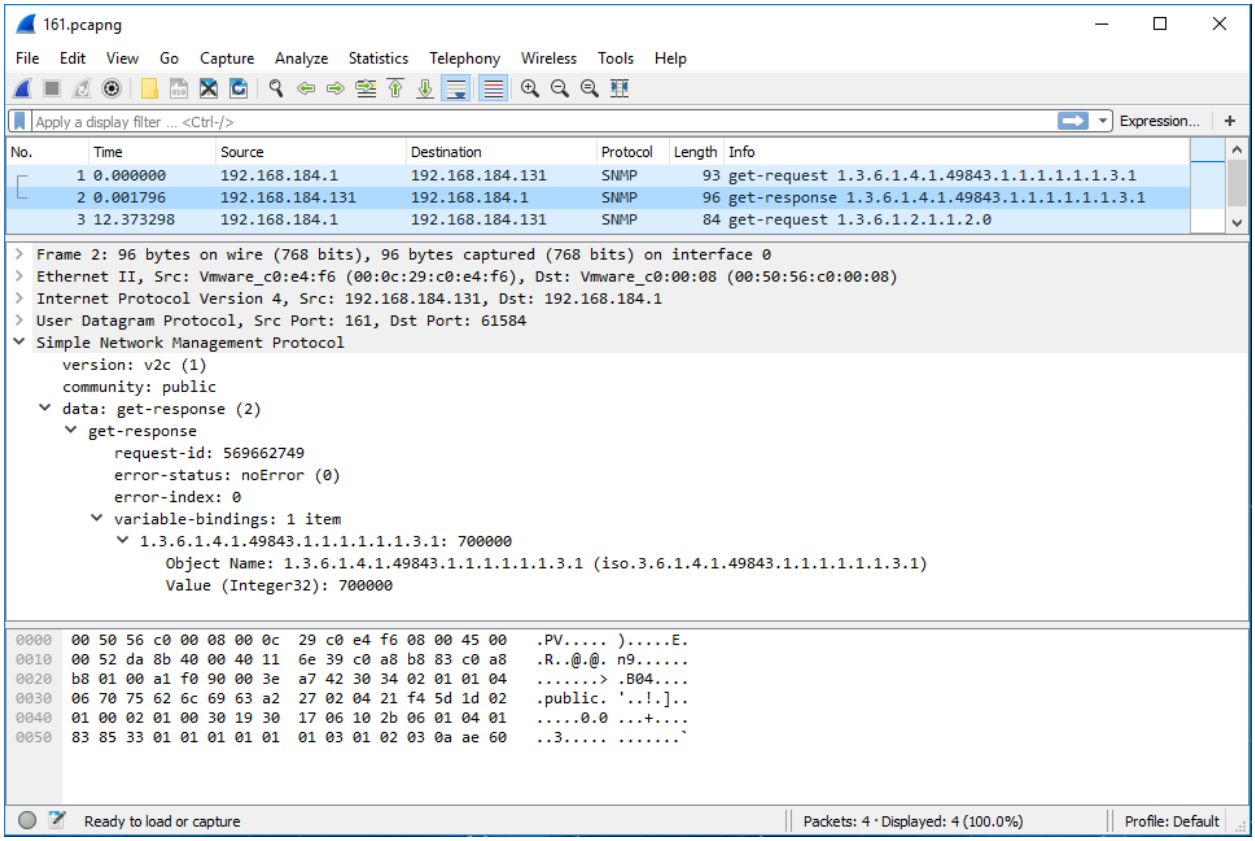


**Figura F-6**.: Mensajes SNMP de respuesta GetResponse
Fuente propia

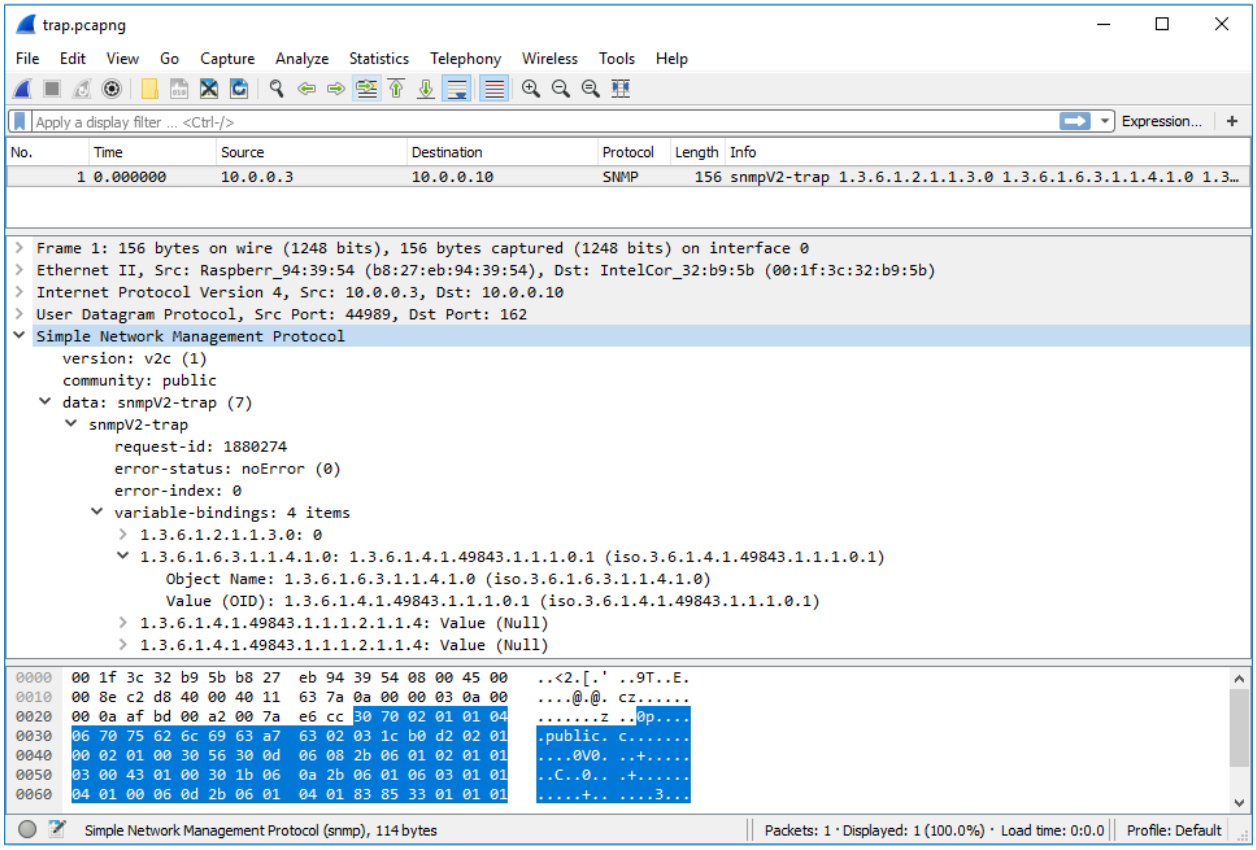


**Figura F-7**.: Mensajes SNMP de notificación Trap
Fuente propia

# Bibliografía


[Axelson, 2007] Axelson, J. (2007). *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems*. Lakeview Research LLC, Madison, 2nd editon edition.

[Benhaddou and Al-Fuqaha, 2015] Benhaddou, D. and Al-Fuqaha, A. (2015). *Wireless Sensor and Mobile Ad-Hoc Networks*. Springer US.

[Bhardwaj and Chandrakasan, 2002] Bhardwaj, M. and Chandrakasan, A. (2002). Bounding the lifetime of sensor networks via optimal role assignments. *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, 00(c):1587–1596.

[Calvert and Lam, 1990] Calvert, K. L. and Lam, S. S. (1990). Formal Methods for Protocol Conversion. *IEEE Journal on Selected Areas in Communications*, 8(1):127–142.

[Case et al., 1990] Case, J., Davin, J., Fedor, M., and Schoffstall, M. (1990). RFC 1157 A Simple Network Management Protocol (SNMP).

[Case et al., 1993] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S. (1993). RFC 1441 Introduction to version 2 of the Internet-standard Network Management Framework Status.

[Case et al., 2002] Case, J., Rose, M., McCloghrie, K., Waldbusser, S., and Presuhn, R. (2002). RFC 3416 Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP).

[Charbonneau, 2013] Charbonneau, N. (2013). Writing an SNMP Agent With a Custom MIB Using Pysnmp. http://www.nealc.com/blog/blog/2013/02/23/writing-an-snmp-agent-with-a-custom-mib-using-pysnmp/.

[Chen et al., 1999] Chen, W., Jain, N., and Singh, S. (1999). ANMP: Ad hoc network management protocol. *IEEE Journal on Selected Areas in Communications*, 17(8):1506–1531.

[Clemm, 2006] Clemm, A. (2006). *Network Management Fundamentals*. Cisco Press, first edition.

[Daogang et al., 2010] Daogang, P. D. P., Hao, Z. H. Z., Hui, L. H. L., and Fei, X. F. X. (2010). Development of the Communication Protocol Conversion Equipment Based on Embedded Multi-MCU and Mu-C/OS-II. *Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on*, 2:15–18.

[Elbert, 2001] Elbert, B. R. (2001). *The Satellite Communication Ground Segment and Earth Station Handbook*. Artech House, Inc., Norwood, MA.

[Etingof, 2017] Etingof, I. (2017). Python SNMP library for Python. http://pysnmp.sourceforge.net/.

[Goka and Shigeno, 2018] Goka, S. and Shigeno, H. (2018). Distributed management system for trust and reward in mobile ad hoc networks. *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6.

[Green, 1986] Green, P. (1986). Protocol Conversion. *IEEE Transactions on Communications*, 34(3):257–268.

[Harrington et al., 2002] Harrington, D., Presuhn, R., and Wijnen, B. (2002). RFC 3411 An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks.

[Heard, 2005] Heard, C. (2005). RFC 4181 - Guidelines for Authors and Reviewers of MIB Documents.

[Huijgens et al., 2016] Huijgens, H., van Deursen, A., and van Solingen, R. (2016). Success Factors in Managing Legacy System Evolution: A Case Study. *Proceedings of the International Workshop on Software and Systems Process*, pages 96–105.

[Hura and Singhal, 2001] Hura, G. and Singhal, M. (2001). *Data and Computer Communications*.

[Irastorza et al., 2006] Irastorza, J. A., Aguero, R., Gutierrez, V., and Munoz, L. (2006). Beyond Management in Ad Hoc, Heterogeneous WPAN Environments: an Experimental Approach. *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 1–4.

[ISO and IEC, 1995] ISO and IEC (1995). ISO/IEC 8602 Information Technology - Protocol for Providing the OSI Connectionless-mode Transport Service.

[ISO and IEC, 1996] ISO and IEC (1996). ISO/IEC 8072 Information technology - Open Systems Interconnection - Transport Service Definition.

[ISO et al., 2006] ISO, IEC, and IEEE (2006). *ISO/IEC 16085 IEEE Std 16085-2006 - Systems and software engineering — Life cycle processes — Risk management*. Geneva, second edition.

[ISO et al., 2011] ISO, IEC, and IEEE (2011). ISO/IEC/IEEE 29148 - Systems and software engineering — Life cycle processes — Requirements engineering. *Iso/ Iec/ Ieee*, pages 1–83.

[ITU-T, 1993] ITU-T (1993). X.210: Data Networks and open system communications.

[Jacquot et al., 2010] Jacquot, A., Chanet, J.-P., Hou, K. M., De Sousa, G., and Monier, A. (2010). A new management method for wireless sensor networks. *2010 The 9th IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–8.

[Jiménez et al., 2014] Jiménez, M., Palomera, R., and Couvertier, I. (2014). Principles of Serial Communication. In *Introduction to Embedded Systems*, chapter Nine, pages 475–536. Springer-Verlag New York, New York, NY, first edition.

[Kabashi and Elmirghani, 2010] Kabashi, A. and Elmirghani, J. (2010). Adaptive rate control & collaborative storage management for challenged ad hoc & sensor networks employing static & dynamic heterogeneity. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, (3).

[Koerner, 1997] Koerner, E. (1997). Design of a proxy for managing CMIP agents via SNMP. *Computer Communications*, 20(5):349–360.

[Lam, 1988] Lam, S. S. (1988). Protocol Conversion. *IEEE Transactions on Software Engineering*, 14(3):353–362.

[Levi et al., 2002] Levi, D., Meyer, P., and Stewart, B. (2002). RFC 3413 Simple Network Management Protocol (SNMP) Applications.

[Liechti, 2015] Liechti, C. (2015). Welcome to pySerial's documentation. https://pythonhosted.org/pyserial/.

[Mathule and Kalema, 2016] Mathule, R. L. and Kalema, B. M. (2016). User Acceptance of Legacy Systems Integration. pages 1–8.

[Mauro and Schmidt, 2005] Mauro, D. and Schmidt, K. (2005). *Essencial SNMP*. O'Reilly Media, Sebastopol, CA, second edition.

[McCloghrie et al., 1999a] McCloghrie, K., Perkins, D., Case, J., Rose, M., and Waldbusser, S. (1999a). RFC2578 Structure of Management Information Version 2 (SMIv2).

[McCloghrie et al., 1999b] McCloghrie, K., Perkins, D., Case, J., Rose, M., and Waldbusser, S. (1999b). RFC2579 Textual Conventions for SMIv2 Status.

[Miller, 1999] Miller, M. A. (1999). *Managing Internetworks with SNMP*. Wiley, Foster City, CA, third edit edition.

[Nowicki and Malinowski, 2016] Nowicki, K. and Malinowski, A. (2016). Topology discovery of hierarchical Ethernet LANs without SNMP support. *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, pages 5439–5443.

[Shen et al., 2002] Shen, C.-C., Jaikaeo, C., Srisathapornphat, C., and Huang, Z. (2002). The Guerrilla Management Architecture for Ad hoc Networks. *Proc. of {IEEE} MILCOM*, pages 1–6.

[Sinha, 2015] Sinha, R. (2015). Conversing at Many Layers: Multi-layer System-on-Chip Protocol Conversion. *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 170–173.

[Sommerville, 2011] Sommerville, I. (2011). *Software Engineering*.

[TLÖN, 2017] TLÖN (2017). Proyecto TLÖN. http://www.tlon.unal.edu.co.

[Walsh, 2008] Walsh, L. (2008). *SNMP MIB Handbook*. Wyndham Press, second edition.

[White et al., 2016] White, K. J. S., Pezaros, D. P., and Knudson, M. D. (2016). A Programmable Resilient High-Mobility SDN + NFV Architecture for UAV Telemetry Monitoring. (June):2–7.

[Zárate-Ceballos et al., 2015] Zárate-Ceballos, H., Sanchez-Cifuentes, J. F., Ospina-López, J. P., and Ortiz-Triviño, J. E. (2015). 44. Sistema de Telecomunicaciones Social-Inspirado mediante Comunidades de Agentes. *Cicom*, page 1.