UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

INTERCONEXIÓN DE UNA RED IOT INTELIGENTE POR MEDIO DEL PROTOCOLO DE COMUNICACIÓN XMPP, IMPLEMENTADA SOBRE RASPBERRY PI 3B+

TESIS Y EXAMEN PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN TELECOMUNICACIONES,
SISTEMAS Y ELECTRÓNICA

PRESENTA:
ALBERTO MARTÍNEZ CONTRERAS

ASESOR:
DR. DAVID TINOCO VARELA

CUAUTITLÁN IZCALLI, ESTADO DE MÉXICO, 2019

VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS
SUPERIORES-CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Interconexión de una red IoT inteligente por medio del protocolo de comunicación XMPP, implementada sobre Raspberry Pi 3b+

Que presenta el pasante: **ALBERTO MARTÍNEZ CONTRERAS**
Con número de cuenta: **41401435-6** para obtener el Título de la carrera: **Ingeniería en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 22 de Febrero de 2019.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Dr. David Tinoco Varela	
SECRETARIO	Ing. Jorge Ramírez Rodríguez	
1er. SUPLENTE	Ing. Noemi Hernández Domínguez	
2do. SUPLENTE	Mtro. Leopoldo Martín del Campo Ramírez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/ntm*

Dedicatoria

Dedicada a todas las personas que me apoyaron y me exigieron excelencia estando seguras que podía lograr grandes cosas.

Agradecimientos

Son muchas las personas que me han apoyado y aconsejado a lo largo de este camino, que no fue sencillo, pero ha valido la pena recorrerlo. En primer lugar, agradezco a mi familia, profesores y compañeros quienes invirtieron parte de su tiempo para apoyarme en mi carrera y en mi proyecto de tesis.

Asimismo, agradezco a la Facultad de Estudios Superiores Cuautitlán donde me he formado y de la que siempre he recibido apoyo y a los proyectos de la UNAM: PIAPIVC06, PAPIIT IN 105219, PAPIME PE111519.

Resumen

En este proyecto se emplea el protocolo XMPP (Protocolo extensible de mensajería y comunicación de presencia) para comunicar en tiempo real nodos IoT (Internet de las Cosas) que comparten entre si datos y comandos. La versatilidad y popularidad del protocolo permiten implementarlo eficientemente en diferentes dispositivos.

Se ha diseñado una red con cuatro nodos y se ha descrito su algoritmo de funcionamiento utilizando un servidor XMPP gratuito. La red diseñada tiene actuadores y monitorea en tiempo real variables físicas. Sus elementos son dos Raspberry Pi, una PC con Windows y un *smartphone* con Android.

Después de someter a distintas pruebas la red, se observó que la comunicación es muy eficiente, el retardo para la entrega de mensajes es mínimo e implementar la comunicación con este protocolo es sencillo debido a la multitud de librerías que existen para todo tipo de sistemas operativos y lenguajes de programación.

Analizando los resultados se ha concluido que la comunicación con este protocolo es muy fácil de implementar y es prácticamente en tiempo real haciéndolo apropiado para proyectos IoT sencillos y complejos.

Palabras clave

XMPP, IoT, Red inteligente, Raspberry Pi.

Abstract

In this project XMPP (Extensible Messaging and Presence Protocol) is used in real-time communications between IoT (Internet of Things) nodes that share data and commands. Popularity and versatility of this protocol allow it to be implemented efficiently in different devices.

A network with four nodes and an algorithm for operation were designed for this project using a free XMPP server. Designed network has actuators and it monitors in real time physical variables. Its elements are two Pi Raspberry, a Windows PC and a smartphone with Android.

After different tests, the results show that communication with this protocol is really efficient, the delay for deliver messages is short and implement communication using this protocol is easy because there are several libraries for different programing languages and operative systems.

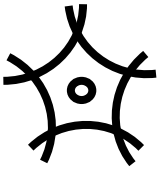
Analyzing the results, it has been concluded that communication with this protocol is very fast to implement and is almost in real time, so it is appropriate for simple and complex IoT projects.

Key words

XMPP, IoT, Smart network, Raspberry Pi.

Índice

1. INTRODUCCIÓN



Introducción

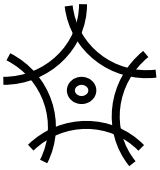
El internet de las cosas o IoT (Internet Of Things), es un paradigma emergente en el que toda clase de objetos usados en la vida cotidiana son conectados a internet para ser monitoreados y controlados de forma remota, ya sea por personas u otros objetos.

Comunicar las "cosas" mediante internet requiere "hacerlas inteligentes", esto se refiere a que tengan un sistema de control que colecte los datos necesarios y efectúe las acciones apropiadas, en la mayoría de casos se utiliza un sistema miniaturizado basado en microprocesador que pueda efectuar funciones de control y comunicación, pero que no demande un gran consumo energético ni de espacio, las computadoras de placa única o SBC (Single Board Computer), como la *Raspberry Pi* son ampliamente utilizadas.

Una vez teniendo los dispositivos "inteligentes" que formaran la red, es requerido un "lenguaje" que todos entiendan para que la comunicación sea exitosa, aquí entran en juego protocolos de comunicación y mensajería que se encargan de hacer llegar mensajes entre diferentes dispositivos conectados a internet, un ejemplo es el protocolo extensible de mensajería y comunicación de presencia o XMPP (Extensible Messaging and Presence Protocol), utilizado ampliamente para servicios de mensajería instantánea.

Actualmente grandes empresas están desarrollando protocolos especializados para funcionar en redes IoT, que se enfocan en los requerimientos específicos de este tipo de redes y son una opción adecuada para la comunicación entre los diferentes dispositivos. Estos protocolos tienen algunas desventajas importantes como su complejidad, compatibilidad con diferentes dispositivos y el tipo de licencia que autoriza su implementación, muchos no son libres, y se requiere invertir en licencias y/o servidores para utilizarlos. Esto no representa problema para las grandes empresas, pero si para los proyectos a pequeña escala.

Este proyecto demuestra que el protocolo de mensajería instantánea XMPP que es un protocolo libre y ofrece servidores gratuitos se puede implementar en redes IoT para lograr una comunicación entre los



dispositivos que la conforman, independientemente del sistema operativo o lenguaje de programación de cada dispositivo.

En el presente trabajo se propone un procedimiento para comunicar nodos dentro de una red IoT empleando el protocolo XMPP. Los nodos pueden estar ubicados en cualquier parte del mundo en donde se tenga conexión a internet. Sin tomar en cuenta el costo de los dispositivos utilizados, la implementación y mantenimiento de la red no requiere ninguna inversión monetaria.

1.1 Objetivos

- Demostrar que el protocolo de mensajería instantánea XMPP se puede implementar en diferentes dispositivos electrónicos basados en microprocesador, independientemente del sistema operativo o lenguaje de programación de cada uno.

- Proponer un procedimiento para comunicar nodos dentro de una red IoT sin importar en que parte del mundo se encuentren, empleando el protocolo XMPP.

- Implementar una red de varios nodos IoT que monitoreen parámetros físicos utilizando sensores, procesen los datos recolectados, interactúen con su entorno mediante actuadores y se comuniquen entre sí y con el usuario, en tiempo real a través de internet.

1.2 Hipótesis

"El protocolo de mensajería instantánea XMPP puede ser implementado en diferentes aparatos electrónicos sin importar el sistema operativo o lenguaje de programación que estos tengan y se puede utilizar como una forma de comunicación adecuada en un sistema IoT, siempre que los nodos y el usuario tengan una conexión a internet."

1.3 Motivación

La motivación que ha generado este trabajo es la necesidad de conectar nodos IoT entre sí y con diferentes usuarios conectados a internet, utilizando un protocolo de código abierto que pueda ser empleado de forma transparente al sistema operativo o lenguaje de programación de los dispositivos que forman la red.

1.4 Estado del arte

Antes de hablar de la historia del internet de las cosas tal y como se conoce hoy en día primero se debe hablar de los elementos que lo hacen posible, pues el concepto de esta tecnología es relativamente nuevo.

Desde varios años atrás se tenía la visión de máquinas comunicándose entre sí pero no se consiguió hasta el año 1830 cuando se inventó el primer telégrafo que revolucionó las comunicaciones, ya que permitía la comunicación a larga distancia y de forma instantánea, algo asombroso en aquella época.

Se le suele dar la autoría a un solo hombre, pero realmente fue el resultado de una cadena de aportes realizados por varios investigadores, sin embargo, fue el fotógrafo y pintor *Samuel F. B. Morse* que nació en Charlestown, Massachusetts, el 27 de abril de 1791 quien se considera consiguió crear en 1837 el primer telégrafo, además de crear un alfabeto para transmitir la información que tiempo después llevaría su nombre, el código morse.

Años más tarde fue creado el primer radio de voz en junio de 1900 basado en la teoría de la propagación de ondas electromagnéticas descrita por *James Clerk Maxwell* en un documento dirigido a la *Royal Society* en 1873 titulado "*Una teoría dinámica del campo electromagnético*".

Durante varios años, a partir de 1894, el inventor italiano *Guglielmo Marconi* construyó el primer sistema completo de telegrafía inalámbrica comercialmente exitoso basado en ondas hertzianas transportadas por el aire (transmisión por radio). Marconi demostró la aplicación de la radio en

comunicaciones militares y marinas e inició una empresa para el desarrollo y la propagación de servicios y equipos de comunicación por radio.

Pero no fue hasta 1977 cuando se comenzaron a desarrollar las computadoras personales como dispositivos electrónicos de consumo para el mercado masivo que el internet de las cosas tal y como se conoce empezó a visualizarse. Pues antes de la introducción del microprocesador a principios de 1970, las computadoras eran sistemas grandes y costosos cuyos dueños eran corporaciones, universidades, agencias gubernamentales, e instituciones de tamaño similar.

Con la comercialización de la "computadora en un chip", el costo para manufacturar un sistema de cómputo cayó dramáticamente. Posteriormente, los avances en el desarrollo de la memoria de estado sólido eliminaron la abultada y costosa memoria de núcleo magnético.

James Finke, Presidente de Commodore International, Febrero de 1982

> *"El actual mercado de computadoras personales es de aproximadamente el mismo tamaño que el total del mercado de patatas fritas. El año que viene va a ser aproximadamente la mitad del tamaño del mercado de alimentos para animales, y se acerca rápidamente al total de las ventas mundiales de pantimedias."*

El internet, componente esencial de IoT, tuvo sus inicios en el año 1962 como el protocolo DARPA (*Defense Advanced Research Projects Agency*) que evoluciono a ARPANET (*Advanced Research Projects Agency Network*) en 1969, pero fue hasta 1980 que algunos proveedores de servicios comenzaron a ofrecer el uso público y comercial de ARPANET que con el paso del tiempo se convirtió en lo que hoy conocemos como internet.

Un pionero fundamental en lo que se refiere a una red mundial (como se conocía en esos años), fue *J. C. R. Licklider* quien comprendió la necesidad de una red mundial, según consta en su documento de enero, 1960, "*Man-Computer Symbiosis*".

> *"Una red de muchas computadoras, conectados mediante líneas de comunicación de banda ancha las cuales proporcionan las funciones que existen hoy en día en las bibliotecas, junto con almacenamiento, adquisición de datos y otras funciones simbióticas."*

Otra tecnología usada en gran número de sistemas IoT es el GPS (*Global Positioning Satellites*) y las redes satelitales, que se hicieron posibles hasta 1993 cuando el departamento de defensa de los Estados Unidos de América desplego una red de 24 satélites con alta responsividad y que ofrecía múltiples servicios, esta idea inmediatamente fue copiada por otros países y por empresas privadas, que lanzaron satélites para uso comercial.

Un elemento más de IoT es el protocolo IP (*Internet Protocol*), que es de gran importancia pues permite que los dispositivos conectados puedan ser identificados con una dirección única a nivel mundial, existen dos versiones de este protocolo, la primera de ellas es IPV4 desarrollada en los inicios de internet cuando aún no se sabía la popularidad que tendría este servicio, por lo que el espacio de direccionamiento de este protocolo esta por agotarse, para sobreponerse a este problema se desarrolló una versión mejorada, IPV6 que tiene una cantidad de direcciones suficientes para que cada átomo de la tierra tenga una dirección IP y se puedan tener 100 planetas tierra más.

El internet de las cosas no fue nombrado oficialmente hasta 1999, a pesar de que en años anteriores se desarrollaran sistemas con este concepto, en 1980 la compañía de Coca Cola, en la universidad de Carnegie Mellon, colocó una maquina dispensadora de refrescos a la que los técnicos de la universidad se podían conectar mediante internet, para revisar si había refrescos fríos en la maquina antes de ir por él.

Para el año 2013 el internet de las cosas ya era una tecnología que utilizaba muchas otras para funcionar como los radios y comunicaciones inalámbricas, la electrónica de microprocesadores, MEMS (*Microelectromechanical Systems*) y sistemas embebidos, tecnología GPS, sensores de diferentes tipos y sistemas de control, entre muchos otros.

Actualmente IoT, consiste en cualquier dispositivo conectado a internet que implemente funciones utilizando la red, esto incluye toda clase de aparatos imaginables desde celulares, aviones, autos, electrodomésticos, sistemas de monitoreo en fábricas, aparatos médicos, chips de rastreo, entre muchos otros.

IoT es una tecnología que avanza constantemente, y de la cual se podrán obtener grandes beneficios (seguridad, monitoreo de información médica,

compras automáticas, entre otros) y podrá aplicarse para solucionar una gran cantidad de fenómenos y problemáticas.

Pero hablar de internet de las cosas hoy en día es hablar de objetos cotidianos que tienen la capacidad de compartir información a través de internet, estos objetos podrían ser vehículos, electrodomésticos, dispositivos mecánicos, o simplemente objetos tales como calzado, muebles, maletas, dispositivos de medición, biosensores, o cualquier otro objeto existente.

Muchos proyectos nuevos sobre internet de las cosas se desarrollan cada año, a continuación, se mencionarán algunos que están relacionados con la línea de investigación de este proyecto.

Desde hace varios años se tenía la visión de interactuar con los objetos de forma parecida a como interactuamos con las personas en las redes sociales como se presenta en el artículo "*Uniting online social networks with places and things*" [1] que a pesar de tener varios años de ser publicado tiene argumentos válidos hasta hoy en día.

Varios autores han realizado análisis de como los objetos inteligentes interactuarán con las personas, a pesar que los expertos han hecho predicciones sobre su inclusión en la vida de las personas siempre pueden surgir comportamientos inesperados como se presenta en el artículo "*Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things*" [2].

En el artículo "A Survey on Application Layer Protocols for the Internet of Things" [3] se hace mención de las diferentes capas de protocolos que componen IoT, así como su análisis y como deben cuidarse los diferentes aspectos del desarrollo de IoT.

En el artículo "Chatty things - Making the Internet of Things readily usable for the masses with XMPP" [4] se propone el protocolo XMPP para comunicaciones IoT, aunque en ese año (2012) aun no existían las placas computadoras ni microcontroladores tan sofisticados como hoy en día para hacer la implementación.

Un artículo más que analiza la posibilidad de integrar XMPP en IoT es "A Service Infrastructure for the Internet of Things based on XMPP" [5] en el que se justifica el potencial que tiene XMPP para esta tecnología.

En el artículo "The VIRTUS Middleware: An XMPP Based Architecture for Secure IoT Communications" [6] los autores analizan las comunicaciones con el protocolo XMPP para utilizarlo en IoT enfatizando en las características de seguridad que proporciona para hacer frente a las diferentes amenazas que enfrentan este tipo de sistemas.

En el *IEEE International Symposium on Consumer Electronics* (ISCE 2010) se presentó el artículo "From instant messaging to cloud computing, an XMPP review" [7] en el que se describe como se puede dar un uso masivo al protocolo XMPP para escalarlo a un nivel de miles de comunicaciones entre computadoras a la vez.

En el año 2016, el *Doctor José Ángel Noguera Arnaldos* presento como tesis doctoral el proyecto "Sistema de diálogo basado en mensajería instantánea para el control de dispositivos en el internet de las cosas" [8] en el que utiliza el protocolo XMPP para establecer un sistema de dialogo entre usuarios y nodos IoT, este trabajo se enfoca en el algoritmo para analizar y generar el dialogo y utiliza el protocolo XMPP solo como el sistema de comunicación entre nodo y usuario.

Por otro lado en el trabajo titulado "*Fuzzification of facial movements to generate humanmachine interfaces in order to control robots by XMPP internet protocol*" [9], se generó el control de un vehículo robotizado, utilizando XMPP como protocolo de comunicación.

Recientemente se han propuestos modelos de comunicación con los objetos IoT como se presenta en el artículo "Method And System For Controlling Internet Of Things (Iot) Device" [10] publicado en el año (2017) cuyos formatos de comunicación se basan en la tecnología de mensajería instantánea o chats en línea.

Como se puede ver, la idea de "chatear con los objetos" viene desde varios años antes, sin embargo, la tecnología para conseguirlo tiene pocos años de desarrollo y aún sigue en fase de pruebas, los microcontroladores actuales

permitirán hacer "inteligentes a los objetos" y el desarrollo de diferentes protocolos como XMPP harán posible que estos se comuniquen a una escala masiva.

1.5 Metodología y cronograma de actividades

El desarrollo de este proyecto se realizará mediante el siguiente proceso:

- **Identificación de objetivos.** Se propondrán los objetivos principales y secundarios del proyecto tales como: Problema que se planea resolver y Tecnologías a usar.

- **Estudio del panorama.** Llevar a cabo un estudio del estado del arte con los avances más significativos relacionados a la línea de investigación. Así mismo se realizará una búsqueda de proyectos similares.

- **Investigación.** Una vez establecidos los objetivos se realizará una investigación bibliográfica tomando como fuentes artículos y libros, además de proyectos similares realizados recientemente.

- **Propuesta de diseño.** Teniendo en cuenta los requerimientos del sistema y la investigación que será realizada sobre componentes y compatibilidad, se diseñara el modelo de conexión, además de los códigos que estarán ejecutando cada uno de los dispositivos involucrados en este proyecto.

- **Implementación.** En esta etapa el diseño propuesto en el punto anterior será implementado y será llevado un registro detallado de los pasos que se realicen, para que en el futuro este trabajo pueda ser utilizado como guía para proyectos similares.

- **Pruebas y resultados.** Se realizarán todas las pruebas necesarias para asegurar que el sistema está operativo y cumple los objetivos propuestos, se contrapondrá el trabajo final con los objetivos e hipótesis para obtener las conclusiones pertinentes.

Cronograma

En la Tabla 1. Se muestra la distribución temporal de las etapas antes mencionadas.

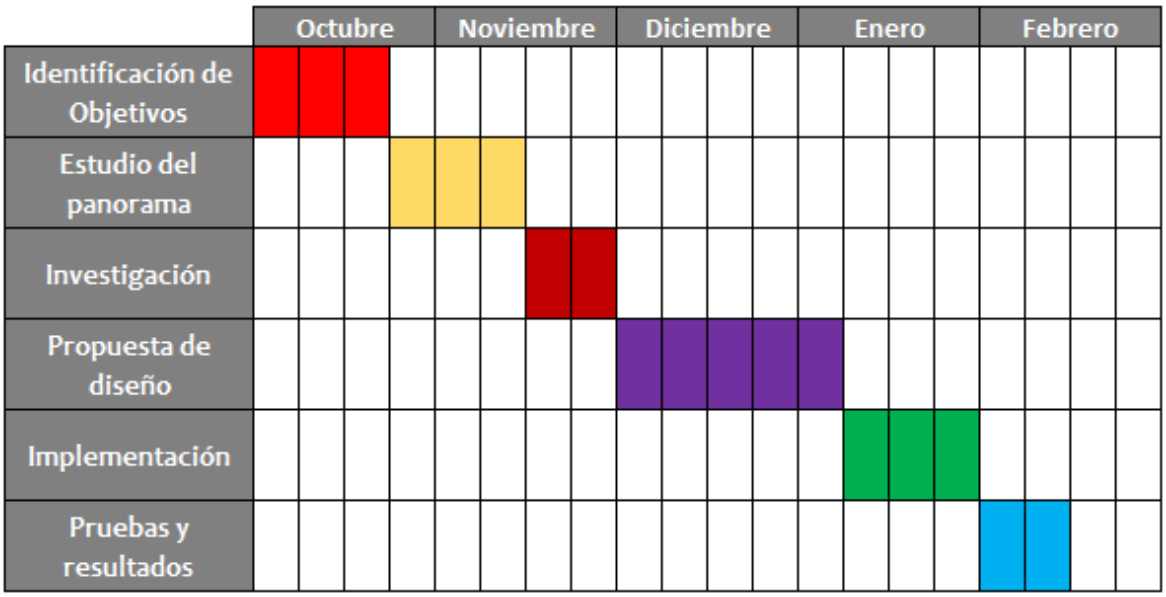
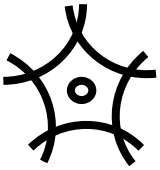


Tabla 1.1 Cronograma de actividades

2. MARCO TEÓRICO



2.1 Internet de las cosas (IoT)

Algunas empresas calculan que un ser humano citadino promedio está rodeado de entre 1.000 y 5.000 objetos en su vida diaria, que, con las nuevas tecnologías podrían convertirse en "objetos inteligentes". Diversas universidades y empresas de desarrollo de tecnológico calculan que en 2020 habrá entre 25.000 y 30.000 millones de dispositivos conectados a internet [11].

Fig. 2.1 Esquema de internet de las cosas

Gracias al crecimiento del internet y el nuevo protocolo IPv6 en un futuro cercano se podrá identificar instantáneamente por medio de una dirección IP única a cualquier objeto a nivel mundial y de esta forma mandarle comandos a través de internet (Fig. 2.1), también será posible la recepción de información recolectada por tales objetos.

2.1.1 Objetivo de IoT

El objetivo que pretende alcanzar esta tecnología es hacer más cómoda la vida de los seres humanos en diversos ámbitos, proporcionando mayor seguridad y conectividad entre otras funciones.

Según diversos investigadores alrededor del mundo en muy poco tiempo será posible conectar toda clase de objetos a internet [2], con el abaratamiento de las tecnologías, los nuevos protocolos de direccionamiento y el crecimiento del internet, en el futuro todo objeto será dotado de "inteligencia".

La "inteligencia" o capacidades que se pretenden agregar a los objetos se clasifican en grupos.

- **Comunicación y cooperación.** Tendrán la capacidad de conectarse a los servicios de internet, para comunicarse y compartir información con otros objetos, con los servidores y con los usuarios.
- **Direccionamiento.** Tendrán una dirección única para poder localizarlos a nivel mundial, este identificador será una dirección IPv6.
- **Identidad.** Además de una dirección, también serán dotados de un identificador único e irrepetible, similar a las huellas digitales, esto para evitar la suplantación de identidad.
- **Localización.** Tendrán información sobre su ubicación por medio de la tecnología GPS o similares.
- **Acción.** Tendrán la capacidad de modificar su entorno, por ejemplo, un calentador podrá aumenta la temperatura ambiental.

2.1.1.1 Capacidades

Los nodos IoT serán dotados de "inteligencia" en mayor o menor medida, se puede clasificar en diferentes niveles según las capacidades del objeto.

- **Nivel 1 - Identidad.** El objeto será capaz de identificarse a sí mismo.
- **Nivel 2 – Ubicación.** El objeto podrá reconocer en donde está actualmente o en donde ha estado en un tiempo anterior.
- **Nivel 3 – Estado.** Sera capaz de comunicar su estado de funcionamiento.
- **Nivel 4 – Contexto.** Sera capaz de percibir y analizar el entorno en el que se encuentra.
- **Nivel 5 – Criterio.** Además de analizar su entorno podrá tomar acciones según los datos analizados, como encender la calefacción si la temperatura es muy baja.

Estos niveles de inteligencia no son estandarizados, cada empresa e institución tiene parámetros diferentes para la clasificación.

2.1.2 Arquitectura

Existen dos aspectos clave en los que se divide la arquitectura IoT para su análisis, el primero son los dispositivos o nodos y el segundo la arquitectura de red.

Los dispositivos se clasifican en tres dependiendo de sus características.

- **Dispositivos pequeños.** Estas plataformas no suelen tener un sistema operativo, por ejemplo, controladores de 8 bits.
- **Dispositivos intermedios.** Algunas de estas plataformas corren un sistema operativo especialmente diseñado, basados en Linux o Windows IoT. Son dispositivos con una arquitectura de 32 bits usualmente basada en ARM.
- **Plataformas.** Son dispositivos completos de 32 o 64 bits, por ejemplo, Raspberry Pi. Tienen instalado un sistema operativo como Linux o Android. También pueden ser Smartphones o dispositivos basado en tecnologías móviles, pueden funcionar como puentes para dispositivos más pequeños. Por ejemplo, un Arduino que se conecta vía Bluetooth a un Smartphone o a una Raspberry Pi y la utiliza como un puente para acceder a Internet.

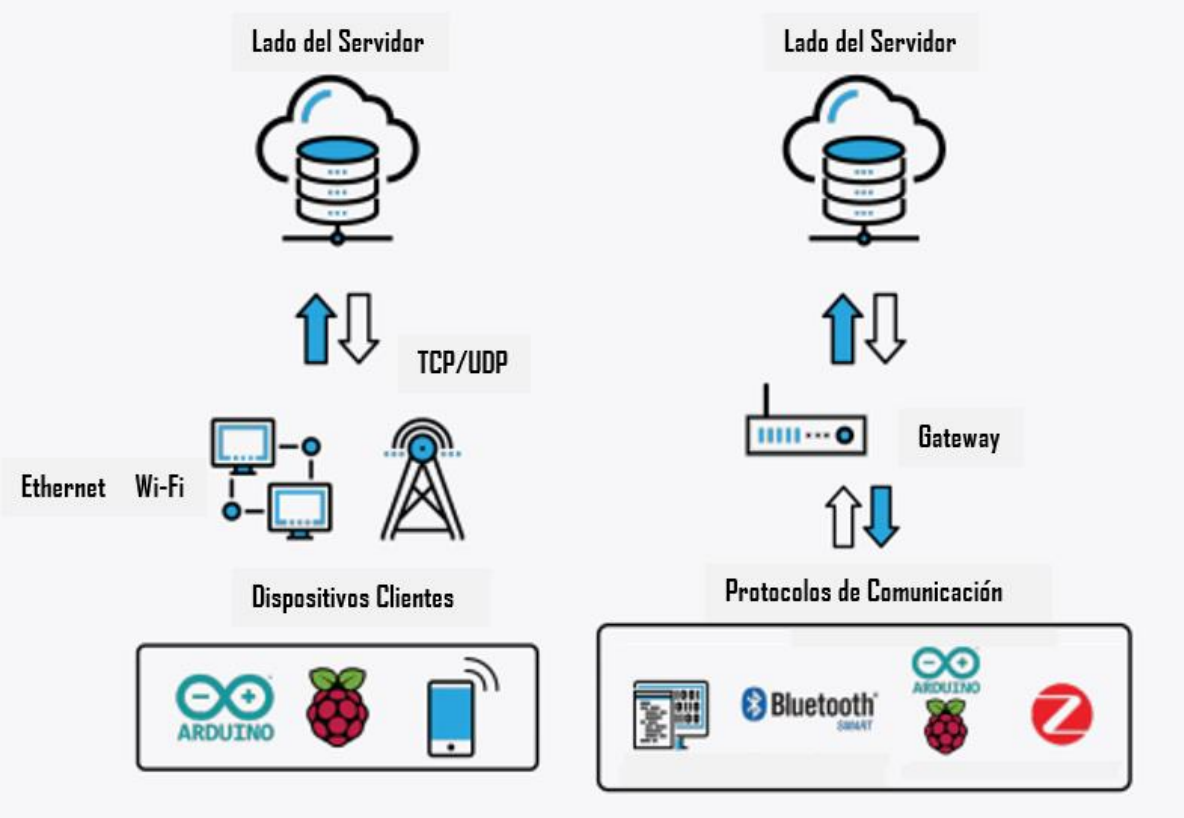


Fig. 2.2 Arquitectura IoT con ejemplos de protocolos

Una arquitectura IoT funcional (Fig. 2.2), debe tener como mínimo las siguientes características.

- **Comunicaciones y conectividad.** Comunicación segura entre dispositivos, clientes y servidores.
- **Gestión y control de dispositivos.** Saber en todo momento que dispositivos están conectados a la red.
- **Captación y análisis de información.** Procesamiento adecuado de los datos recolectados por los dispositivos, para poder tomar acciones.
- **Escalabilidad.** La red debe admitir nuevos usuarios, dispositivos y tecnologías con un mínimo de modificaciones.
- **Alta disponibilidad.** Los dispositivos siempre deben estar disponibles, se deben tomar medidas inmediatas en caso de problemas en la conexión.

Además de las características antes mencionados, la arquitectura IoT debe poder realizar las siguientes funciones elementales.

- Desconexión de un dispositivo malicioso o robado.
- Actualización de software en los dispositivos.
- Actualizaciones de credenciales de seguridad.
- Habilitar o deshabilitar ciertas opciones de hardware.
- Localización un dispositivo perdido o robado.
- Eliminar la información de un dispositivo robado.
- Re-configurar la configuración de red remotamente.

Como es evidente la arquitectura de una red IoT es muy compleja y tiene diversos componentes, para hacer más fácil el análisis se divide en cuatro capas (Fig. 2.3).

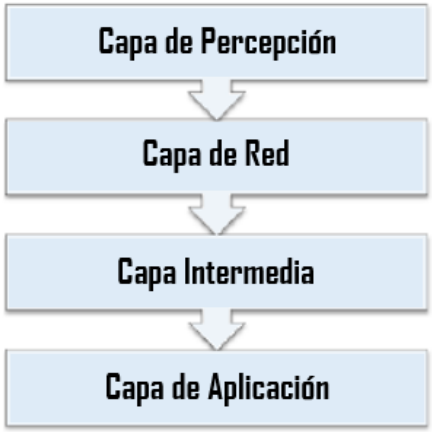


Fig. 2.3 Capas de la arquitectura IoT

- **Capa de percepción.** Consiste en diferentes tipos de sensores para la adquisición de datos como pueden ser, sensores de temperatura, sensores de humedad, etc. La tarea de esta capa es identificar los objetos que están en la red y colectar los datos que están recolectando.

- **Capa de red.** El propósito de esta capa es transmitir la información colectada por los objetos a el sistema de procesamiento que la analizará, mediante internet o algún otro protocolo.

- **Capa de Intermedia.** Esta capa consiste en el sistema para procesar los datos colectados, la toma de decisiones en base a estos y su transferencia a una base de datos para ser almacenados.

- **Capa de Aplicación.** En esta capa están las aplicaciones que interactúan con el usuario ya sea para grandes industrias o usuarios individuales.

Las capas mencionadas no son un estándar, cada empresa o institución puede proponer su propia arquitectura.

2.1.2.1 Tipos de redes

Un sistema IoT se divide en tres niveles (Fig. 2.4).

- **Dispositivos.** También conocidos como nodos, usualmente se encuentran distribuidos en el campo recolectando datos, no tienen una conexión directa a internet, para esto deben conectarse con un gateway.

- **Gateway.** Funciona como frontera entre los nodos IoT e internet, un sistema puede tener más de un gateway para implementar redundancia. Este dispositivo no debe confundirse con un gateway utilizado en redes de datos.

- **Sistema de datos.** Suelen ser servidores ubicados en alguna parte de la nube, los nodos, mediante los gateway se conectan a estos para transferir la información recolectada. Los sistemas de datos suelen implementar algoritmos para el análisis de la información recolectada.

El flujo de datos entre los niveles sigue alguno de los siguientes tipos de transmisión.

- **Dispositivo a dispositivo.** Es la comunicación entre dos nodos IoT que intercambian información de forma directa, sin intermediarios.
- **Dispositivo a gateway.** Es la comunicación entre un nodo IoT y un gateway, tiene el propósito de enviar los datos colectados por el nodo a los sistemas de datos.
- **Gateway a sistema de datos.** El gateway envía la información de los nodos, a un sistema de datos en la nube que se encarga de almacenarla, procesarla, o comunicarla a otros sistemas.
- **Sistema de datos a sistema de datos.** Cuando comparten información entre ellos suele ser para aplicaciones avanzadas de reconocimiento, monitoreo o rastreo.

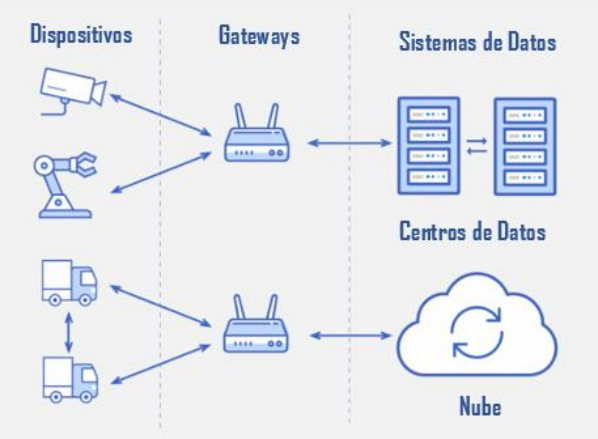


Fig. 2.4 Tipos de elementos en la arquitectura IoT

Las redes IoT se clasifican en varios grupos según el alcance que tienen (Fig. 2.5).

- **Nano.** Un grupo de pequeños dispositivos usualmente distribuidos en un área muy pequeña, para aplicaciones biométricas, de nanotecnología o militares.
- **NFC (Near-Field Communication).** Comunicación de campo cercano. Comunicaciones entre dispositivos a una distancia de unos pocos centímetros, como los sistemas de pago por tarjeta implementados en transporte público.

- **BAN (Body Area Network).** Red de área corporal. Dispositivos ubicados en el cuerpo de una persona comunicándose para aplicaciones como implantes o estudios médicos.
- **PAN (Personal Area Network).** Red de área personal. Los dispositivos están ubicados a pocos metros entre sí, como en un sistema domótico dentro de una casa.
- **LAN (Local Area Network).** Red de área local. Una red que comunica un edificio completo.
- **CAN (Campus/Corporate Area Network).** Red de área corporativa. Esta red comunica a varios dispositivos dentro del mismo campus o unidad habitacional.
- **MAN (Metropolitan Area Network).** Red de área metropolitana. Una red que comunica varios edificios o campus dentro de una ciudad.
- **WAN (Wide Area Network).** Red de área amplia. Redes a gran escala que comunican entre ciudades o países diferentes.

Fig. 2.5 Clasificación de las redes IoT por su alcance

Otra clasificación de las redes IoT es por su topología tanto física como lógica (Fig. 2.6). Cada topología tiene sus ventajas en cuanto a conectividad, pero las desventajas serán el costo y la complejidad del sistema.

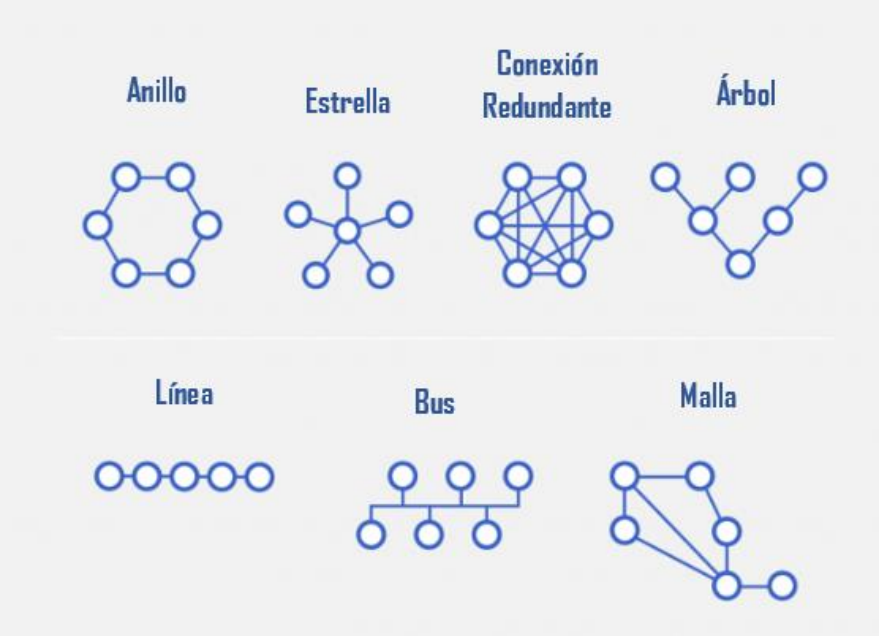


Fig. 2.6 Clasificación de las redes IoT por su topología

2.1.2.2 Protocolos usados en IoT.

Para comunicaciones y mensajería se utilizan diferentes protocolos, estos protocolos no son exclusivos para IoT, a continuación, se muestra una gráfica de su popularidad en IoT (Fig. 2.7).

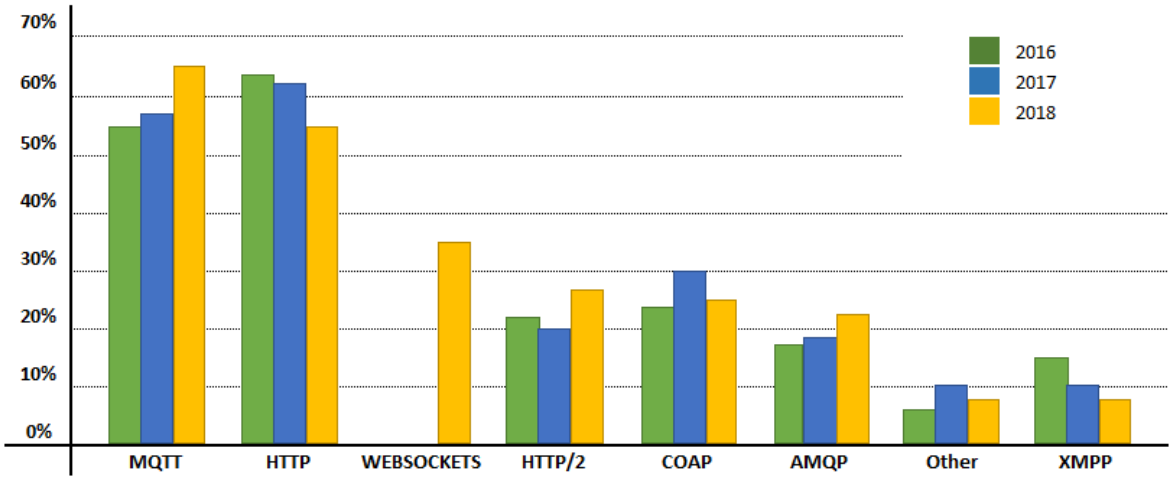


Fig. 2.7 Tendencia en el uso de protocolos de mensajería en IoT

La elección de los protocolos a utilizar en un sistema dependerá de sus requerimientos, las características de los protocolos (Tabla 2.1) deberán adecuarse a las del sistema.

Protocolo	Transporte	Tipo de Mensajería	Recursos del Dispositivo	Seguridad	Arquitectura
CoAP	UDP	Pregunta/ Respuesta	10Kb/RAM/Flash	Opcional	Arbol
HDP	UDP	Suscripción/Pregunta/ Respuesta	10Kb/RAM/Flash	No	Estrella
DDS	UDP	Suscripción/Pregunta/Respuesta	100Kb/RAM/Flash	Opcional	Bus
DPWS	TCP		100Kb/RAM/Flash	Opcional	Cliente/ Servidor
HTTP	TCP	Pregunta/Respuesta	10Kb/RAM/Flash	Opcional	Cliente/ Servidor
MQTT	TCP	Suscripción/Pregunta/ Respuesta	10Kb/RAM/Flash	Opcional	Arbol
SNMP	UDP	Pregunta/Respuesta	10Kb/RAM/Flash	Opcional	Cliente/ Servidor
UPnP		Suscripción/Pregunta/ Respuesta	10Kb/RAM/Flash	No	P2P
XMPP	TCP	Suscripción/Pregunta/ Respuesta	10Kb/RAM/Flash	Opcional	Cliente/ Servidor
ZeroMQ	UDP	Suscripción/Pregunta/ Respuesta	10Kb/RAM/Flash	Opcional	P2P

Tabla 2.1 Comparación de protocolos IoT para comunicaciones

2.1.2.3 Seguridad en IoT

El rápido desarrollo de IoT ha incrementado la demanda de dispositivos inteligentes generando gran preocupación por la seguridad. Los ataques malintencionados podrán pasar del mundo digital al mundo físico, pues los hackers podrían tomar control de dispositivos mediante la red y usarlos para sus propios fines. IoT puede ser el futuro, pero también un potencial desastre en seguridad.

El fácil acceso a los dispositivos puede ser aprovechado por personas malintencionadas, no importa lo mucho que las compañías gasten en seguridad, siempre está la posibilidad de que sean expuestas vulnerabilidades.

Algunos aspectos esenciales en los que se divide la seguridad en IoT son:

- **Confidencialidad.** Consiste en prevenir que datos sensibles puedan ser vistos por personas no autorizadas, se deben implementar mecanismos para que los usuarios no puedan ver datos para los que no estén autorizados. Algunos mecanismos para este fin son, el cifrado de la información, verificaciones biométricas, etc.

- **Integridad.** Durante el proceso de comunicación los datos pueden ser alterados por cibercriminales o por muchos otros factores, incluyendo algunos que no pueden ser controlados por los humanos como factores ambientales, la meta de la integridad es proteger los datos de estos cambios, algunos métodos usados son, sumas de verificación, redundancia cíclica, control de cambios, etc.

- **Disponibilidad.** Se refiere a que la información sea accesible por los usuarios en el momento que la necesiten y bajo las peores condiciones, previniendo que sea el usuario verdadero quien la solicita.

La seguridad en IoT engloba varias capas de abstracción, a su vez cada capa engloba varias dimensiones más, cualquier vulnerabilidad en cualquier dimensión de cualquier capa puede comprometer el sistema entero.

- **Capa de percepción.** Es la capa más baja de la arquitectura IoT y debe brindar las siguientes funciones de seguridad:

    - **Autenticación.** Proporcionar firmas digitales a cada nodo del sistema.

    - **Privacidad.** Usar algoritmos de cifrado para evitar que los datos sean comprometidos cuando son medidos y transferidos a la siguiente capa.

    - **Resguardo de información sensible.** Esconder la información sensitiva, como la localización e identidad de los nodos.

- **Evaluación de riesgos**. Prevenir las brechas de seguridad del sistema e implementar estrategias para detectar intrusiones. Poder dejar inoperable un nodo comprometido.

- **Capa de red.** Esta capa puede ser alámbrica o inalámbrica, debe proporcionar las siguientes funciones de seguridad.

  - **Autenticación**. Con un adecuado cifrado punto a punto, el acceso no autorizado a los sensores y nodos puede ser prevenido.

  - **Seguridad de enrutamiento.** Garantiza que la información llegue de los nodos al sistema de procesamiento, poniendo redundancia para que los paquetes lleguen a su destino a cualquier costo.

  - **Privacidad de datos.** Garantizar que la información no pueda ser vista por usuarios no autorizados.

- **Capa media y de aplicación.** Se puede analizar estas dos capas como una sola para la implementación de seguridad.

  - **Autenticación.** En todas las capas se debe proveer mecanismos de autenticación, pero en esta capa los sistemas de seguridad serán más complejos, los sistemas se basan en cómputo en la nube y los ataques pueden venir de cualquier medio.

  - **Detección de Intrusos.** Generar alarmas cuando se detecte un intruso o actividad sospechosa.

  - **Evaluación de Riesgos.** Prevé los riesgos en la seguridad y los trata de corregir antes de que sucedan.

  - **Seguridad de los datos.** Debe ser implementada por medio de cifrado y firewalls.

A continuación, se muestra un resumen de las capas y funciones de seguridad de cada una (Fig. 2.8).

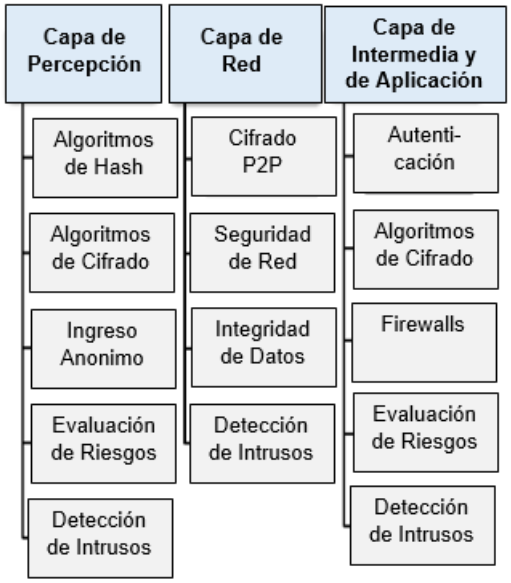


Fig. 2.8 Retos de seguridad en cada capa de IoT

En generan se deben tener las siguientes consideraciones de seguridad en los sistemas IoT.

- La mayoría de ataques ocurren en las señales físicas, y ocurren durante el procesamiento de datos y la toma de decisiones.
- Un gran porcentaje de nodos IoT operan en modo pasivo con baterías, por lo que soportan hardware con muy poco gasto de energía y con una propuesta ultra compacta de seguridad.
- Se debe coordinar la seguridad entre los centros de datos y los nodos IoT, una propuesta es dar a cada nodo un identificador único y rastreable en todo momento.
- Garantizar que los datos sean íntegros, asegurando a los usuarios que los datos que son recolectados por cierto dispositivo IoT son verdaderos.

2.1.3 Aplicaciones

Esta tecnología aún está en fase de desarrollo, sin embargo, varias aplicaciones ya son una realidad hoy en día.

- **Ubicación de activos.** Las empresas pueden saber la ubicación en tiempo real de sus unidades y piezas de inventario, como automóviles, maquinarias, etc.
- **Servicios de trasporte.** Algunas empresas implementan una variación del punto anterior, para saber la ruta y ubicación en tiempo real de sus unidades, pero además incorporan funciones para monitorear el funcionamiento de la unidad.
- **Red eléctrica.** Da un seguimiento y control a la red eléctrica de empresas y particulares.
- **Automatización industrial.** Control de procesos automatizados dentro una fábrica, desde un centro de control a través de internet.
- **Agricultura.** Con sistemas que reporten las condiciones de los campos de cultivo en todo momento se puede tener mayor rendimiento de las cosechas.
- **Medicina.** Monitorear pacientes de forma remota y en tiempo real.
- **Casas inteligentes.** Los sistemas domóticos ahora pueden ser controlados de forma remota a través de internet.
- **Ciudades inteligentes.** Sistemas que monitorean el tráfico, la contaminación, la delincuencia, y otros problemas de las ciudades.

Si bien es cierto que se ofrecerá una mayor seguridad física de los objetos, la seguridad personal puede ser más vulnerable, pues la información que viajará en la red y se intercambiará entre los dispositivos será de carácter personal.

El control que se tendrá sobre los objetos será mayor, los usuarios tendrán acceso y control remoto de muchos objetos cotidianos, esto a cambio de la privacidad, pues grandes empresas y corporativos serán los encargados de mantener estas redes y tendrán acceso a los datos personales y de comportamiento de los usuarios.

Si todos y cada uno de los objetos tuviesen conexión a Internet se podría saber en cada momento dónde se encuentran y reducir prácticamente a cero la posibilidad de perderlo, pero al estar en constante conexión a la red también se sabría la ubicación de los usuarios.

Si no se lleva a cabo un buen tratamiento de la seguridad los atacantes tendrán un poder nunca antes visto, pues no solo tendrán acceso a datos, sino también se tendrán repercusiones directamente en el mundo físico, por ejemplo, si se toma control de un auto podrían causar un accidente con víctimas humanas involucradas.

2.2 Protocolos de mensajería instantánea

Los protocolos de mensajería instantánea permiten la comunicación en tiempo real entre dos o más usuarios conectados a internet, generalmente la comunicación es mediante texto, aunque algunas aplicaciones permiten la transmisión de video llamadas y archivos.

La mayoría de protocolos de mensajería instantánea, tienen una topología cliente-servidor, los clientes son los usuarios e inician sesión en el servidor, este se encarga de mantenerlos conectados y entregar los mensajes. Algunos ejemplos de protocolos que son o fueron populares se mencionan a continuación.

- Yahoo! Messenger
- Windows Live Messenger
- XMPP (Google Talk, Facebook Chat, WhatsApp, etc.)

Los dos primeros protocolos mencionados son propietarios, haciendo necesarias aplicaciones especiales para la intercomunicación entre usuarios de diferentes protocolos, pero con tecnologías como XMPP es posible conectar más de un servicio desde la misma cuenta.

Es importante mencionar que estos protocolos no implementan ningún tipo de seguridad por sí mismos, transmitiendo el texto plano y haciéndolo vulnerable a numerosos ataques cuando viaja por internet.

2.2.1 Arquitectura general

En un principio, cada grupo desarrollador proponía su arquitectura, haciendo muy complicada la comunicación entre usuarios de diferentes protocolos, pero con la popularidad que gano la mensajería instantánea, surgió una comisión reguladora denominada IMPPWG (*Instant Messaging and Presence Protocol Working Group*) que proponen un modelo genérico de arquitectura.

La arquitectura común de los sistemas de mensajería instantánea, se basa en un esquema cliente-servidor para evitar los problemas que representan los firewalls (Fig. 2.9), pero agregando los inconvenientes que supone una arquitectura de servicio centralizada, desviando ciertos servicios como la transmisión de ficheros o video-llamadas hacia una arquitectura P2P (Punto a Punto).

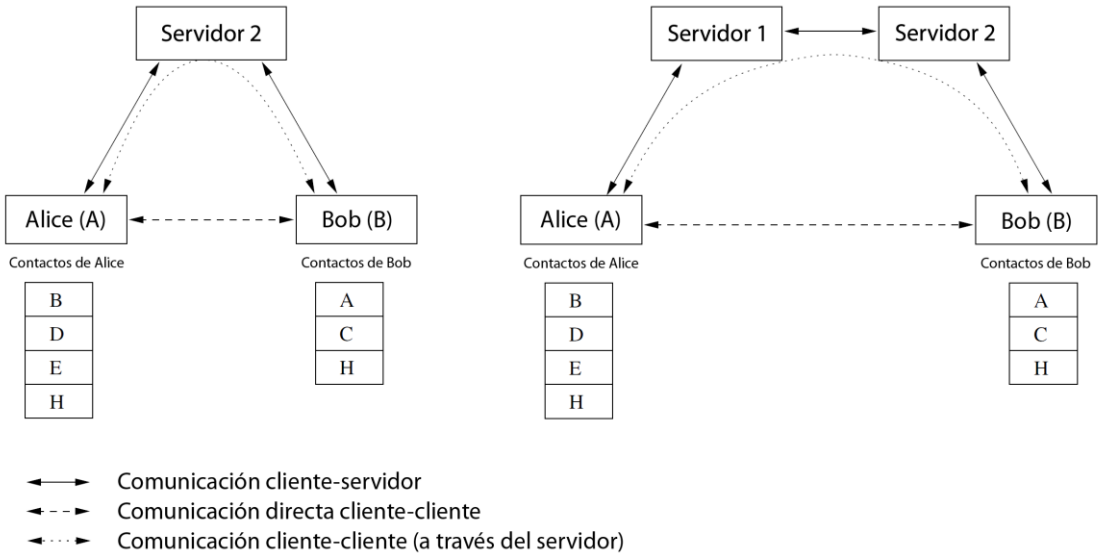


Fig. 2.9 Arquitectura clásica de los sistemas de mensajería instantánea

En esta arquitectura los usuarios inician sesión en el servidor y reciben actualizaciones del estado de sus contactos, cuando un usuario quiere establecer comunicación con otro, manda una petición al servidor y este, establece la comunicación entre ellos, en caso de transferencia de ficheros, el servidor manda al destinatario la petición de transferencia, que se realizará directamente entre los usuarios.

2.2.2 Protocolos más populares

A continuación, se analizarán algunos protocolos de mensajería instantánea, algunos podrían estar actualmente descontinuados, pero es necesaria su mención para comprender su funcionamiento.

2.2.2.1 Protocolo MSNP

El protocolo MSNP (*Mobile Status Notification Protocol)* es un protocolo propietario de Microsoft y su cliente oficial era *Windows Live Messenger*, esta aplicación fue descontinuada a principios del 2013 forzando la migración de los usuarios a Skype.

El protocolo implementa en su arquitectura tres tipos distintos de servidores "*Dispatch Server*" (DS), "*Notificación Server*" (NS) y "*Switchboard Server*" (SS). En primer lugar, el cliente se conecta con el DS que lo redireccionará a un NS que realiza el proceso de autenticación.

Una vez finalizado con éxito el proceso de autenticación, se entregará al cliente la lista y el estado de sus contactos, para evitar que la lista completa de contactos se descargue continuamente, únicamente se envían actualizaciones de estados y contactos nuevos. El servidor NS también se encarga de asignar a los clientes su servidor SS el cual mantendrá las comunicaciones entre usuarios, haciendo de intermedio. Los mensajes que se intercambian se envían en forma de comandos, cada comando tiene un índice que lo representa formado por tres letras mayúsculas o tres números si se trata de un error.

2.2.2.2 Protocolo Skype

Es un protocolo basado en la arquitectura P2P (*Peer-to-peer*), excepto para la función de autenticación de usuarios que está basada en un modelo cliente-servidor. En su arquitectura se tienen dos tipos de nodos, los clientes, que son los usuarios de Skype, y los supernodos que se encargan de gestionar el tráfico.

Sobre la seguridad es importante notar lo siguiente.

- Se cuenta con cifrado seguro y las medidas suficientes para garantizar la seguridad de las cuentas y que estas no sufran suplantación.
- Como se utiliza P2P, es muy fácil acceder a ambas direcciones ip en la comunicación.

- La comunicación se limita a usuarios dentro de la lista de contactos, para evitar ataques en la transmisión de ficheros.

No es un protocolo de código abierto y por lo tanto solo se distribuye en formato ejecutable y con medidas de seguridad. Es importante mencionar que Microsoft lleva a cabo un continuo monitoreo de las comunicaciones y analiza algunos mensajes.

2.2.2.3 Protocolo OSCAR

El protocolo OSCAR (*Open System for Communication in Real-time*) es el protocolo de mensajería instantánea propietario de AOL (*America Online*), no es de código abierto. La arquitectura detrás de este protocolo consta de varios servidores con distintas funciones, siendo los principales el *'Authorization Server'* (AS) y el *'Basic OSCAR Service Server'* (BOSS).

Cuando un usuario quiere iniciar sesión en el servicio, se contacta con el AS que lo autentica enviando un MD5 (*Message-Digest Algorithm 5*) formado por una clave de autenticación, recibida del servidor tras validar que el usuario y la contraseña existen. Una vez completada esta primera fase, se redirige al servidor BOSS que entregará al cliente información de la cuenta, la velocidad de transferencia y la lista de contactos.

Las comunicaciones con este protocolo son a través de diferentes canales, lo que permite comunicaciones paralelas sin la necesidad de conectarse a diferentes servidores.

2.2.2.4 Protocolo YMSG

El protocolo (*Yahoo! Messenger Protocol*) es propietario de grupo Yahoo, no es de código abierto, fue publicado en el siglo pasado en 1999, tiene una arquitectura cliente-servidor, pero a diferencia de otros protocolos los clientes se conectan con un servidor aleatorio.

El cliente por defecto es Yahoo! Messenger, no implementa cifrado de comunicaciones, para tener acceso al cifrado se necesita utilizar el cliente Yahoo! Bussiness Messenger.

2.2.2.5 Protocolo OTR

El protocolo OTR (*Off-the-record communication*) propone un modelo de comunicación diseñado para cumplir con los requisitos de autenticación y confidencialidad de los mensajes.

En este modelo los usuarios se autentican entre ellos utilizando sus claves públicas, hace uso del protocolo criptográfico *Diffie-Hellman* (DH) para establecer una clave de cifrado y una MAC (*Message Authentication Code*), cada mensaje que se envíe será firmado con una clave diferente.

La desventaja es que los usuarios de mensajería no acostumbran a tener llaves de firma digital y su creación y mantenimiento supondría un problema, además de la imposibilidad de intercambiar ficheros entre usuarios, realizar video-llamadas y otros servicios.

2.2.3 Análisis de seguridad

Los protocolos de mensajería históricamente no son reconocidos por darle gran importancia a la seguridad o a la identidad y privacidad de los usuarios, tampoco se incluye el cifrado de datos por defecto o si se incluye es muy básico, para registrarse a muchos estos servicios el único requisito es un correo electrónico.

2.2.3.1 Medidas de seguridad

Actualmente importantes corporativos se comunican a través de estos protocolos, los ciberataques y robo de identidad son cada vez más comunes, los protocolos se han visto obligados a garantizar dos puntos estratégicos de seguridad, en algunos casos el protocolo como tal no lo implementa, pero la aplicación que lo utiliza si lo hace.

- **Confidencialidad.** Los mensajes que se envían solo deben llegar a los destinatarios, no deben ser leídos por terceros.
- **Integridad.** Los mensajes enviados deben llegar al receptor sin que su contenido sufra alteraciones o modificaciones en el trayecto.

Además de las medidas básicas mencionadas anteriormente existen otras cuya implementación es altamente recomendable para poder garantizar un nivel mayor de seguridad.

- Una política de contraseñas que exija al menos una longitud de 8 caracteres y que de preferencia contenga diferentes tipos de símbolos.
- Conexiones seguras y cifradas para transmitir datos de autenticación y para comunicaciones entre usuarios.
- Almacenamiento de mensajes en el servidor, en caso de que los usuarios extravíen los dispositivos con su cuenta.
- Certificados de seguridad de clientes y servidores para garantizar la autenticidad, integridad y confidencialidad de los mensajes.
- Los equipos y software antivirus siempre deben estar actualizados para prevenir ataques o incompatibilidad.

2.2.3.2 Amenazas

Las amenazas más comunes a las que están expuestos los clientes de servicios de mensajería instantánea se mencionan a continuación.

- **Conexiones inseguras.** Una vez que el usuario se ha autenticado, la mayoría de protocolos no exige la autenticación de cada mensaje intercambiado, por lo que estos pueden ser capturados y leídos o se pueden enviar mensajes falsos.
- **Denegación de servicio.** Puede ser a un usuario en concreto inundándolo con mensajes autogenerados, o a un servidor haciendo que se pierda la conexión de los usuarios conectados a él.
- **Suplantación de identidad.** En el caso que un atacante tenga los datos de un usuario en concreto, puede obligarlo a desconectarse sin dar aviso al servidor de su desconexión, por lo que la sesión quedaría activa y el atacante puede hacerse con el control.

- **Suplantación de servidores.** Si un programa malicioso cambia la dirección del servidor a la que el usuario por defecto se conecta, este podría establecer conexión con un servidor que no es el oficial y estaría expuesto a otros ataques, para evitarlo es necesario solicitar el certificado de seguridad del servidor.

- **Acceso a la dirección IP del usuario.** Un atacante podría obtener la dirección ip del usuario si no se tomaron las medidas pertinentes para ocultarla, obteniendo la dirección ip, el atacante podría acceder a contenido público o mal configurado de la máquina del usuario.

- **Almacenamiento inseguro de información.** Algunos clientes almacenan en la máquina del usuario información como las credenciales de acceso o la lista de contactos o los mismos mensajes intercambiados, dando acceso a esta información a virus o programas maliciosos que estén en el PC del usuario.

- **Propagación de virus.** La posibilidad de transferir archivos entre los usuarios de mensajería abre la posibilidad de transferir también archivos maliciosos si el programa cliente o el antivirus del usuario no hacen el análisis del fichero recibido.

- **Spam.** El spam está presente en muchos otros servicios de comunicación, es el envío masivo de mensajes sin sentido, causando problemas de disponibilidad o hasta el colapso de la red. Muchos protocolos implementan algoritmos para evitar un colapso, como un límite de mensajes por segundo a cada cliente.

2.2.4 Protocolo XMPP

El protocolo XMPP (*Extensible Messaging and Presence Protocol*) es un protocolo de código abierto basado en XML (*Extensible Markup Language*) que se utiliza principalmente en servicios de mensajería instantánea, fue desarrollado en 1999 con el nombre de *Jabber*, en el 2002 fue aceptado por la IETF (*Internet Engineering Task Force*) con el nombre de XMPP.

Este protocolo tiene muchas ventajas contra sus similares y es por esto que ha ganado gran popularidad en los últimos años, algunas de estas ventajas son las siguientes.

- **XMPP está probado.** Más de 10 años de desarrollo han dado como resultado un protocolo estable que ha sido probado a gran escala por muchas empresas importantes, cuenta con millones de usuarios activos en la actualidad.
- **XMPP es seguro.** Provee las medidas necesarias para resistir a la mayoría de ataques, y su infraestructura descentralizada no tiene un punto de fallo único.
- **XMPP es descentralizado.** Implementa una infraestructura descentralizada con múltiples servidores a lo largo de internet, cada empresa o individuo puede tener su propio servidor de mensajería.
- **XMPP es extensible.** Gracias a su rápida entrega de mensajes XML, tiene muchas aplicaciones a lo largo de la web y se adapta a las nuevas tecnologías.
- **XMPP es escalable.** El modelo de transferencia implementado por XMPP resuelve la mayoría de los problemas de escalabilidad haciendo posible implementar servicios a una escala nunca entes vista.
- **XMPP es comunitario.** Gracias a que es un estándar de código abierto tiene una gran comunidad de desarrolladores que brindan apoyo en nuevos proyectos.

Por estas y otras razones cada día más desarrolladores y proveedores utilizan XMPP en la creación de nuevas aplicaciones y en escalar las ya existentes.

2.2.4.1 Características de XMPP

Este protocolo o variaciones del mismo son utilizadas ampliamente por servicios como Google Talk, Hangouts, Tuenti, Facebook y Whatsapp para ofrecer el servicio de mensajería instantánea, integra el uso de "Connection Managers" para utilizar el protocolo a través de conexiones HTTP

(*Hypertext Transfer Protocol o HTTP*) persistentes (*long-lived*), para integrarlo en web-chats. Por su gran versatilidad tiene muchos campos de aplicación, algunas de las características que lo hacen tan útil se mencionan a continuación.

- **Encriptado de canal.** Provee cifrado en la comunicación entre clientes y servidores o entre dos servidores.
- **Autenticación.** Es una pieza fundamental en el desarrollo de aplicaciones seguras, las entidades son autenticadas por un servidor antes de poder comunicarse.
- **Presencia.** Permite consultar el estado de otras entidades dentro de la lista de contactos.
- **Lista de contactos.** Permite almacenar una lista de contactos en el servidor con los que se puede establecer comunicaciones.
- **Mensajes uno a uno.** Permite el intercambio de mensajes entre dos entidades que pueden ser clientes, servidores, aplicaciones, etc.
- **Mensajes multi destino.** Permite el intercambio de mensajes entre múltiples participantes de una comunicación, las actualizaciones y mensajes llegaran a todos los destinos.
- **Notificaciones.** Permite él envió de un mensaje especial de notificación hacia una o múltiples entidades a la vez.
- **Información del servicio.** Permite consultar las características soportadas por las entidades.
- **Notificación de capacidades.** Informa las capacidades de las entidades
- **Formularios estructurados.** Permite intercambiar formularios estructurados con información de configuración entre las entidades.
- **Sesiones punto a punto.** Permite la negociación de sesiones multimedia entre dos entidades como chat de voz, chat de video, transferencia de archivos.

2.2.4.2 Código abierto y estándares abiertos

El protocolo XMPP por sí mismo no es de código abierto, pero si es un estándar abierto, lo que significa que puede ser utilizado en cualquier programa y por cualquier compañía.

XMPP únicamente propone los estándares y el formato de datos a utilizar en las comunicaciones, puede ser usado para cualquier fin, ya sea comercial, personal u otro.

2.2.4.3 Arquitectura

La infraestructura física de XMPP consiste en miles de computadoras ejecutando programas servidores como Ejabberd u Openfire para gestionar las comunicaciones y millones de clientes utilizando aplicaciones como Adium, Gajim, Pidgin, o Psi para establecer comunicaciones, todos comunicándose bajo el mismo estándar llamado XMPP.

La red XMPP utiliza una arquitectura cliente-servidor descentralizada (Fig. 2.10 , Fig. 2.11). Al ser de código abierto cada usuario puede implementar su propio servidor. Esta arquitectura permite la separación de tareas, una fácil escalabilidad y organización además no se tiene un único punto de fallo, si un servidor deja de funcionar otro puede reemplazarlo en segundos.

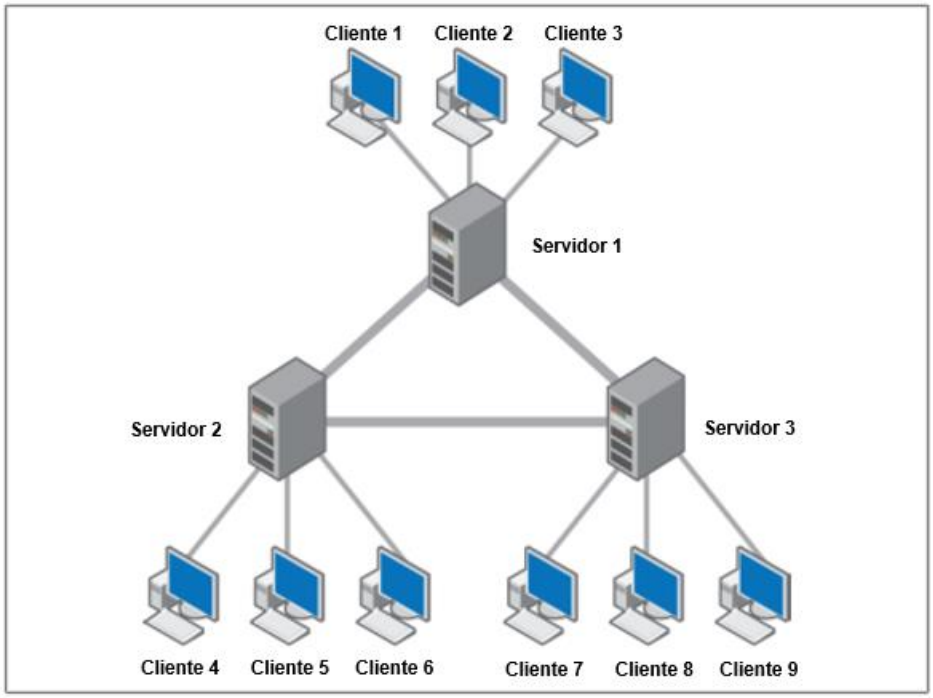


Fig. 2.10 Arquitectura clásica del protocolo XMPP

Cuando se envía un mensaje, el cliente se conecta a su servidor local que se conectara directamente con el servidor local del destinatario, sin saltos intermedios entre servidores.

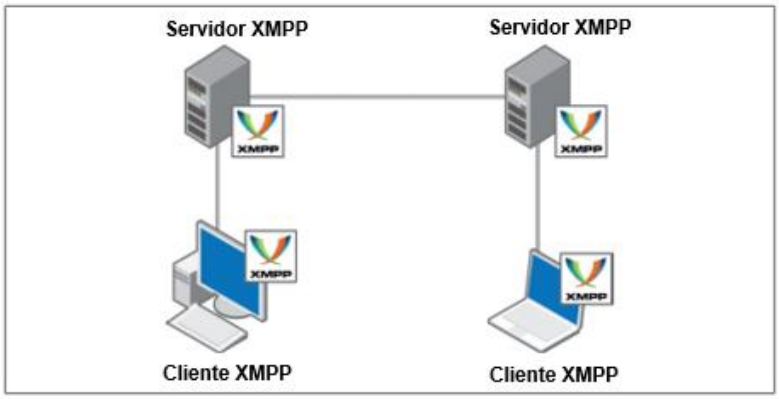


Fig. 2.11 Arquitectura de una comunicación entre dos clientes XMPP

Todos los clientes tienen asignado un identificador único llamado JID (Fig. 2.12), con una estructura parecida a la de un correo electrónico (usuario@dominio), este JID funciona como la identidad virtual del usuario dentro de la red XMPP.

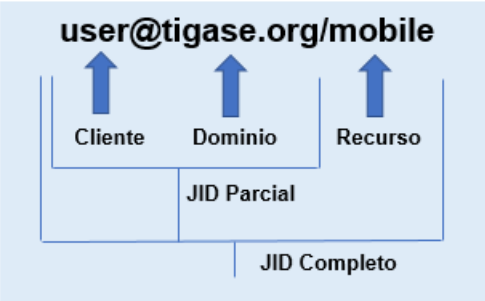


Fig. 2.12 Partes del JID o identificador único

Cuando el usuario se conecta al servidor se le asigna un identificador de recurso para esa conexión en particular, pensado los clientes que tienen múltiples sesiones abiertas con el mismo JID, una en su casa otra en su trabajo, etc. (Fig. 2.13).

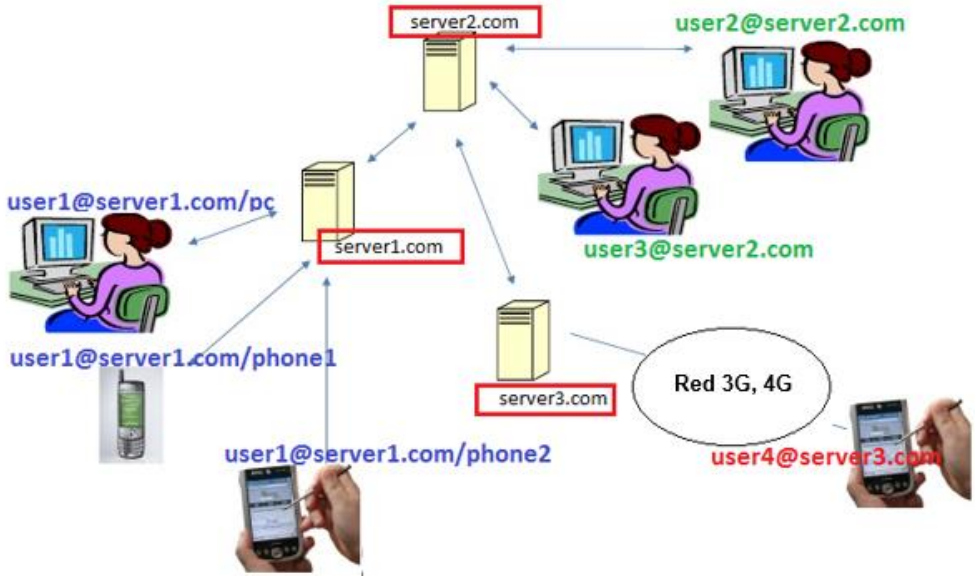


Fig. 2.13 Interacción entre diferentes usuarios XMPP

2.2.4.4 Primitivas

En esencia XMPP es una tecnología para enviar un flujo de mensajes XML, cuando el cliente inicia sesión en un servidor se negocia el flujo en ambas direcciones, una vez terminada la negociación se pueden intercambiar tres diferentes tipos de mensajes llamados "*stanzas*" que son la unidad de comunicación base de XMPP, el número de mensajes que se pueden enviar es ilimitado.

El intercambio de datos cliente-servidor es asíncrono asignando a cada petición un identificador con un valor secuencial pero no predecible. Cada tipo de stanza será tratada y direccionada diferente tanto por clientes como por servidores.

Tipo 1 (Message stanza)

Este tipo de stanza es el método "push" básico para hacer llegar información de un equipo a otro, existen diferentes subtipos según la aplicación que se dará.

- **Normal.** Son similares a mensajes de correo electrónico, pues únicamente son enviados y una respuesta no es esperada.
- **Chat.** Son enviados en tiempo real dentro de un flujo previamente negociado entre dos clientes.
- **Groupchat.** Enviados en tiempo real de un origen a múltiples destinos
- **Headline.** Usados para enviar alertas o notificaciones, una respuesta no es esperada.
- **Error.** Si ocurre un error en el envío de un mensaje, la entidad que lo detecta enviara un mensaje de error.

Este tipo de stanza también contiene información del origen y del destino además de un identificador para rastrar el mensaje. Una parte importante es la "*payload*" o cuerpo del mensaje que contiene el texto que será entregado al destino. En la (Fig. 2.14) podemos ver los campos que tiene este tipo de mensaje y en la (Fig. 2.15) un ejemplo.

```
msg = sleekxmpp.Message()
msg['type']   # Puede ser 'normal', 'chat', 'headline', 'error', 'groupchat'
msg['to']   #Este campo contiene el JID del dispositivo destino
msg['from']   #Este campo contiene el JID del dispositivo origen
msg['id']   #Contiene un string con un identificador
msg['body']   #Contiene texto plano que es el cuerpo del mensaje
```

Fig. 2.14 Campos de la stanza tipo Message

```
<message
        from='hag66@shakespeare.lit/pda'
        id='hysflv37'
        to='coven@chat.shakespeare.lit'
        type='groupchat'>
<body>Harpier cries: 'tis time, 'tis time.</body>
</message>
```

Fig. 2.15 Ejemplo de una stanza tipo Message recibida en una comunicación

Tipo 2 (Presence stanza)

Una de las características de las comunicaciones en tiempo real son las notificaciones de presencia, estas avisan a los usuarios cuando uno de sus contactos inicia sesión y además permiten a un usuario consultar cuales de sus contactos están en línea.

Las notificaciones de presencia solo serán enviadas a los usuarios agregados a la lista de contactos, un usuario puede elegir si enviar o no estas notificaciones o qué tipo de presencia enviar (en línea, ocupado, ausente, etc.). En la (Fig. 2.16) se muestra un ejemplo.

```
CS:   <presence   from='romeo@example.net/bar'
                  id='ps6t1fu3'
                  to='juliet@example.com'>
        <show>away</show>
      </presence>
```

Fig. 2.16 Ejemplo de una stanza tipo Presence recibida en una comunicación

Tipo 3 (IQ stanza)

El tipo IQ (Info/Query) es un tipo de mensaje que cuando un usuario lo envía espera una respuesta del destino, lo que asegura que el mensaje fue recibido o informa que se ha generado algún problema en la entrega, la diferencia de esta stanza y el tipo "message" es que aquí si se envía una respuesta y en el tipo "message" la respuesta no es obligatoria, dentro de esta stanza existen cuatro subtipos. En la (Fig. 2.17), se muestra un ejemplo.

- **get.** La entidad solicitante pide información.
- **set.** La entidad solicitante envía información.

- **result.** La entidad a la que se solicitó información con Get envía su respuesta
- **error.** La entidad a la que se solicitó información con Get o una entidad intermedia avisa que ocurrió un problema y la solicitud no se completó.

```
C:  <iq  from='juliet@example.com/balcony'
         id='hu2bac18'
         type='get'>
       <query xmlns='jabber:iq:roster'/>
     </iq>


S:  <iq  id='hu2bac18'
         to='juliet@example.com/balcony'
         type='result'>
       <query xmlns='jabber:iq:roster' ver='ver11'>
         <item  jid='romeo@example.net'
                name='Romeo'
                subscription='both'>
           <group>Friends</group>
         </item>
         <item  jid='mercutio@example.net'
                name='Mercutio'
                subscription='from'/>
         <item  jid='benvolio@example.net'
                name='Belvolio'
                subscription='both'/>
       </query>
     </iq>
```

Fig. 2.17 Ejemplo de una stanza tipo IQ recibida en una comunicación

2.2.4.5 Seguridad en XMPP

En este protocolo tanto la seguridad de las comunicaciones como la privacidad de los usuarios están presentes en su estructura base y cabe mencionar que en toda su historia nunca ha tenido problemas de seguridad importantes, toma medidas frente a las diferentes amenazas.

- **Seguridad de las comunicaciones.** Implementa un cifrado entre el cliente y el servidor.

- **Autenticación.** Evita la suplantación verificando la autenticidad de todos los mensajes recibidos.

- **Spam.** Implementa un máximo número de mensajes enviados en un tiempo determinado.

Aunque el protocolo implementa medidas de seguridad por defecto es importante que el cliente y los servidores utilizados también tomen las medidas pertinentes para garantizarla.

2.2.4.6 Aplicaciones

Por su completa arquitectura, este protocolo es utilizado en la construcción de muchas aplicaciones populares en internet.

- **Mensajería instantánea.** Se puede programar un servicio de mensajería clásico con las funciones más populares como chats uno a uno, lista de contacto y notificaciones de inicio se sesión, pero se pueden agregar otras características.

- **Chats de grupo.** Permite la creación de aplicaciones que ofrezcan los servicios de mensajería grupal, como conferencias virtuales para empresas, salones de clase virtuales y muchas otras.

- **Videojuegos.** Uniendo las dos aplicaciones antes mencionadas se pueden crear aplicaciones para que los usuarios de videojuegos estén en contacto mientras juegan y puedan crear grupos de juego con múltiples participantes.

- **Sistemas de control.** La combinación de mensajería y envió de formularios hace posible crear aplicaciones ligeras para control remoto de sistemas de mecánicos, robóticos y de otros tipos.

- **Middleware y computo en la nube.** Muchas compañías y grupos de investigación utilizan sistemas basados en XMPP para servicios y mantenimiento de la infraestructura de computo en la nube. Aunque la mayoría de estas aplicaciones utilizan otros servicios de mensajería más complejos, XMPP se está haciendo presente gracias a su simplicidad de implementación.

- **Actualización de datos.** Las redes sociales utilizan XMPP para la constante actualización de datos y notificaciones, pues ofrece la escalabilidad requerida.

- **Voz sobre ip.** La aplicación "*Google Talk*" fue lanzada en 2005 y se popularizo gracias al uso de XMPP para el chat de voz, desde entonces las extensiones de este protocolo para los servicios multimedia ha tomado popularidad.

- **Servicios de identidad.** Gracias a la existencia de identificadores y un robusto servicio de autenticación es posible utilizar XMPP para servicios de autenticación como *OpenID*.

2.3 Computadoras de placa única

Una computadora de placa única o SBC (Single-Board Computer), es una computadora completa en una sola placa, tiene una arquitectura con microprocesador o microcontrolador, memoria RAM (*Random Access Memory*), memoria ROM (*Read-only memory*) y entradas/salidas todo en la misma placa, en la (Fig. 2.18) se muestra un ejemplo de SBC, la JaguarBoard.

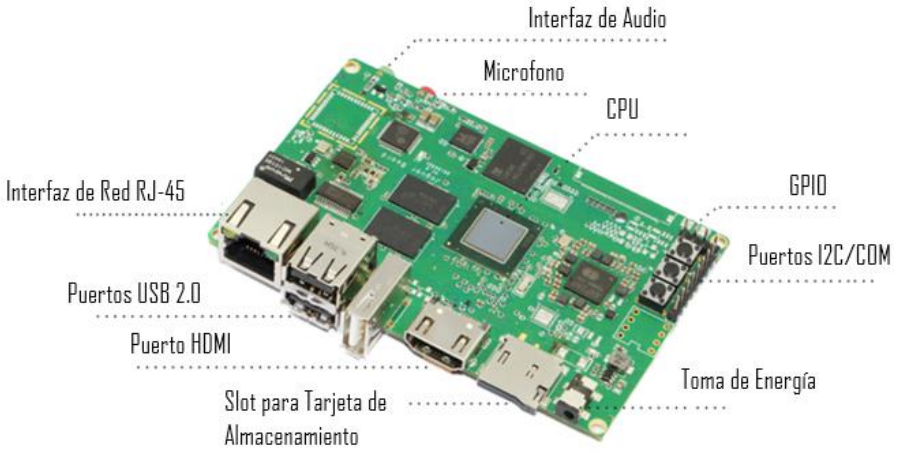


Fig. 2.18 Partes de una SBC (JaguarBoard)

2.3.1 Aplicaciones

Este tipo de computadoras se utiliza sobretodo como controladores en sistemas embebidos que no demanden una gran potencia de procesamiento. Debido a la reducción de los componentes para colocarlos en la misma placa, la velocidad del procesador, tamaño de memoria RAM, espacio de almacenamiento, suelen ser también reducidos, no es común utilizarlas como computadoras personales ni como servidores, pues no tienen el rendimiento necesario.

Las SBC ofrecen excelentes prestaciones en sistemas de control o automatización y tienen muchas ventajas contra las computadoras de escritorio.

- **Tamaño reducido.** La mayoría no rebasa los pocos centímetros y son ideales para integrarlas en sistemas con espacio limitado.

- **Alta eficiencia.** Gastan muy poca energía y producen muy poco calor.
- **Fácil aislamiento.** Gracias a su reducido tamaño es muy sencillo proteger la placa de la computadora en condiciones ambientales adversas.
- **Bajo costo.** Muchas de estas placas tienen un costo accesible y ofrecen buenas prestaciones.

Pero también se tienen algunas desventajas al trabajar son estos sistemas.

- **Falta de escalabilidad.** A diferencia de las computadoras modulares a las que se les conectan componentes por separado eligiendo cada uno de ellos según los requerimientos, en una computadora modular esto es imposible, pues los componentes vienen soldados en la misma placa.
- **Difícil reparación.** Ya que no son modulares, si alguno de los componentes presenta fallos, remplazarlo es una tarea muy difícil y se opta por remplazar la placa completa.

2.3.2 Arquitectura ARM

La arquitectura ARM (*Advanced RISC Machines*) es una familia de arquitecturas basadas en RISC (*Reduced Instruction Set Computer*). Los procesadores RISC están diseñados con un pequeño número de instrucciones diferentes, lo que les permite operar a gran velocidad, realizando un gran número de instrucciones por segundo, pues cada instrucción realizada es muy sencilla, utilizando únicamente uno o dos ciclos de reloj, pueden ofrecer un rendimiento excepcional con solo una fracción de la energía que demanda un procesador CISC (*Complex instruction set computing*).

Los procesadores ARM tienen su mercado principalmente en pequeños aparatos electrónicos, como Smartphones, tabletas, sistemas embebidos, dispositivos IoT. Gracias a que tienen pocas instrucciones se componen de menos transistores y circuitería, dando como resultado chips más pequeños y con un bajo consumo de energía. La arquitectura ARM tiene licencia, el

negocio principal de *ARM Holdings* es la venta de la arquitectura no fabricar procesadores, pues estos son fabricados por otras compañías.

2.3.3 Raspberry Pi

Es una SBC de bajo coste y de tamaño reducido, desarrollada en el Reino Unido por la Fundación Raspberry PI en la Universidad de Cambridge en 2011, con un propósito educacional, empezó su comercialización hasta el año 2012 y pese a su tamaño que es aproximadamente el de una tarjeta de crédito permite realizar grandes cosas.

Existen varias versiones de esta placa, en cada versión se han ido actualizando e incorporando componentes más modernos, pero todas comparten algunas características.

- Un Chipset Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz.
- Un procesador gráfico (GPU) VideoCore IV
- Un módulo de 512 MB de memoria RAM
- Un conector de RJ45 conectado a un integrado lan9512 -jzx de SMSC que proporciona conectividad a 10/100 Mbps
- 2 buses USB 2.0
- Salida analógica de audio estéreo por Jack de 3.5 mm.
- Salida digital de video y audio HDMI
- Salida analógica de video RCA
- Pines de entrada y salida de propósito general
- Conector de alimentación microUSB
- Lector de tarjetas SD
- Conectividad Wi-Fi

- Conectividad Bluetooth

2.3.3.1 Historia

Nació en 2006 como un proyecto de la Universidad de Cambridge con la misión de fomentar la enseñanza de las ciencias de la computación, la idea base era conseguir ordenadores portables y muy baratos, Aunque los primeros diseños de Raspberry se basaban en un microcontrolador de Atmel, el ATmega644, se cambió por un microprocesador basado en ARM para los primeros prototipos.

En agosto de 2011, se fabricaron cincuenta placas del Modelo Alpha (Modelo A). En diciembre de 2011, 25 placas del modelo Beta (Modelo B) fueron ensambladas y probadas. Durante la primera semana de diciembre de 2011, se pusieron a subasta diez placas en eBay con un gran éxito.

El primer lote de 10.000 placas se fabricó en Taiwán China, para abaratar los costes de producción y acortar el plazo de entrega del producto. Las primeras ventas comenzaron el 29 de febrero de 2012 (Modelo B). En los seis meses siguientes llegarían a vender 500.000 unidades.

El 6 de septiembre se anunció que se llevaría la producción de placas al Reino Unido, a una fábrica de Sony y que en ella se producirían 30.000 unidades cada mes.

Actualmente existen 2 modelos diferentes de Raspberry Pi. El primero, el modelo A, se diferencia del modelo B, en que tiene un solo tiene un puerto USB, carece de controlador Ethernet, tiene 256MB de RAM por los 512MB del otro modelo y cuesta menos que el modelo B. A pesar que el Modelo A no tiene un puerto RJ45, se puede conectar a una red usando un adaptador USB-Ethernet. En la (Fig. 2.19) se pueden ver los diferentes modelos de esta placa.

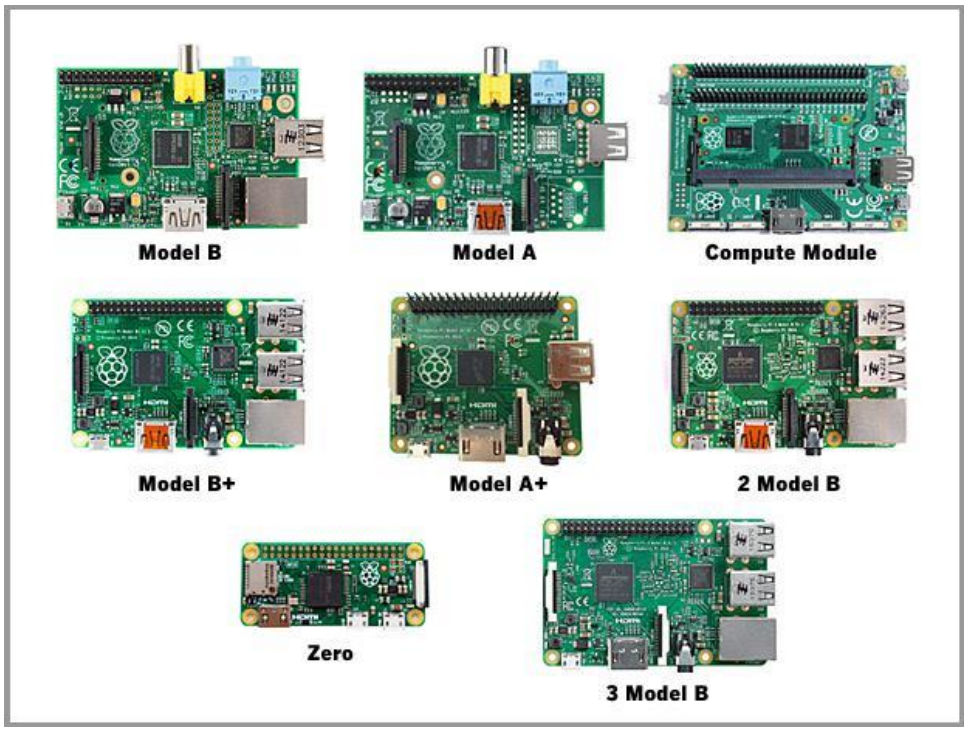


Fig. 2.19 Familia de placas de desarrollo Raspberry Pi.

2.3.3.2 Componentes

En la Raspberry Pi mostrada en la (Fig. 2.20) se muestran todos los componentes de la placa, que son prácticamente los mismos que necesita un computador de escritorio.

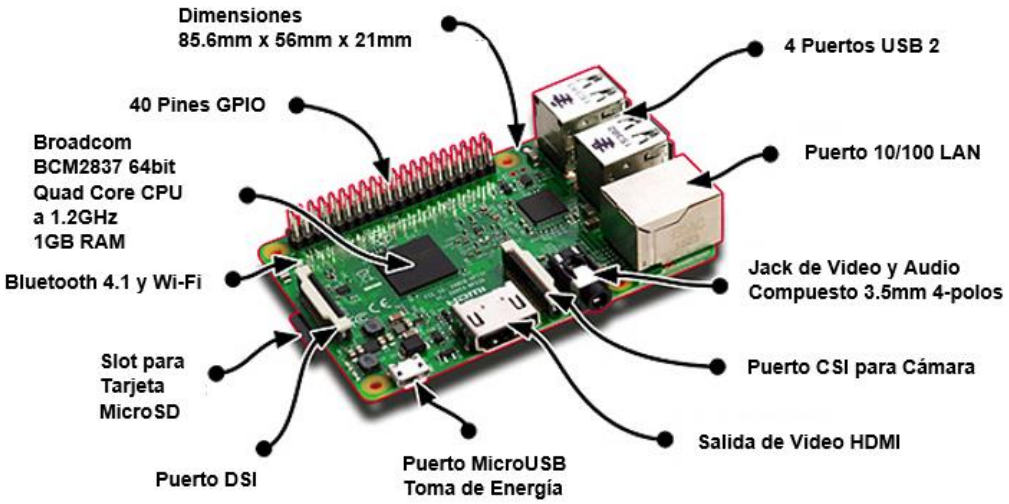


Fig. 2.20 Componentes de la placa de desarrollo Raspberry Pi 3B.

CPU

Como unidad de procesamiento en la placa, se tiene un ARM de sexta generación, que funciona a 700Mhz de base con una frecuencia turbo de 1Ghz

GPU, video y audio

Si bien la unidad de procesamiento grafica es bastante básica y no soportara una gran demanda de video, es suficiente para el propósito de la placa, es una Dual Core VideoCore IV Multimedia, capaz de reproducir Blu-ray, tiene un núcleo 3D con soporte para las librerías OpenGL ES2.0 y OpenVG.

Para la salida de video, la Raspberry posee un Conector RCA, un conector HDMI (*High Definición Multimedia Interface*) y una Interfaz DSI para paneles LCD. La mejor calidad de imagen se obtiene usando el conector HDMI que proporciona una conexión digital para mostrar imágenes en monitores de PC o en televisores de alta definición. Al utilizar el puerto HDMI, la Rasberry Pi puede desplegar imágenes con resolución de 1920×1080 (Full HD).

El tercer tipo de conector de video que tiene la Raspberry es un puerto DSI (*Display Serial Interface*), que se puede utilizar para conectar una pantalla muy parecida a la encontrada en celulares y tabletas.

Para el audio se cuenta con un Jack de 3,5mm, además del HDMI, que si se está usando transporta ambas señales, la de video y la de audio. Si el display no tiene entrada HDMI se tendría que utilizar la salida de audio por separado.

RAM

El modelo B tiene una memoria de 512MB SDRAM en un único módulo, a una frecuencia de 400Mhz en modo normal y 600Mhz en modo turbo. El modelo A tiene menor cantidad de memoria RAM disponible.

Memoria no Volátil

Como la Raspberry no cuenta con un disco duro tradicional, se necesita instalar una tarjeta microSD de buena calidad y mínimo de 8GB en su lector

que se encuentra en la parte inferior de la placa, desde esta tarjeta arrancara el sistema operativo y almacenara los programas y archivos.

Entradas y salidas de propósito general

Esta placa cuenta con puertos USB de propósito general, se pueden utilizar para conectar teclado, mouse, memoria USB o cualquier otro dispositivo soportado, el modelo B tiene integrados dos puertos mientras que el modelo A solo uno.

Una de las características más importantes de la Raspberry y que la hace muy útil para aplicaciones electrónicas son sus puertos GPIO (*General Purpose Input/Output*) que son digitales y tienen diferentes interfaces de comunicaciones dependiendo el modelo, estos puertos pueden ser activados o desactivados en cualquier momento por el usuario. Dependiendo del modelo será el número de pines GPIO disponibles, es importante revisar la hoja de datos del modelo antes de intentar cualquier conexión.

Conectividad

Se tienen varias opciones de conectividad, la más importante es un conector RJ-45 conectado a un integrado LAN9512 -JZX que proporciona 10/100Mbps de velocidad, se puede conectar directamente a un Router o a una PC, el protocolo DHCP (*Dynamic Host Configuration Protocol*) está activado por defecto, pero también se puede asignar una dirección ip estática desde la configuración de red.

Los modelos más recientes de la placa, tienen integrado un módulo Wi-fi para poder realizar una conexión inalámbrica, aunque los que no cuenten con esta característica se puede conectar un adaptador inalámbrico en el puerto USB.

Otra característica que también está presente únicamente en las placas más recientes es un módulo Bluetooth.

Alimentación

Esta placa en todas sus versiones carece de botón de encendido o de reinicio, para apagar el modulo o reiniciarlo cuando no responda, se debe desconectar la fuente de alimentación, esto no es recomendable si no es un caso extremo, pues puede dañar severamente la tarjeta micro SD pudiendo dejarla inservible.

La alimentación de la placa será mediante una entrada microUSB estándar de 5V y mínimo 700mA, pero es recomendable que sea de 3A o más para soportar el consumo de la placa. No todos los cargadores de Smartphone funcionaran con la Raspberry, se debe revisar el voltaje y la corriente de salida antes de intentar usarlo.

2.3.3.3 Arquitectura y Sistema Operativo

El procesador de la Raspberry Pi está basado en la arquitectura ARM, esta arquitectura no es común en computadoras de escritorio, es más utilizada en Smartphones y sistemas embebidos pues ofrece un bajo consumo energético a diferencia de los microprocesadores diseñados para PC's de escritorio.

Como el procesador no comparte la misma arquitectura que un PC, no se puede utilizar el mismo sistema operativo ni los mismos programas, esto no es una desventaja pues el objetivo de las Raspberry no es competir con las PC de escritorio, muchos programas para PC tienen una versión más ligera y adaptada para Raspberry.

Para instalar un sistema operativo en la Raspberry Pi debe ser utilizada una tarjeta microSD, el proceso de instalación es muy sencillo y consta de los siguientes pasos.

- Descargar la imagen del sistema operativo.
- Grabar la imagen del sistema operativo en la tarjeta micro SD.
- Insertar la tarjeta en la Raspberry Pi y encender la tarjeta.

El sistema operativo más común para la Raspberry pi es Raspbian, un sistema operativo basado en la distribución de Linux conocida como Debian, pero optimizada para arquitecturas ARM, al ser un sistema operativo de código abierto cualquiera puede modificarlo, la versión oficial se puede descargar desde la página de Raspberry. Este no es el único sistema operativo adecuado para la Raspberry, sin embargo, es el más popular.

2.3.3.4 Python y Raspberry

Es el lenguaje más común para trabajar con la Raspberry Pi, algunos sistemas operativos lo tienen preinstalado, al ser un lenguaje de programación de alto nivel tiene una sintaxis sencilla. Es un lenguaje interpretado y su filosofía se basa en el código sencillo. Python es un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa, programación funcional, etc.

Es actualizado por la "*Python Software Foundation*" y posee una licencia de código abierto, denominada "*Python Software Foundation License*", equivalente a la Licencia pública general de GNU.

2.3.3.5 Modelo B 3+

Como fue descrito anteriormente, existen varios modelos de Raspberry Pi, cada uno con diferentes características, antes de comenzar un proyecto es importante saber los requerimientos del sistema, así como el presupuesto, el modelo B3 plus, es el último que ha salido al mercado.

El modelo B3+ o B3 plus es una versión mejorada del modelo B3, tiene mejores prestaciones que su predecesora prácticamente al mismo precio, las características de ambas placas se pueden ver comparadas en la (Tabla 2.2).

Características

	RASPBERRY PI 3 MODEL B	RASPBERRY PI 3 MODEL B+
PROCESADOR	Broadcom BCM2837, Cortex-A53 (ARMv8) 64-bit SoC	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC
FRECUENCIA DE RELOJ	1,2 GHz	1,4 GHz
GPU	VideoCore IV 400 MHz	VideoCore IV 400 MHz
MEMORIA	1GB LPDDR2 SDRAM	1GB LPDDR2 SDRAM
CONECTIVIDAD INALÁMBRICA	2.4GHz IEEE 802.11.b/g/n Bluetooth 4.1	2.4GHz / 5GHz IEEE 802.11.b/g/n/ac Bluetooth 4.2, BLE
CONECTIVIDAD DE RED	Fast Ethernet 10/100 Gbps	Gigabit Ethernet sobre USB 2.0 (300 Mbps de máximo teórico)
PUERTOS	GPIO 40 pines HDMI 4 x USB 2.0 CSI (Cámara Raspberry Pi) DSI (Pantalla táctil) Audio / Vídeo compuesto Micro SD Micro USB (Alimentación)	GPIO 40 pines HDMI 4 x USB 2.0 CSI (Cámara Raspberry Pi) DSI (Pantalla táctil) Audio / Vídeo compuesto Micro SD Micro USB (Alimentación) Power-over-Ethernet (PoE)
FECHA DE LANZAMIENTO	29/2/2016	14/3/2018
PRECIO	$800 - $1200 MXN	$1000 - $1400 MXN

Tabla 2.2 Comparación entre los modelos B y B+ de Raspberry.

Descripción de pines

El modelo Raspberry 3B+ tiene la siguiente distribución de pines (Fig. 2.21).

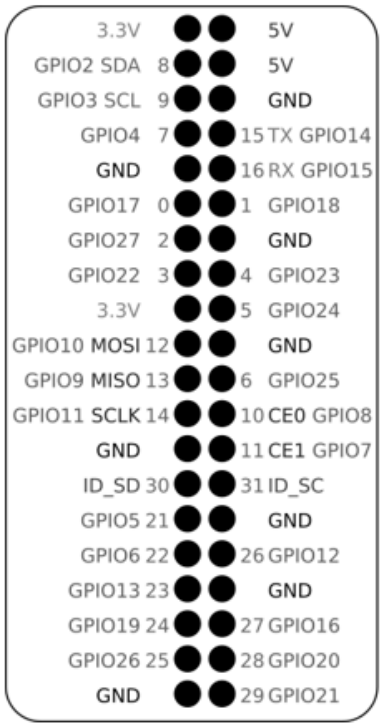


Fig. 2.21 Descripción de pines de la Raspberry pi 3B plus

Desempeño

A continuación (Fig. 2.22), se mostrarán los resultados de una serie de pruebas realizadas a diferentes sistemas de cómputo, se puede ver en donde se sitúa la Raspberry Pi 3, antes de sacar conclusiones se debe tener en cuenta el precio y objetivo de cada sistema.

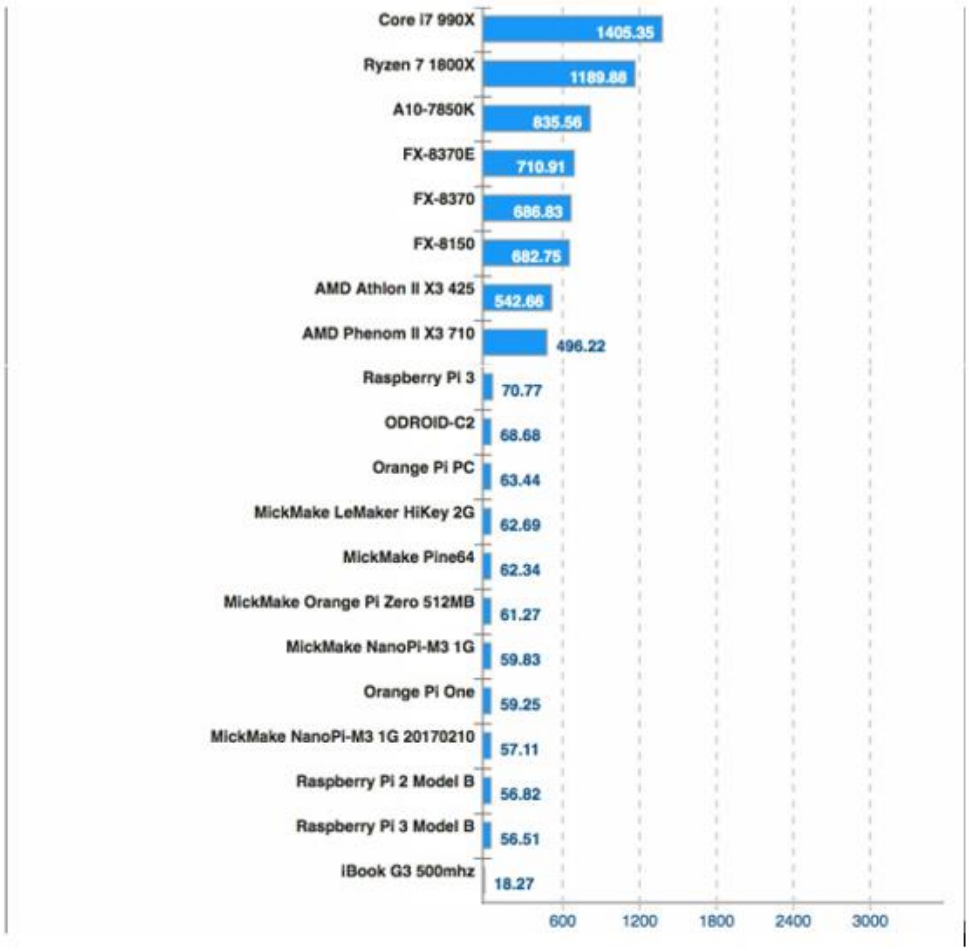


Fig. 2.22 Comparación del desempeño de la Raspberry pi y otros sistemas (MFLOPS)

Otra característica importante de la Raspberry Pi es su bajo consumo energético, con una disipación adecuada no se tendrá problemas con la temperatura, aunque esté trabajando las 24 horas del día, la mayor parte del calor como se verá en la (Fig. 2.23), proviene del microprocesador.

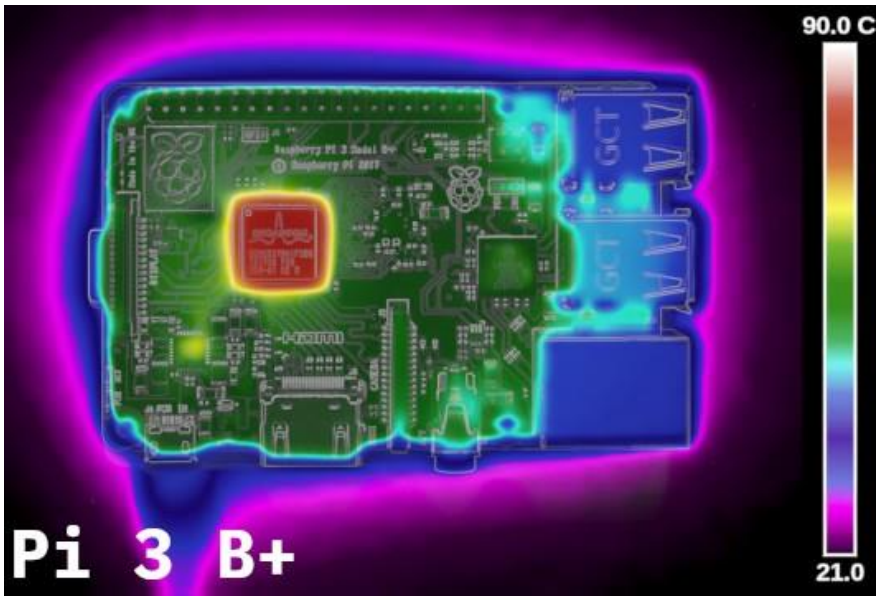


Fig. 2.23 Imagen térmica de la placa Raspberry pi 3B plus en funcionamiento

Se debe considerar la opción de instalar disipadores de cobre o hasta un ventilador (Fig. 2.24).

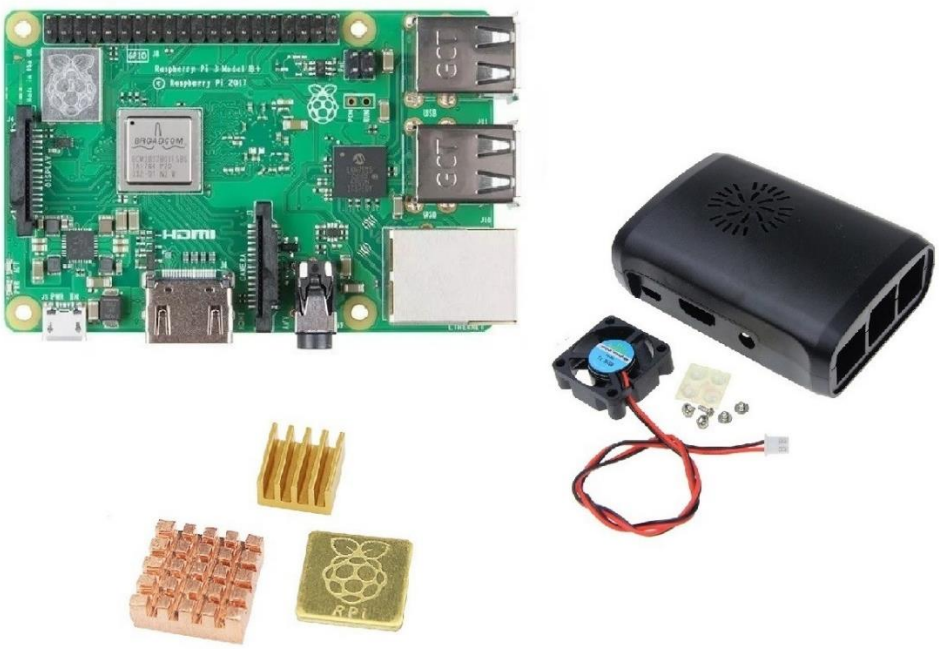
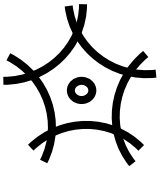


Fig. 2.24 Ejemplo de disipadores para la placa Raspberry pi 3B plus



3. DESARROLLO

3.1 Diseño del proyecto

En este capítulo se detallarán los elementos, el diseño y el funcionamiento de la red IoT propuesta en este trabajo.

3.1.1 Elementos

La red estará compuesta de cinco tipos de nodos (refiriéndose a nodos de red, no a nodos IoT que se mencionaran más adelante), cada uno encargado de tareas específicas y con características y capacidades diferentes, todos los nodos que formen parte de la red deben tener una conexión directa a internet, ya sea cableada o inalámbrica.

3.1.1.1 Nodos IoT simples.

- **Función.** Recolectar datos, pre procesarlos y transferirlos vía XMPP.
- **Características.** Contaran con los sensores apropiados para recolectar información. Para la comunicación tendrán instalado un cliente XMPP y un algoritmo para responder a las consultas.
- **Capacidades.** Pueden recolectar datos y modificar el estado de los actuadores que tengan conectados directamente. Únicamente responden a consultas provenientes de otros dispositivos. Ellos no pueden solicitar información a otros dispositivos.

3.1.1.2 Clientes

- **Función.** Visualizar la información recolectada y procesada por el sistema. Modificar el estado del sistema de forma remota, como los actuadores o parámetros de funcionamiento.
- **Características.** Para la comunicación tendrán instalado un cliente XMPP. Serán los usuarios finales del sistema, comúnmente personas desde sus celulares o computadoras.
- **Capacidades.** No tienen sensores ni actuadores conectados directamente. Podrán solicitar información a los nodos que la

recolectan o a los servidores de análisis. Pueden solicitar cambios en el estado y parámetros de funcionamiento del sistema.

3.1.1.3 Nodos IoT

- **Función.** Tendrán funciones de nodos IoT simples y de clientes a la vez, pues además de recolectar datos podrán solicitar información de otros nodos o servidores y podrán modificar el estado del sistema.

- **Características.** Contaran con los sensores apropiados para recolectar información. Para la comunicación tendrán instalado un cliente XMPP y un algoritmo para responder a las consultas. Tendrán una base de datos de los nodos con los que han establecido comunicación recientemente.

- **Capacidades.** Responden a consultas provenientes de otros dispositivos. Podrán solicitar información a otros dispositivos. Pueden recolectar datos y modificar el estado de los actuadores que tengan conectados directamente.

3.1.1.4 Servidor XMPP

- **Función.** Permitirán el inicio de sesión a los dispositivos

- **Características.** Estará instalado un programa servidor XMPP para gestionar la comunicación. Tendrán una base de datos de todos los nodos en el sistema y la función de cada uno.

- **Capacidades.** Gestionar las comunicaciones XMPP. Almacenar la ID (Identificador de inicio de sesión) de los usuarios y nodos en el sistema. Autenticación de los diferentes dispositivos que quieran formar parte de la red. Almacenar información de los sensores y actuadores en cada nodo

3.1.1.5 Servidor de análisis

- **Función.** Analizar y almacenar los datos recolectados y dar aviso en caso que el sistema esté funcionando fuera de los límites establecidos.

- **Características.** Para la comunicación tendrán instalado un cliente XMPP. Estará instalado un gestor de bases de datos para almacenar la información recolectada. La programación necesaria para el análisis de la información recolectada.

- **Capacidades.** No tienen sensores instalados ni recolectarán información directamente, pero podrán solicitarla a los nodos que la recolectan. Analizan los datos recolectados por los nodos de todo el sistema y dan aviso si se detectan anomalías en el funcionamiento.

3.1.2 Algoritmo de funcionamiento

La red IoT propuesta puede ser desde algo muy simple con pocos elementos conectados hasta algo muy complejo con cientos de elementos trabajando en conjunto. Si se agregan nuevos nodos y clientes a la red, se deben agregar también servidores XMPP para poder brindar conectividad a todos, la cantidad de nodos que soporta un servidor XMPP estará determinada por las características del servidor y de la conexión a internet. Si se requiere almacenar y analizar los datos recolectados, un servidor de análisis debe ser agregado a la red.

El algoritmo de funcionamiento es el siguiente.

- Para que la red este activa, los servidores XMPP y los servidores de análisis deben estar forzosamente en funcionamiento y en espera de peticiones.

- Cuando un nuevo nodo o cliente se incorpora a la red, iniciará sesión en el servidor XMPP con su ID y permanecerá en el estado "disponible" en todo momento.

- El orden en que inicien sesión los nodos y clientes no afecta el funcionamiento de la red, ni tampoco si estos se desconectan. El único inconveniente es que si se solicita un nodo que no está conectado los mensajes los recibirá hasta el momento que inicie sesión.

- Los nodos estarán todo el tiempo recolectando información de sus sensores y monitoreando el estado de sus actuadores, esta información será almacenada en el mismo nodo, o enviada al servidor de análisis en caso de que exista.

- Los nodos sabrán con certeza el numero especifico de cada actuador y sensor que tengan directamente conectados, así como la ID de sus vecinos y los actuadores y sensores que estos tengan.

- Los clientes podrán enviar peticiones a los nodos o servidores de análisis en cualquier momento, estas peticiones pueden ser para solicitar datos o para modificar el estado de un actuador.

- Si un cliente envía una petición y el nodo no tiene el sensor o actuador solicitado, buscara en su base de datos por información de quien lo tiene para reenviar la petición, si no encuentra coincidencias consultara al servidor por esta información, si el servidor tampoco tiene coincidencias se informará al cliente.

3.1.3 Niveles de complejidad

Ahora se analizará la red clasificándola por su nivel de complejidad.

- **Red simple.** Es el caso más sencillo, consta de pocos elementos y es el tipo de red que se implementara en este proyecto.

  Elementos. Un servidor XMPP para gestionar las comunicaciones, de 1 a 10 nodos IoT o nodos IoT simples, de 1 a 10 clientes.

  Características. Se puede usar un servidor XMPP gratuito, pues serán pocas las cuentas XMPP y el gasto que implica tener un servidor propio no lo compensa. Sera ideal para proyectos pequeños.

- **Red mediana.** En este nivel se puede considerar agregar un servidor de análisis para almacenar la información recolectada.

  Elementos. Un servidor XMPP para gestionar las comunicaciones, de 10 a 100 nodos IoT o nodos IoT simples, de 10 a 100 clientes, un servidor de análisis, opcional.

Características. Se puede usar un servidor XMPP gratuito, pero se debe considerar tener uno propio. Sera ideal para proyectos a mediana escala.

- **Red compleja.** Es forzosa la implementación de servidores de análisis para almacenar los datos recolectados por los nodos, además de servidores XMPP propios.

  Elementos. Uno o más servidores XMPP para gestionar las comunicaciones, desde 100 nodos IoT o nodos IoT simples, desde 100 clientes.

  Características. Se deben usar servidores XMPP propios. Se debe tener al menos un servidor de análisis. Sera ideal para proyectos a gran escala.

3.2 Diseño del modelo de prueba

Para las pruebas de la red propuesta se diseñó una topología que consiste en dos nodos IoT, dos clientes (Un PC con Windows y un Smartphone con Android), como servidor para las comunicaciones se utilizará un servidor XMPP público.

En la (Fig. 3.1) se muestra la topología lógica del sistema, como se puede apreciar los elementos pueden estar en cualquier ubicación, siempre y cuando tengan una conexión a internet.

- **Nodo 1.** Será una SBC Raspberry Pi 3b+. Tendrá directamente conectado un sensor de temperatura (Sensor 1) y tres actuadores (Actuador 1, 2, 3).

- **Nodo 2.** Será una SBC Raspberry Pi 3b+. Tendrá directamente conectado un sensor de temperatura (Sensor 2) y tres actuadores (Actuador 4, 5, 6).

- **Cliente.** Los dispositivos clientes serán una PC con Windows y un Smartphone con Android, cualquier versión de los sistemas

operativos que soporte las aplicaciones a utilizar es adecuada. Puede estar activo cualquiera de los dos clientes o los dos al mismo tiempo.

- **Servidor XMPP.** El servidor utilizado será uno público y gratuito, existen muchos servidores de este tipo en la web, para este proyecto se utilizará **jabber.at**

Los procesos de configuración de los dispositivos y la creación de cuentas en el servidor serán detallados más adelante.

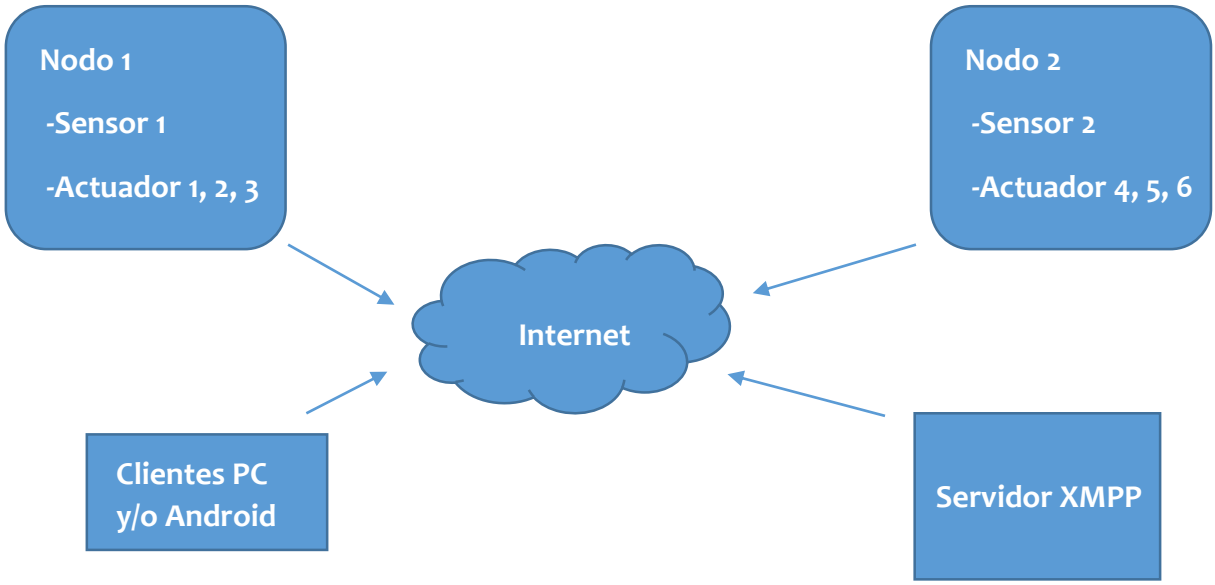


Fig. 3.1 Topología del sistema de prueba desarrollado

El funcionamiento de este sistema es el siguiente.

1. Los nodos mixtos recolectarán información en tiempo real con sus sensores y también tendrán varios actuadores conectados.

2. Cada nodo identificara con un número a cada sensor y actuador que tiene conectado, además tendrá información de los que tiene su nodo vecino. Todos los dispositivos tendrán almacenado su JID de XMPP y el de sus vecinos.

3. Al activarse el sistema, todos los dispositivos se autenticarán en el servidor XMPP para poder iniciar sesión y permanecerán en el estatus de disponible mientras estén activos.

4. El cliente tendrá el ID de XMPP o JID de ambos nodos, pero no tendrá información de los sensores y actuadores que están conectados a cada nodo.

5. El cliente podrá mandar peticiones en cualquier momento a cualquiera de los dos nodos solicitando información de los sensores o la activación de un actuador.

6. Si el nodo al que se mandó la petición tiene directamente conectado el sensor o actuador solicitado responderá a la petición.

7. En caso de que el nodo al que se manda la petición no tenga conectado el sensor o actuador solicitado, este buscara si el otro nodo del sistema lo tiene y le enviara la petición.

8. El cliente recibirá la información que solicito, pero no la información de que nodo tenía el elemento solicitado.

3.2.1 Descripción de componentes

- 2 Protoboards
- 2 Raspberry Pi 3b+
- 2 Sensores de temperatura DS18B20 tipo sonda
- 2 Resistencias de 4.7K Ohm
- 2 Resistencias de 330 Ohm
- 2 Leds RGB ánodo común
- 6 Leds de cualquier color
- 1 PC con Windows

- 1 Smartphone con Android

3.2.2 Configuraciones iniciales

Antes de poder programar, se deben hacer las conexiones del circuito y las configuraciones iniciales de los dispositivos. Como la Raspberry pi es más avanzada que otras placas como Arduino, es necesario instalar un sistema operativo y configurar las interfaces antes de comenzar a programar.

3.2.2.1 Configuración inicial de la Raspberry pi

La Raspberry Pi 3B+ es una minicomputadora y por lo tanto se le debe instalar un sistema operativo antes de poder utilizarla, para este proyecto se utilizará el sistema operativo Raspbian, que está basado en Debian y por lo tanto en Linux, los pasos a seguir para la instalación son los siguientes:

Descarga del sistema operativo

Desde el navegador web de cualquier pc, se accede al siguiente url:

https://www.raspberrypi.org/downloads/raspbian/

De las tres opciones disponibles, la utilizada en este proyecto (Fig. 3.2) es "Raspbian Stretch with desktop and recomended software". Esta versión tiene un escritorio muy parecido al de Windows y viene con las aplicaciones que se utilizaran ya preinstaladas.

Para realizar la descarga se da clic en la opción "Download ZIP", una vez descargado el archivo, se descomprime.

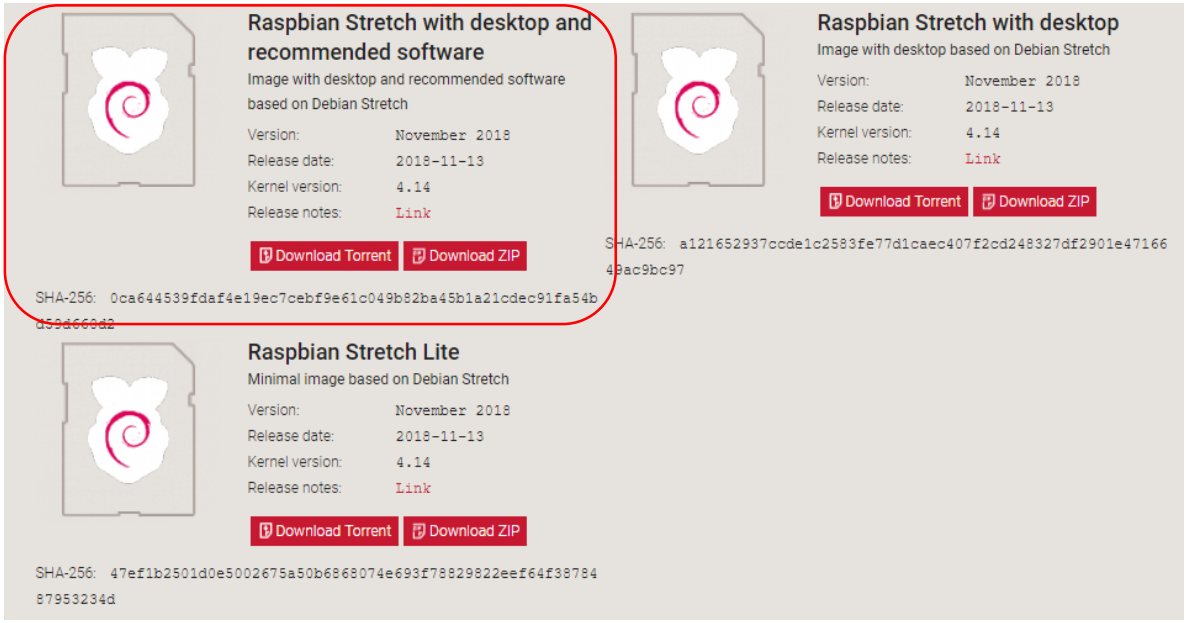


Fig. 3.2 Descarga del sistema operativo para la Raspberry

Preparación de la tarjeta micro SD

La tarjeta microSD a utilizar debe ser mayor a 8GB, es muy importante utilizar una tarjeta de buena calidad, de lo contrario no funcionara correctamente el sistema operativo. La tarjeta utilizada en este proyecto es la "Sandisk MicroSD HC UHS-1 16gb Clase 10".

Para grabar el sistema operativo en la tarjeta se necesitará el programa gratuito "Rufus", descargable desde la siguiente url:

https://rufus.akeo.ie/

Una vez instalado Rufus, se ejecutará desplegando la ventana (Fig. 3.3), en la opción "Dispositivo" se elige la tarjeta microSD a utilizar (Es muy importante cerciorarse de seleccionar el dispositivo correcto, de lo contrario podría dañarse severamente el PC), en la opción "Seleccionar", se elegirá la imagen de Raspbian que fue descargada y descomprimida anteriormente., se da clic en "Empezar", el proceso demorara varios minutos.

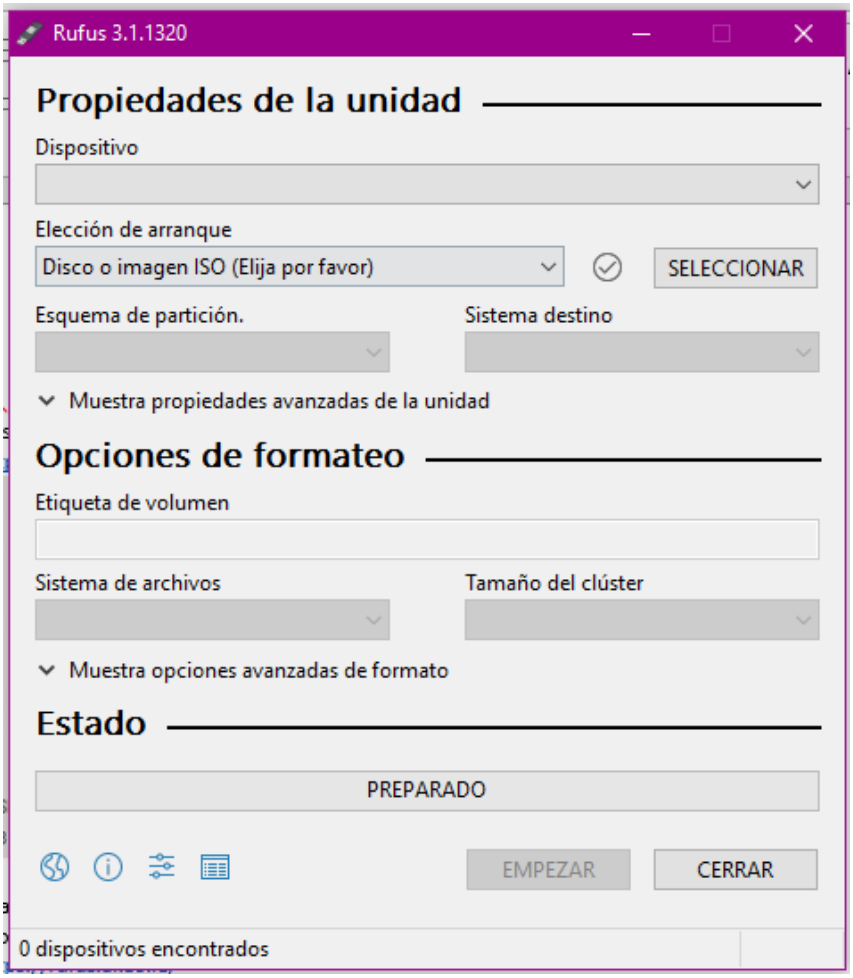


Fig. 3.3 Pantalla de inicio de Rufus

Instalación del Sistema Operativo

Para instalar el sistema operativo en la Raspberry, únicamente se introduce en el lector la tarjeta microSD grabada en el paso anterior. La primera vez que se enciende debe conectarse un monitor en cualquiera de las salidas de video, un teclado y un mouse en los puertos USB, pues el sistema operativo aún no está configurado para escritorio remoto.

Cuando el sistema solicite las credenciales de acceso, se utilizaran las que tiene por defecto (Usuario: pi Contraseña: raspberry).

La primera vez que se enciende demorara un poco, pues necesita configurar el sistema para su primer uso, en el transcurso de la instalación se puede

solicitar información (Idioma, tipo de teclado, etc.) que debe ser proporcionada para continuar el proceso.

Después de unos minutos el sistema operativo quedará configurado y en la pantalla se mostrará el escritorio de Raspbian (Fig. 3.4). Indicando que la instalación fue exitosa y que el sistema está listo para ser utilizado.

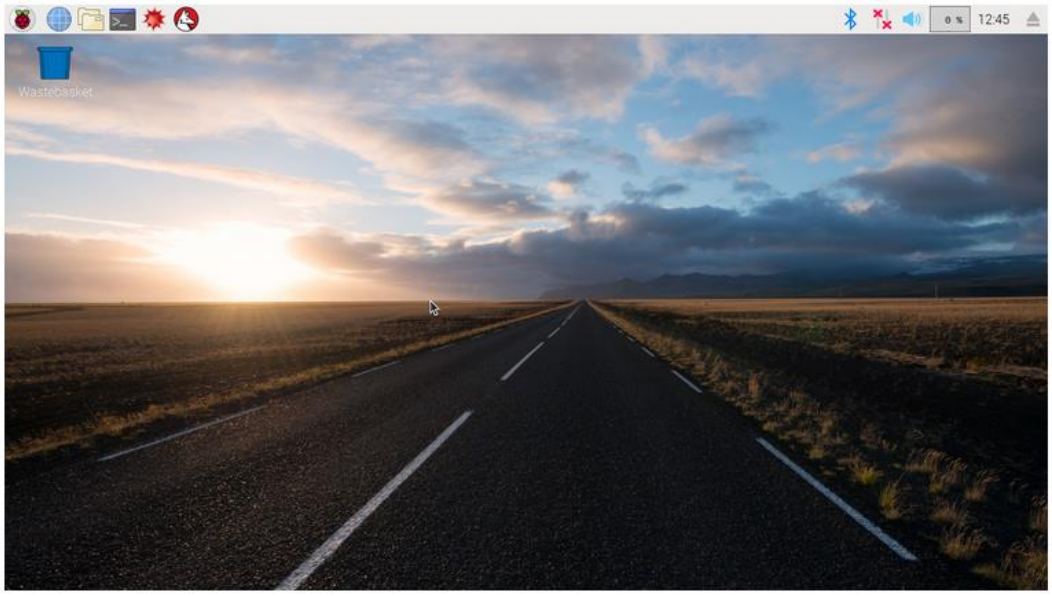


Fig. 3.4 Escritorio de Raspbian

Si no se tiene la Raspberry Pi conectada a internet mediante cable, se debe activar el Wi-fi, esta opción se encuentra en la parte superior derecha del escritorio (Fig. 3.4), su configuración es idéntica a cualquier otro dispositivo, solicitando el nombre y la contraseña de la red inalámbrica, los siguientes pasos y configuraciones necesitan conexión a internet.

Acceso remoto

En ocasiones es útil trabajar en la Raspberry sin tener monitor, teclado y mouse conectados directamente, afortunadamente se tienen varias opciones de conectividad remota.

- **VNC (Virtual Network Computing).** Este acceso virtual permite visualizar y trabajar en el escritorio del dispositivo desde cualquier otro ordenador con conexión a internet.

- **SSH (Secure SHell).** Permite acceder a la línea de comandos del Raspberry Pi con un programa de terminal como Putty a través de internet.

Existen otras opciones para una conexión remota con la Raspberry, pero en este proyecto serán utilizadas únicamente las dos anteriores.

VNC (Virtual Network Computing)

Es una aplicación grafica que permite controlar el escritorio de cualquier computador que este ejecutando "VNC server" de forma remota desde cualquier dispositivo en el que se tenga instalado "VNC viewer". Los comandos de teclado y los movimientos del mouse serán transmitidos por la conexión VNC.

El sistema operativo instalado a la Raspberry Pi en este proyecto, tiene por defecto VNC server y VNC viewer, pero deben ser activados y configurados manualmente para poder utilizarse. El proceso para habilitar VNC server es el siguiente.

- **Actualizar**

En una ventana de la línea de comandos de la Raspberry Pi se escriben las siguientes instrucciones.

#sudo apt-get update

#sudo apt-get install realvnc-vnc-server realvnc-vnc-viewer

- **Habilitar en la Raspberry**

Una vez actualizado VNC, Se procede a habilitarlo desde la línea de comandos.

#sudo raspi-config

- Interfaz opciones

- Seleccionar VNC -> Yes

- **Habilitar en el PC**

Se debe instalar el "VNC viewer" en el equipo desde el que será controlada la Raspberry, se descarga el programa para PC desde la siguiente url:

https://www.realvnc.com/es/connect/download/viewer/

Se debe seguir el proceso de instalación y en unos minutos el programa estará instalado y listo para establecer una conexión.

- **Conectar**

Para la conexión se necesita la dirección ip de la Raspberry Pi, se puede visualizar usando el siguiente comando.

#sudo ifconfig

Con la Raspberry Pi encendida, se ejecuta en la PC el programa "VNC viewer" como se muestra en la (Fig. 3.5), se escribe la dirección ip de la Raspberry Pi, se da clic en "Conectar" y "Aceptar" en caso que aparezca un mensaje de seguridad (Fig. 3.5).

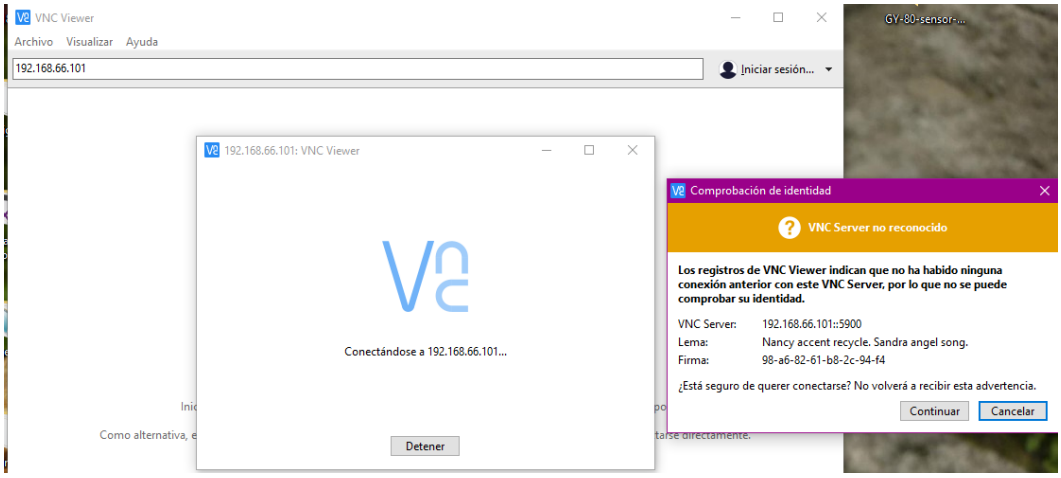


Fig. 3.5 Inicio de sesión en Raspbian mediante VNC viewer

Aparecerá una ventana como la mostrada en la (Fig. 3.6) solicitando el usuario y la contraseña de la Raspberry Pi (Usuario: pi Contraseña: raspberry).

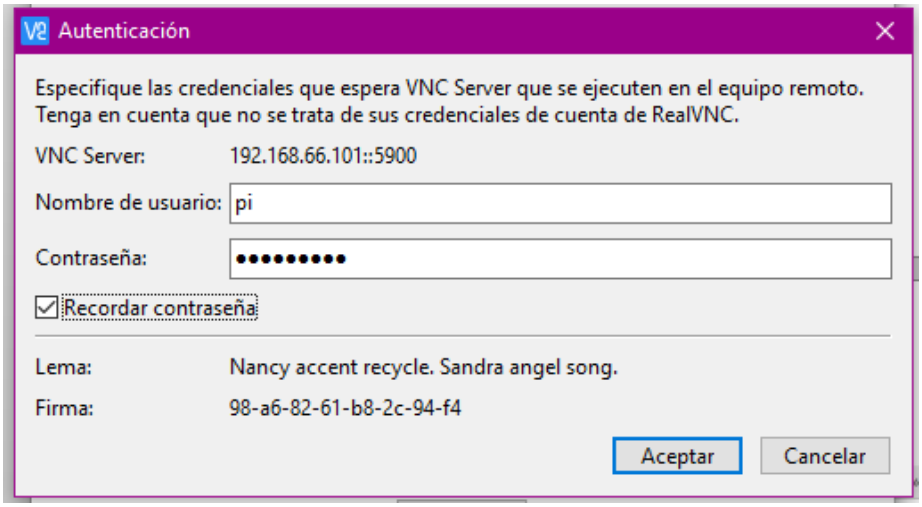


Fig. 3.6 Pantalla para escribir las credenciales en el inicio de sesión con VNC viewer

La conexión se habilitará y enseguida se podrá controlar el escritorio de la Raspberry Pi de forma remota (Fig. 3.7).

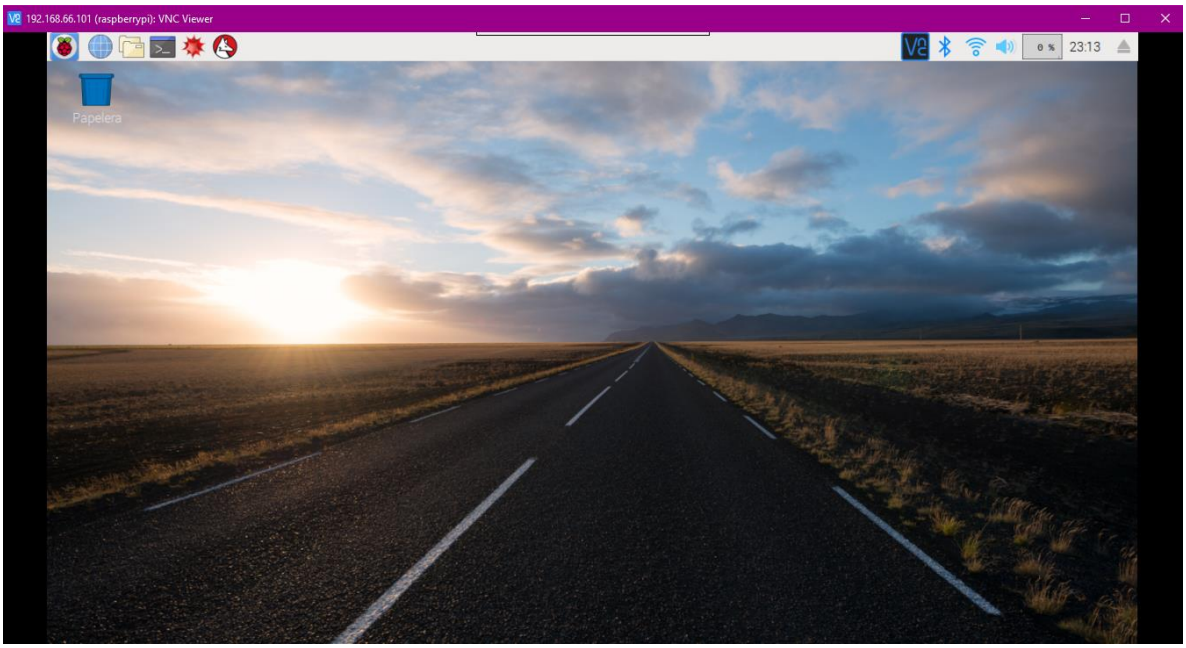


Fig. 3.7 Escritorio de Raspbian mostrado un una PC con VNC viewer

Para finalizar la conexión, se debe apagar la Raspberry Pi y cerrar el programa "VNC viewer".

SSH (Secure SHell)

Una conexión por SSH únicamente permite escribir comandos en la consola, no muestra la interfaz gráfica, pero es necesario tener este acceso de respaldo. En las versiones actuales de Raspbian el servidor SSH esta deshabilitado, así que debe ser activado manualmente, con los siguientes comandos.

#sudo raspi-config

- Navegar a la interfaz opciones

- Seleccionar SSH -> Yes

Para acceder se necesitará una aplicación cliente, en Windows puede utilizarse Putty, descargable de forma gratuita desde la siguiente url:

https://www.putty.org/

Se ejecuta el programa y se escribe la dirección ip de la Raspberry Pi (Fig. 3.8), esta se puede visualizar con el comando.

#sudo ifconfig

Se da clic en el botón "Open" y se abre la línea de comandos de la Raspberry Pi (Fig. 3.9).

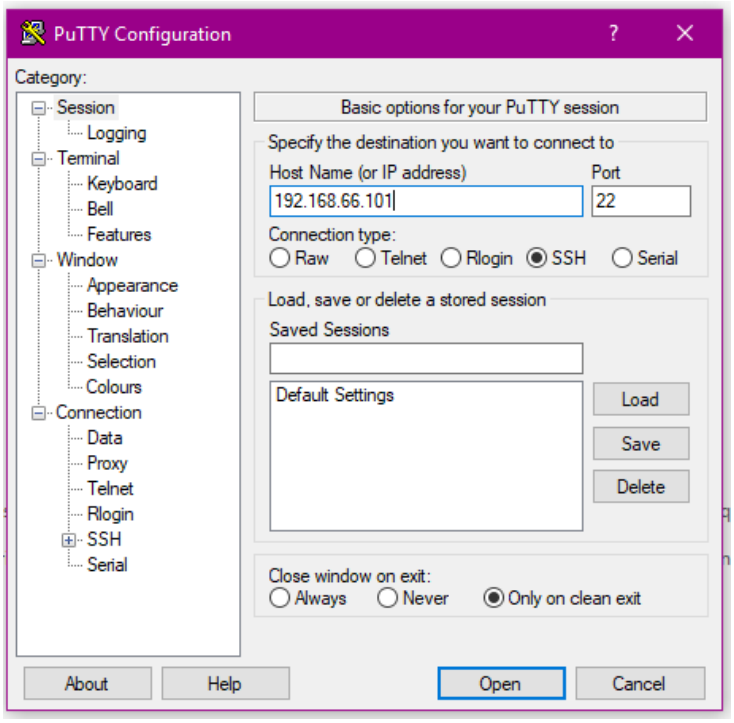


Fig. 3.8 Pantalla de inicio de Putty

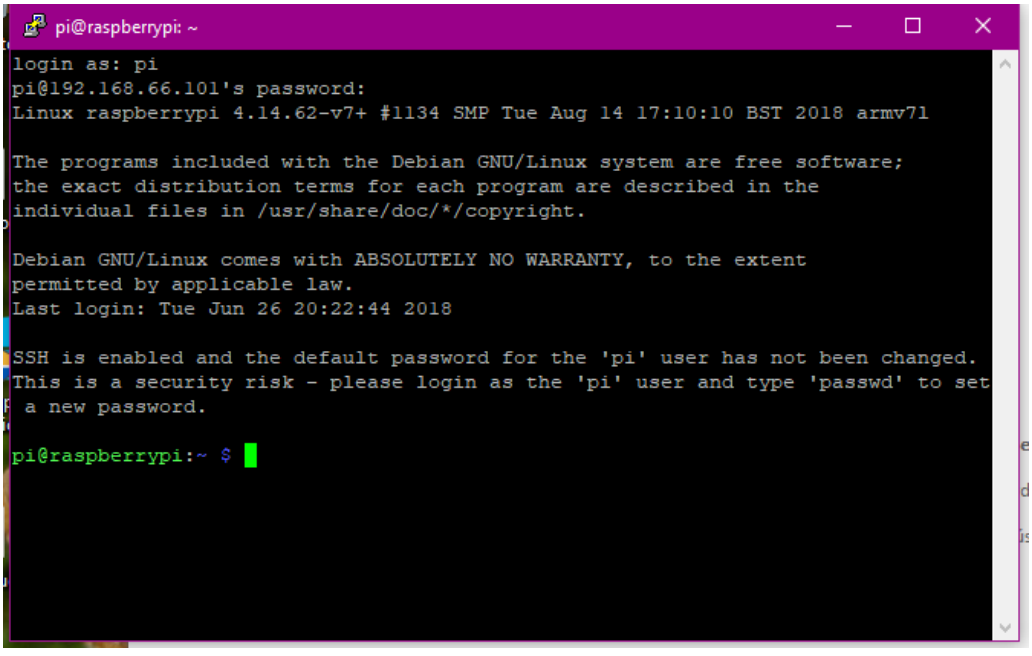


Fig. 3.9 Sesión remota en la línea de comandos con Putty

Para terminar la sesión se cierra la ventana de Putty.

3.2.2.2 Configuración de XMPP

Para utilizar el protocolo XMPP, se debe seguir el siguiente proceso.

Instalación de bibliotecas

Para trabajar con XMPP se deben descargar las bibliotecas que lo contienen. Estas bibliotecas no son creadas por la fundación XMPP sino por empresas y desarrolladores independientes.

En este proyecto se utilizará la biblioteca "SleekXMPP" que está programada en Python, para poder utilizarla, los pasos a seguir son los siguientes.

Instalación en Windows

Después de instalar correctamente la versión apropiada de Python en la PC, se escribe el siguiente comando en el símbolo del sistema.

```
>pip install sleekxmpp
```

Instalación en Raspberry

Para instalar esta biblioteca en la Raspberry Pi, se utiliza el siguiente comando.

```
#sudo pip install sleekxmpp
```

Ahora se tiene instalada la biblioteca con las funciones XMPP en ambos sistemas operativos.

Instalación de cliente XMPP

Se utilizará el cliente "Gajim", desde su página oficial se puede descargar el programa.

https://gajim.org/

Una vez descargado, se ejecuta el archivo y se siguen las instrucciones del instalador.

Registro de cuentas en el servidor

Para poder establecer comunicaciones se requiere de un servidor, en este proyecto se utilizará un servidor gratuito para registrar las cuentas, el servidor jabber.at cuya página oficial es la siguiente.

https://jabber.at/

En esta página se puede hacer el registro de las cuentas teniendo un correo electrónico para cada una, para este proyecto se registrarán tres cuentas con tres correos diferentes.

3.2.2.3 Sensor de Temperatura

Este sensor puede ser intercambiado por cualquier otro siempre y cuando se tenga una biblioteca en Python para poder utilizarlo. Si es usado otro sensor, se debe tener en cuenta que cumpla con las características mencionadas a continuación.

Características

El sensor de temperatura utilizado en este proyecto es el DS18B20 en empaque tipo sonda (Fig. 3.13). Con las siguientes características.

- Interfax de comunicaciones a un hilo 1-Wire
- No requiere componentes externos para su funcionamiento
- Puede ser alimentado desde 3.3V hasta 5.5V
- Rango de temperatura desde -55° hasta 125°
- Precisión, 0.5° en el rango de -10° a 85°
- Resolución programable desde 9 a 12 bits

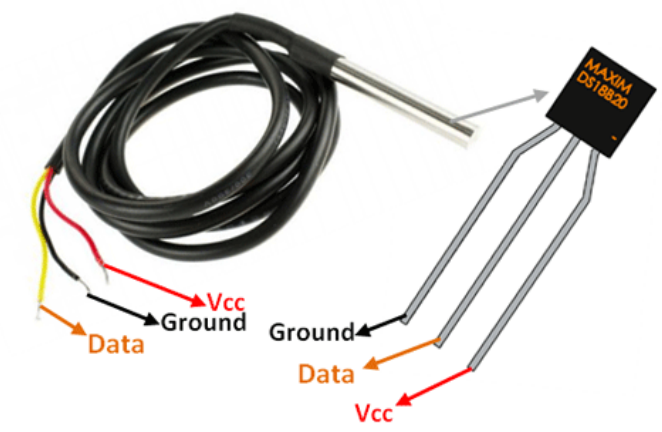


Fig. 3.10 Termómetro DS18B20

Es un sensor muy completo, su diagrama de bloques se puede ver en la (Fig. 3.14).

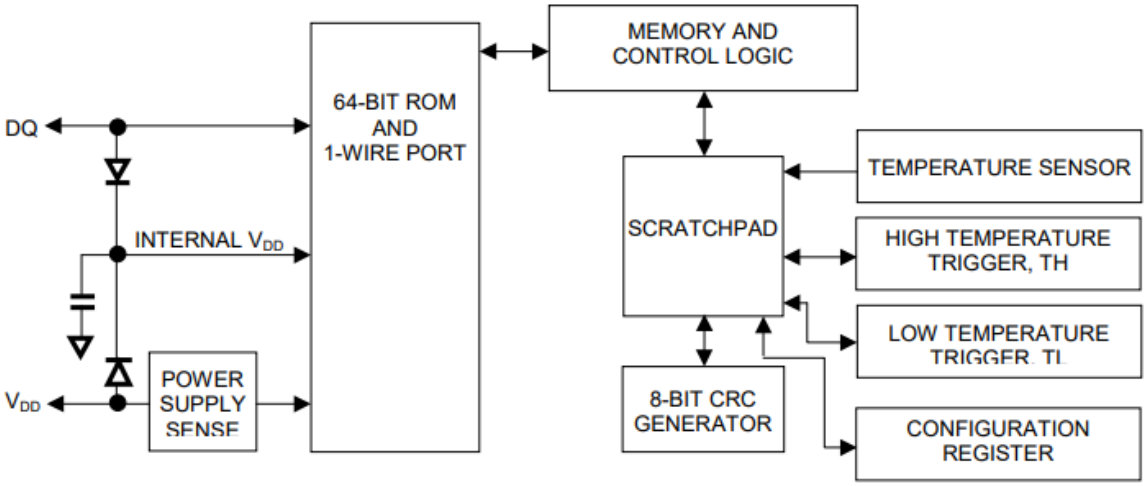


Fig. 3.11 Diagrama de bloques del termómetro DS18B20

Conexión y configuración

Con la Raspberry Pi apagada, se realizarán las siguientes conexiones: El sensor tiene tres cables de diferentes colores, el rojo es de voltaje, el amarillo es de datos y el negro es tierra, cada color va con correspondiente en el diagrama de la (Fig. 3.15).

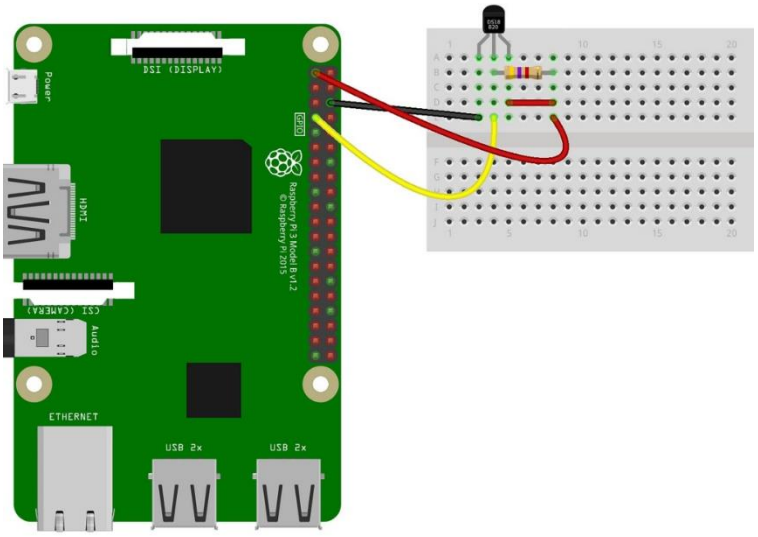


Fig. 3.12 Conexión del termómetro DS18B20

Se debe descargar la biblioteca con las funciones para comunicar el sensor con la Raspberry Pi, esta biblioteca se llama "w1thermsensor" y se descarga desde la línea de comandos de la Raspberry Pi, escribiendo la siguiente línea.

#sudo pip3 install w1thermsensor

Ejecutado el comando, comenzará la descarga e instalación de la biblioteca, una vez que aparezca el texto de la (Fig. 3.16) la instalación habrá concluido con éxito.

```
pi@datura:~ $ sudo pip3 install w1thermsensor
Collecting w1thermsensor
  Downloading https://files.pythonhosted.org/packages/3f/36/a10501cc5bcc138d9809
d368e627db1ca078e2e8bb3ca6b4ccadaaa0bc09/w1thermsensor-1.1.2-py2.py3-none-any.wh
l
Requirement already satisfied: click in /usr/lib/python3/dist-packages (from w1t
hermsensor)
Installing collected packages: w1thermsensor
Successfully installed w1thermsensor-1.1.2
pi@datura:~ $
```

Fig. 3.13 instalación de la librería w1thermsensor

Este sensor utiliza el protocolo de comunicación a un hilo (OneWire) y se debe indicar a la en que pin está conectado, para este procedimiento se abre el panel de configuración de la Raspberry Pi como se muestra en la (Fig. 3.17).

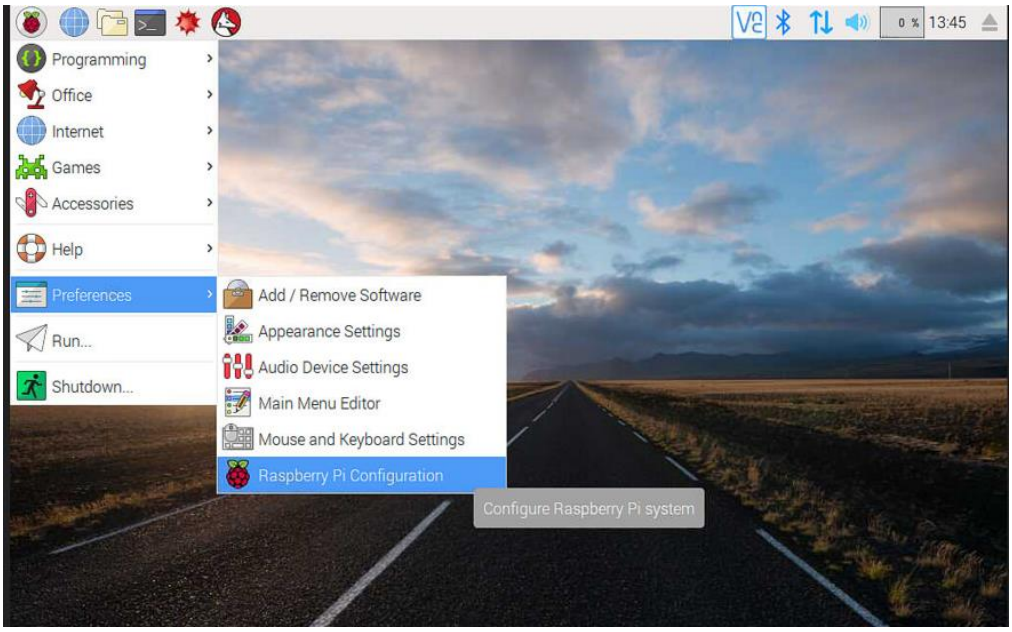


Fig. 3.14 Configuración del sistema Raspbian

Se desplegará la ventana de la (Fig.3.18), se selecciona la pestaña "Interfaces" y se habilita la interfaz "OneWire".

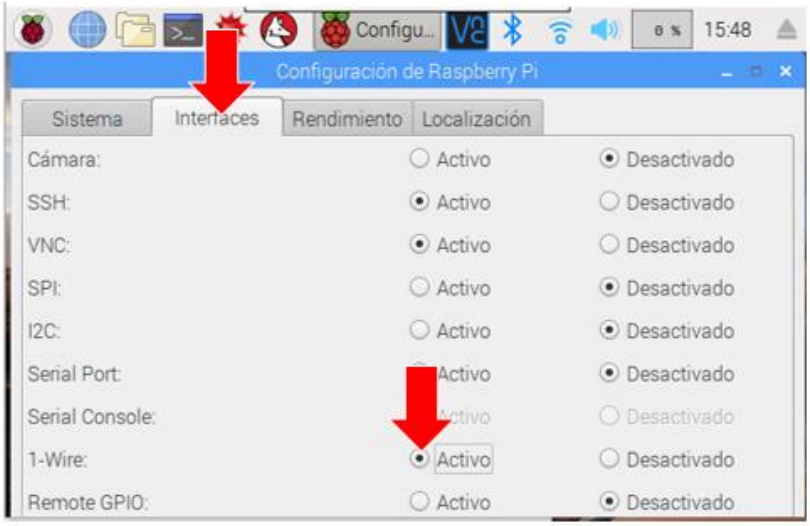


Fig. 3.15 Configuración de las interfaces en Raspbian

Una vez activada la interfaz, se da clic en "Aceptar" y aparecerá el cuadro de dialogo (Fig. 3.19), se reiniciará la Raspberry Pi después de dar clic en "Yes".

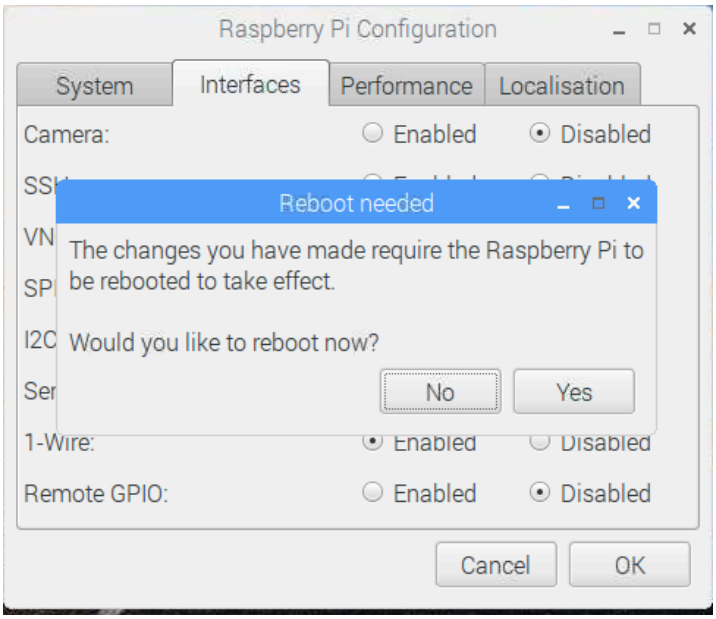


Fig. 3.16 Activación de la interfaz OneWire

3.2.2.4 Diodo LED

Los LED's son componentes básicos de cualquier sistema electrónico, indican el estado o los procesos que se están llevando a cabo. Para encender un LED con la Raspberry Pi, se utilizan las GPIO (Entrada/salida multipropósito), el circuito necesario se muestra en la (Fig. 3.20). Es importante apagar la Raspberry Pi cuando se modifican conexiones, de lo contrario se puede producir un corto accidentalmente.

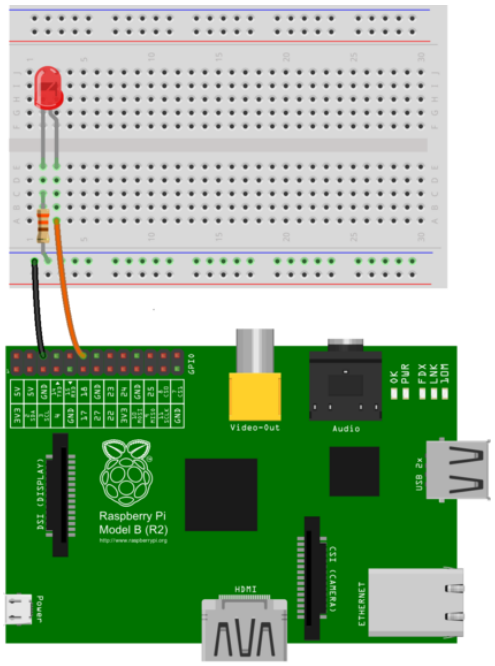


Fig. 3.17 Conexión de un LED

No es necesario descargar ninguna biblioteca ni realizar ninguna configuración pues ya están instaladas por defecto las necesarias.

3.2.3 Diseño y conexión del circuito

El diagrama del circuito a conectar para este proyecto es mostrado en la (Fig.3.21). Este se repetirá idéntico para ambos nodos.

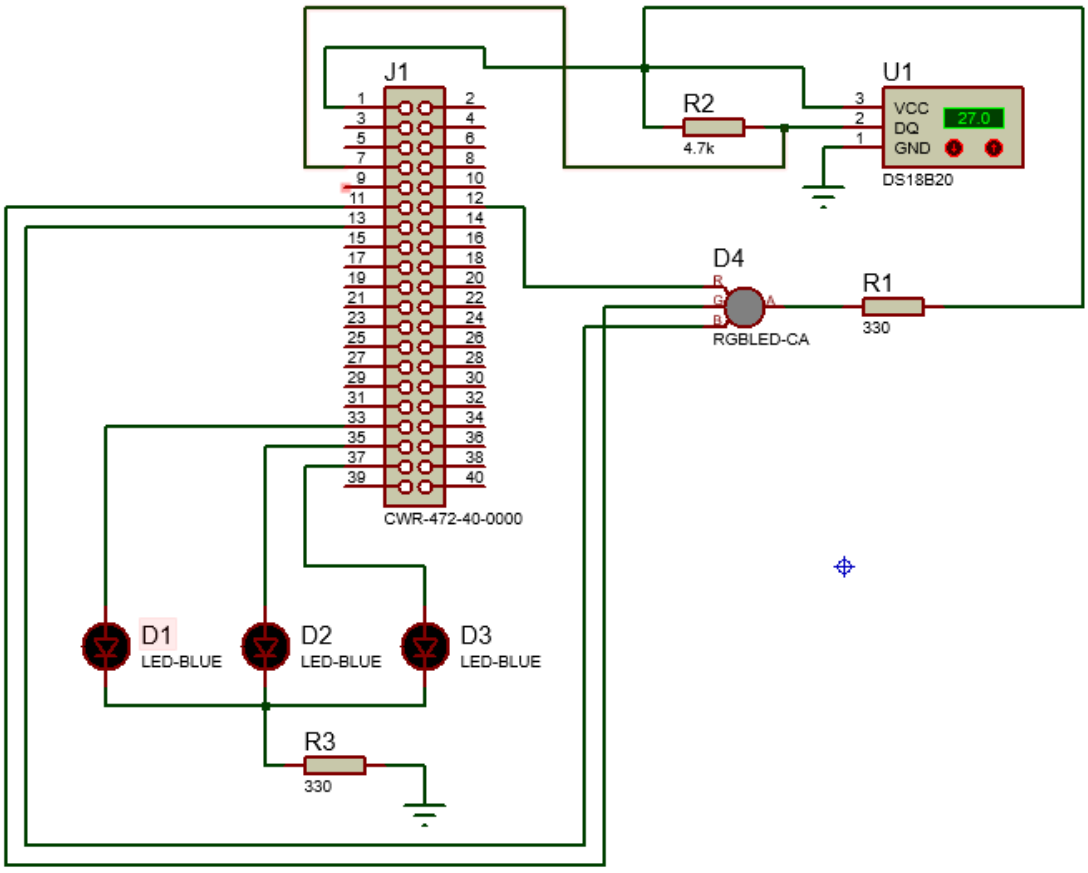


Fig. 3.18 Diagrama de conexión para el modelo de prueba

3.2.4 Programación y descripción del código

Una vez conectado el circuito se procederá a programar ambas Raspberry Pi, la programación será en Python, el código es mostrado a continuación (Es parecido, pero no idéntico en ambas Raspberry Pi).

3.2.4.1 Código nodo 1

```
## INICIALIZACION PARA XMPP
import logging
from sleekxmpp import ClientXMPP
from sleekxmpp.exceptions import IqError, IqTimeout

## INICIALIZACION PARA DS18B20 SENSOR DE TEMPERATURA
import time
from w1thermsensor import W1ThermSensor
sensor=W1ThermSensor()

## INICIALIZACION PARA GPIO
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT)
GPIO.output(18,GPIO.HIGH)
GPIO.setup(17,GPIO.OUT)
GPIO.output(17,GPIO.HIGH)
GPIO.setup(27,GPIO.OUT)
GPIO.output(27,GPIO.HIGH)
GPIO.setup(13,GPIO.OUT)
GPIO.setup(19,GPIO.OUT)
GPIO.setup(26,GPIO.OUT)

## LIBRERIAS DEL SISTEMA
import time

## FUNCIONES PARASABER EL ESTADO DE XMPP
def mesg_rec():
   GPIO.output(18,GPIO.LOW)
```

```
    time.sleep(.2)
    GPIO.output(18,GPIO.HIGH)
def mesg_sen():
    GPIO.output(17,GPIO.LOW)
    time.sleep(.2)
    GPIO.output(17,GPIO.HIGH)
def mesg_prc_st():
    GPIO.output(27,GPIO.LOW)
    time.sleep(.2)
def mesg_prc_en():
    GPIO.output(27,GPIO.HIGH)
    time.sleep(.2)

## FUNCIONES PARA ENCENDER Y APAGAR LEDS
def led_on(a):
    if a==1:
        GPIO.output(13,GPIO.HIGH)
    elif a==2:
        GPIO.output(19,GPIO.HIGH)
    elif a==3:
        GPIO.output(26,GPIO.HIGH)
def led_off(a):
    if a==1:
        GPIO.output(13,GPIO.LOW)
    elif a==2:
        GPIO.output(19,GPIO.LOW)
    elif a==3:
        GPIO.output(26,GPIO.LOW)
def led_ison(a):
    if a==1:
        if GPIO.input(13):
            return True
        else:
            return False
    elif a==2:
        if GPIO.input(19):
            return True
        else:
            return False
    elif a==3:
```

```
        if GPIO.input(26):
            return True
        else:
            return False
    def led_isoff(a):
        if a==1:
            if GPIO.input(13):
                return False
            else:
                return True
        elif a==2:
            if GPIO.input(19):
                return False
            else:
                return True
        elif a==3:
            if GPIO.input(26):
                return False
            else:
                return True

    ## CLASE PARA ACTIVAR XMPP
    class EchoBot(ClientXMPP):

        def __init__(self, jid, password):
            #Se ingresan como parametros de la function los datos de la cuenta XMPP
            ClientXMPP.__init__(self, jid, password)

            self.add_event_handler("session_start", self.session_start)
            self.add_event_handler("message", self.message)

        def session_start(self, event):
            self.send_presence()
            self.get_roster()

        def message(self, msg):
            suma = 0
            if msg['type'] in ('chat', 'normal'):
                #Avisa que ha recivido un mensaje
                mesg_rec()
```

```
#Guardamos el contenido del mensaje recivido
ms_tx = msg['body'].lower()
msg.reply("").send()

if "y ahora" in ms_tx:
   self.send_message(mto='pronnus@jabber.at',
         mbody=ms_tx,
         mtype='chat')
   mesg_sen()

elif ("baño" in ms_tx) or ("otra recamara" in ms_tx) or ("mi recamara" in ms_tx):
   self.send_message(mto='brugmansia@jabber.at',
         mbody=ms_tx,
         mtype='chat')
   mesg_sen()

elif ("hola" in ms_tx):
   msg['body'] = ("Hola, aqui estoy !!") #Cambia el texto del mensaje
   msg.send() #Envia este mensaje
   mesg_sen()

elif ("adios" in ms_tx):
   msg['body'] = ("Yo no ire a ningun lado") #Cambia el texto del mensaje
   msg.send() #Envia este mensaje
   mesg_sen()

elif (("cual" in ms_tx) or ("que" in ms_tx)) and ("temperatura" in ms_tx):
   msg['body'] = ("Y ahora le informo la temperatura") #Cambia el texto del mensaje
   msg.send() #Envia este mensaje
   mesg_sen()
   #Empieza un ciclo para leer varios valores de temperatura y darlos
   mesg_prc_st()
   temp = 0
   for i in range (5):
      temp = temp + sensor.get_temperature()
      time.sleep(.1)
   temp = temp/5
   mesg_prc_en()
   msg['body'] = ("Y ahora la temperatura es %s grados centigrados" % temp)
   msg.send() #Envia este mensaje
```

```
            mesg_sen()

        #Encender las luces
        elif ("enciende" in ms_tx) or ("encender" in ms_tx):
            if "cocina" in ms_tx:
                if led_isoff(1):
                    led_on(1)
                    msg['body'] = ("Y ahora ya encendi la luz de la cocina") #Cambia el texto del
mensaje
                else:
                    msg['body'] = ("Y ahora esa luz ya esta encendida") #Cambia el texto del
mensaje
                msg.send() #Envia este mensaje
                mesg_sen()
            elif "sala" in ms_tx:
                if led_isoff(2):
                    led_on(2)
                    msg['body'] = ("Y ahora ya encendi la luz de la sala") #Cambia el texto del
mensaje
                else:
                    msg['body'] = ("Y ahora esa luz ya esta encendida") #Cambia el texto del
mensaje
                msg.send() #Envia este mensaje
                mesg_sen()
            elif "comedor" in ms_tx:
                if led_isoff(3):
                    led_on(3)
                    msg['body'] = ("Y ahora ya encendi la luz del comedor") #Cambia el texto del
mensaje
                else:
                    msg['body'] = ("Y ahora esa luz ya esta encendida") #Cambia el texto del
mensaje
                msg.send() #Envia este mensaje
                mesg_sen()

        elif ("apaga" in ms_tx) or ("apagar" in ms_tx):
            if "cocina" in ms_tx:
                if led_ison(1):
                    led_off(1)
```

```
            msg['body'] = ("Y ahora ya apague la luz de la cocina") #Cambia el texto del
mensaje
          else:
            msg['body'] = ("Y ahora esa luz ya esta apagada") #Cambia el texto del
mensaje
          msg.send() #Envia este mensaje
          mesg_sen()
        elif "sala" in ms_tx:
          if led_ison(2):
            led_off(2)
            msg['body'] = ("Y ahora ya apague la luz de la sala") #Cambia el texto del
mensaje
          else:
            msg['body'] = ("Y ahora esa luz ya esta apagada") #Cambia el texto del
mensaje
          msg.send() #Envia este mensaje
          mesg_sen()
        elif "comedor" in ms_tx:
          if led_ison(3):
            led_off(3)
            msg['body'] = ("Y ahora ya apague la luz del comedor") #Cambia el texto del
mensaje
          else:
            msg['body'] = ("Y ahora esa luz ya esta apagada") #Cambia el texto del
mensaje
          msg.send() #Envia este mensaje
          mesg_sen()

## INICIA EL PROGRAMA

if __name__ == '__main__':

  logging.basicConfig(level=logging.DEBUG,
            format='%(levelname)-8s %(message)s')

  #Se ingresan los datos de la cuenta XMPP
  xmpp = EchoBot('datura@jabber.at', 'd4tur46&')
  #Inicia el proceso de conexión con el servidor y se bloquea
  xmpp.connect()
  xmpp.process(block = True)
```

```
    print("YASAALIIIIIII")
```

3.2.4.2 Código nodo 2

```
## INICIALIZACION PARA XMPP
import logging
from sleekxmpp import ClientXMPP
from sleekxmpp.exceptions import IqError, IqTimeout

## INICIALIZACION PARA DS18B20 SENSOR DE TEMPERATURA
import time
from w1thermsensor import W1ThermSensor
sensor=W1ThermSensor()

### INICIALIZACION PARA GPIO
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT)
GPIO.output(18,GPIO.HIGH)
GPIO.setup(17,GPIO.OUT)
GPIO.output(17,GPIO.HIGH)
GPIO.setup(27,GPIO.OUT)
GPIO.output(27,GPIO.HIGH)
GPIO.setup(13,GPIO.OUT)
GPIO.setup(19,GPIO.OUT)
GPIO.setup(26,GPIO.OUT)

## LIBRERIAS PARA EL SISTEMA
import time

## FUNCIONES PARA ESTADO DE XMPP
def mesg_rec():
    GPIO.output(18,GPIO.LOW)
    time.sleep(.2)
    GPIO.output(18,GPIO.HIGH)
def mesg_sen():
    GPIO.output(17,GPIO.LOW)
```

```
    time.sleep(.2)
    GPIO.output(17,GPIO.HIGH)
def mesg_prc_st():
    GPIO.output(27,GPIO.LOW)
    time.sleep(.2)
def mesg_prc_en():
    GPIO.output(27,GPIO.HIGH)
    time.sleep(.2)

## FUNCIONES PARA ENCENDER Y APAGAR LEDS
def led_on(a):
    if a==1:
        GPIO.output(13,GPIO.HIGH)
    elif a==2:
        GPIO.output(19,GPIO.HIGH)
    elif a==3:
        GPIO.output(26,GPIO.HIGH)
def led_off(a):
    if a==1:
        GPIO.output(13,GPIO.LOW)
    elif a==2:
        GPIO.output(19,GPIO.LOW)
    elif a==3:
        GPIO.output(26,GPIO.LOW)
def led_ison(a):
    if a==1:
        if GPIO.input(13):
            return True
        else:
            return False
    elif a==2:
        if GPIO.input(19):
            return True
        else:
            return False
    elif a==3:
        if GPIO.input(26):
            return True
        else:
            return False
```

```
def led_isoff(a):
    if a==1:
        if GPIO.input(13):
            return False
        else:
            return True
    elif a==2:
        if GPIO.input(19):
            return False
        else:
            return True
    elif a==3:
        if GPIO.input(26):
            return False
        else:
            return True

## CLASE PARA ACTIVAR XMPP
class EchoBot(ClientXMPP):

    def __init__(self, jid, password):
        #Se ingresan como parametros de la function los datos de la cuenta XMPP
        ClientXMPP.__init__(self, jid, password)

        self.add_event_handler("session_start", self.session_start)
        self.add_event_handler("message", self.message)

    def session_start(self, event):
        self.send_presence()
        self.get_roster()

    def message(self, msg):
        suma = 0
        if msg['type'] in ('chat', 'normal'):
            #Avisa que ha recivido un mensaje
            mesg_rec()
            #Guardamos el contenido del mensaje recivido
            ms_tx = msg['body'].lower()
            msg.reply("").send()
```

```
if "y ahora" in ms_tx:
   self.send_message(mto='pronnus@jabber.at',
         mbody=ms_tx,
         mtype='chat')
   mesg_sen()

elif ("cocina" in ms_tx) or ("sala" in ms_tx) or ("comedor" in ms_tx):
   self.send_message(mto='datura@jabber.at',
         mbody=ms_tx,
         mtype='chat')
   mesg_sen()

elif ("hola" in ms_tx):
   msg['body'] = ("Hola, aqui estoy !!") #Cambia el texto del mensaje
   msg.send() #Envia este mensaje
   mesg_sen()

elif ("adios" in ms_tx):
   msg['body'] = ("Yo no ire a ningun lado") #Cambia el texto del mensaje
   msg.send() #Envia este mensaje
   mesg_sen()

elif (("cual" in ms_tx) or ("que" in ms_tx)) and ("temperatura" in ms_tx):
   msg['body'] = ("Y ahora le informo la temperatura") #Cambia el texto del mensaje
   msg.send() #Envia este mensaje
   mesg_sen()
   #Empieza un ciclo para leer varios valores de temperatura y darlos
   mesg_prc_st()
   temp = 0
   for i in range (5):
      temp = temp + sensor.get_temperature()
      time.sleep(.1)
   temp = temp/5
   mesg_prc_en()
   msg['body'] = ("Y ahora la temperatura es %s grados centigrados" % temp)
   msg.send() #Envia este mensaje
   mesg_sen()

#Encender las luces
elif ("enciende" in ms_tx) or ("encender" in ms_tx):
```

```
            if "baño" in ms_tx:
               if led_isoff(1):
                  led_on(1)
                  msg['body'] = ("Y ahora ya encendi la luz del baño") #Cambia el texto del
mensaje
               else:
                  msg['body'] = ("Y ahora esa luz ya esta encendida") #Cambia el texto del
mensaje
               msg.send() #Envia este mensaje
               mesg_sen()
            elif "otra recamara" in ms_tx:
               if led_isoff(2):
                  led_on(2)
                  msg['body'] = ("Y ahora ya encendi la luz de esa recamara") #Cambia el texto
del mensaje
               else:
                  msg['body'] = ("Y ahora esa luz ya esta encendida") #Cambia el texto del
mensaje
               msg.send() #Envia este mensaje
               mesg_sen()
            elif "mi recamara" in ms_tx:
               if led_isoff(3):
                  led_on(3)
                  msg['body'] = ("Y ahora ya encendi la luz de tu recamara") #Cambia el texto
del mensaje
               else:
                  msg['body'] = ("Y ahora esa luz ya esta encendida") #Cambia el texto del
mensaje
               msg.send() #Envia este mensaje
               mesg_sen()

         elif ("apaga" in ms_tx) or ("apagar" in ms_tx):
            if "baño" in ms_tx:
               if led_ison(1):
                  led_off(1)
                  msg['body'] = ("Y ahora ya apague la luz del baño") #Cambia el texto del
mensaje
               else:
                  msg['body'] = ("Y ahora esa luz ya esta apagada") #Cambia el texto del
mensaje
```

```
            msg.send() #Envia este mensaje
            mesg_sen()
        elif "otra recamara" in ms_tx:
            if led_ison(2):
                led_off(2)
                msg['body'] = ("Y ahora ya apague la luz de esa recamara") #Cambia el texto
del mensaje
            else:
                msg['body'] = ("Y ahora esa luz ya esta apagada") #Cambia el texto del
mensaje
            msg.send() #Envia este mensaje
            mesg_sen()
        elif "mi recamara" in ms_tx:
            if led_ison(3):
                led_off(3)
                msg['body'] = ("Y ahora ya apague la luz de tu recamara") #Cambia el texto
del mensaje
            else:
                msg['body'] = ("Y ahora esa luz ya esta apagada") #Cambia el texto del
mensaje
            msg.send() #Envia este mensaje
            mesg_sen()

## INICIA EL PROGRAMA

if __name__ == '__main__':

    logging.basicConfig(level=logging.DEBUG,
                        format='%(levelname)-8s %(message)s')

     #Se ingresan los datos de la cuenta XMPP
    xmpp = EchoBot('brugmansia@jabber.at', '8rugm4ns14&')
    #Inicia el proceso de conexión con el servidor y se bloquea
    xmpp.connect()
    xmpp.process(block = True)
    print("YASAALIIIIIII")
```

3.2.4.3 Lista de comandos

A continuación, se describirán brevemente algunas funciones utilizadas (No se describirá la sintaxis de Python ni las funciones propias del lenguaje como bucles y estructuras, para más información de estas últimas o cualquier otra función consultar la documentación de Python [11]).

import logging

Importa la biblioteca "logging" para poder trabajar con eventos y módulos.

from sleekxmpp import ClientXMPP

Importa el modulo "ClientXMPP" de la biblioteca "sleekxmpp" que provee funcionalidades específicas para aplicaciones XMPP cliente.

from sleekxmpp.exceptions import IqError, IqTimeout

Importa los módulos "IqError" y "IqTimeout" de la biblioteca "sleekxmpp" para agregar excepciones propias de XMPP, como errores en los mensajes y cuando se excede el tiempo de espera.

import time

Importa la biblioteca "time" que se utilizara para crear retardos o delay's

from w1thermsensor import W1ThermSensor

Importa el modulo "W1ThermSensor" de la biblioteca "w1thermsensor" para implementar las funciones del sensor de temperatura.

sensor=W1ThermSensor()

Se crea el objeto “sensor” a partir de la clase “W1ThermSensor()” que se importó anteriormente, este objeto se utiliza para interactuar con el sensor de temperatura.

import RPi.GPIO as GPIO

Importa el modulo “RPi.GPIO” como “GPIO” que permite tener el control de entradas y salidas.

GPIO.setmode(GPIO.BCM)

Indica que los pines de la tarjeta se numeraran según el "Broadcom SOC channel", y no según su posición.

GPIO.setwarnings(False)

Deshabilita los avisos “warnings” para los puertos GPIO

GPIO.setup(numero_pin, entrada_o_salida)

Declara el pin “numero_pin” con el string indicado en “entrada_o_salida”.

GPIO.output(numero_pin,nuevo_estado)

Pone el pin “numero_pin” en el estado “nuevo_estado”.

def nombre_funcion(parametros):

Declara una nueva función con el nombre “nombre_funcion” y con los parámetros indicados.

time.sleep(segundos)

Genera un retardo de “segundos”.

GPIO.input(numero_pin):

Lee el estado actual del pin "numero_pin"

class EchoBot(ClientXMPP):

Se crea la clase "EchoBot" heredada de la clase "ClientXMPP".

ClientXMPP.__init__(self, jid, password)

Especifica la plantilla para solicitar los datos de la cuenta XMPP que iniciara sesión en el dispositivo.

self.add_event_handler("session_start", self.session_start)

Agrega un controlador de eventos para los eventos generados en el programa

self.send_presence()

Indica al servidor que se informe que un cliente especificado inicio sesión.

self.get_roster()

Solicita el "Roster" (Lista de contactos) al servidor.

if msg['type'] in ('chat', 'normal'):

Verifica que el mensaje recibido sea tipo "chat", son los usados en la comunicación

ms_tx = msg['body'].lower()

Convierte el texto del mensaje recibido a minúsculas, para evitar problemas de coincidencia.

```
msg.reply("").send()
```

Envía una respuesta al último mensaje recibido modificando el campo 'body' y enviándolo al usuario del que se recibió.

```
self.send_message(mto='pronnus@jabber.at', mbody=ms_tx, mtype='chat')
```

Envía un nuevo mensaje con los argumentos especificados.

```
msg['body'] = ("Cuerpo del mensaje")
```

Cambia el contenido del mensaje especificado.

```
msg.send()
```

Envía inmediatamente el mensaje indicado.

```
sensor.get_temperature()
```

Solicita al sensor el valor actual de la temperatura.

```
logging.basicConfig(level=logging.DEBUG, format='%(levelname)-8s %(message)s')
```

Configura el sistema de entradas creando manejadores en diferentes jerarquías.

```
xmpp = EchoBot('datura@jabber.at', 'd4tur46&')
```

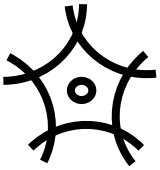
Crea el objeto "xmpp" de la clase "EchoBot()" que se definió anteriormente con los datos de la cuenta que iniciara sesión en el dispositivo.

xmpp.connect()

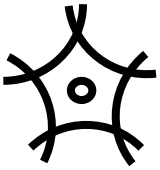
Se conecta al servidor XMPP especificado en los argumentos, si no es especificado se utilizará el servidor dado en el JID.

xmpp.process(block = True)

Bloquea el hilo de ejecución actual para mantener la sesión iniciada.



4. PRUEBAS Y RESULTADOS EXPERIMENTALES



4.1 Inicio del Sistema

Una vez que se tiene el circuito conectado y las Raspberry Pi programados se puede proceder a iniciar el sistema, el proceso de inicio es el siguiente.

1. Inicio del cliente
2. Inicio Nodo 1
3. Inicio Nodo 2

A continuación, se detallará el inicio de sesión de cada uno de los dispositivos.

4.1.1 Inicio del cliente PC

Una vez instalado Gajim y creadas las cuentas como se explicó en el capítulo anterior, se ejecuta el programa, aparecerá el siguiente cuadro de dialogo (Fig. 4.1). Se selecciona la opción señalada y se da clic en "Adelante".

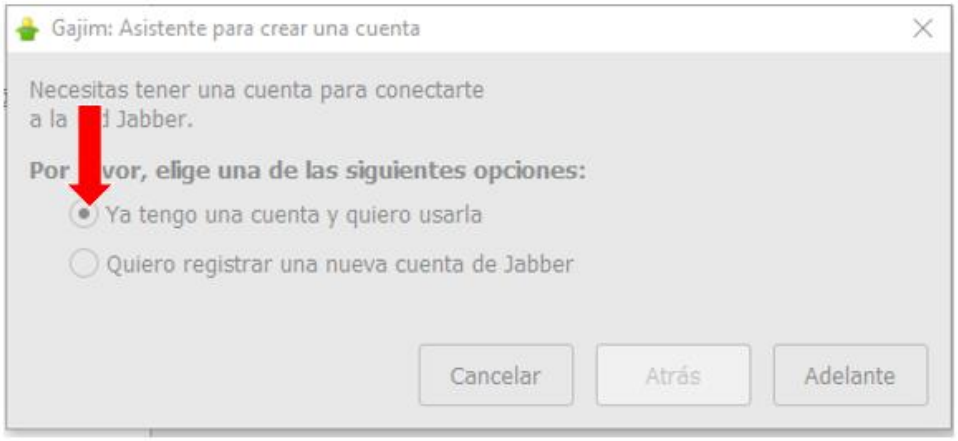


Fig. 4.1 Inicio de sesión en Gajim

En este punto (Fig. 4.2), se tienen que introducir los datos de la cuenta que fue creada anteriormente, una vez introducidos los datos se da clic en "Adelante".

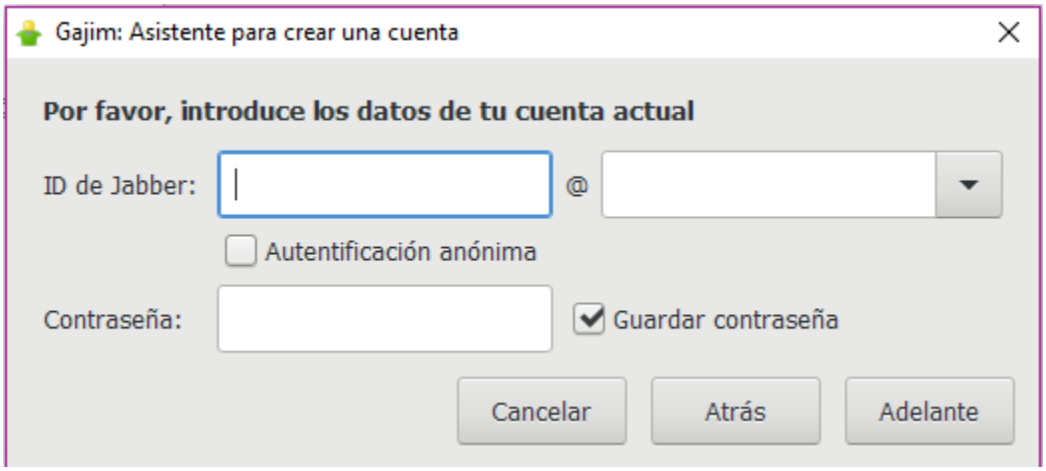


Fig. 4.2 Inicio de sesión en Gajim con cuenta previamente creada

Si los datos de la cuenta son correctos, se inicia sesión en el servidor remoto, si es la primera vez que se inicia sesión en la cuenta no aparecerán contactos así que el siguiente paso es agregar como contactos las otras dos cuentas creadas, en la barra de menús se selecciona la opción (Fig. 4.3).

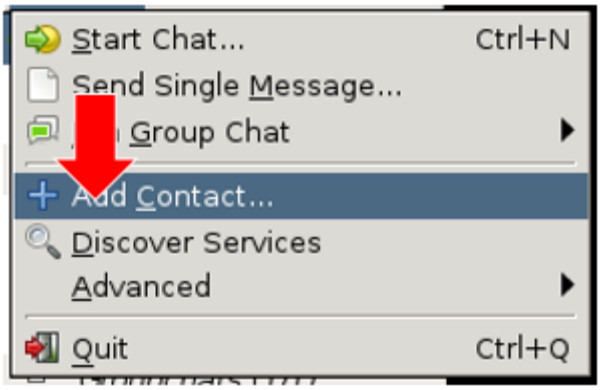


Fig. 4.3 Agregar contacto nuevo en Gajim

Se despliega la ventana (Fig. 4.4) para colocar la información del contacto, este proceso se repetirá una vez para cada contacto.

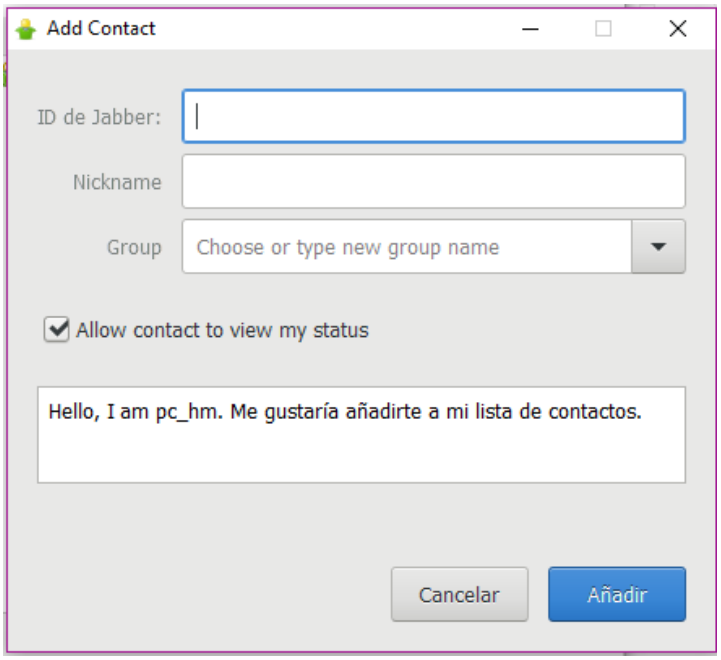


Fig. 4.4 Información del contacto nuevo en Gajim

Una vez agregados ambos contactos, aparecerán en la lista de contactos (Fig. 4.5). Aparecen como desconectados porque no se ha iniciado el programa en las Raspberry pi.

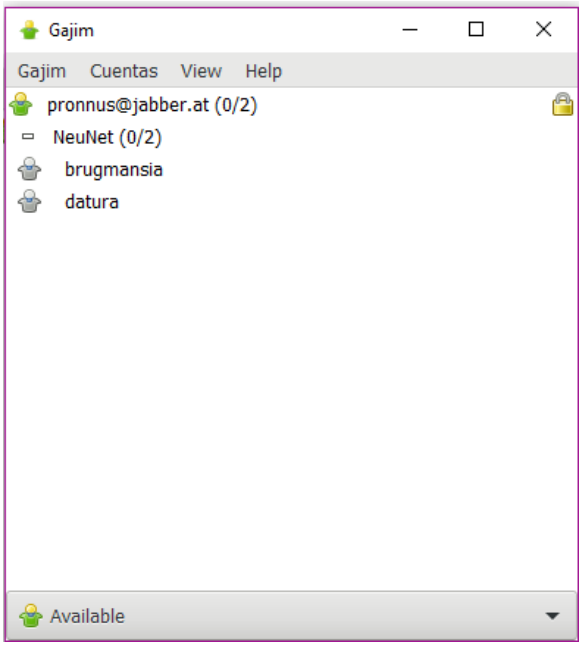


Fig. 4.5 Lista de contactos en Gajim

4.1.2 Inicio del cliente Android

Se descarga desde la tienda oficial de Android la aplicación AstraChat (Fig. 4.6). Se puede utilizar cualquier otra aplicación que sea un cliente XMPP.

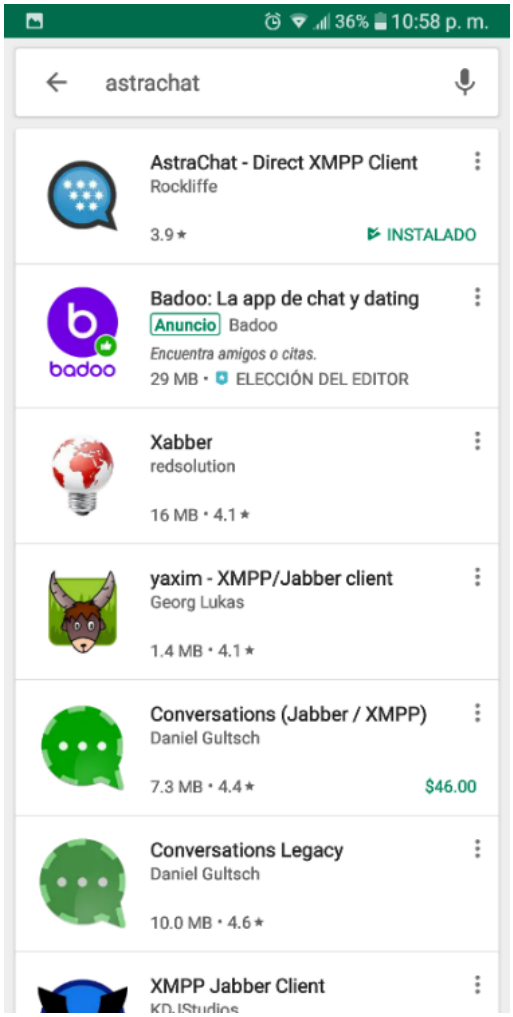


Fig. 4.6 Aplicación AstraChat en Android

Una vez instalada la aplicación, se da clic en "Abrir" (Fig. 4.7).

Fig. 4.7 Instalación de Astrachat en Android

Una vez abierta la aplicación (Fig. 4.8), se introducen los datos de la cuenta, son los mismos datos que fueron utilizados para el cliente PC.

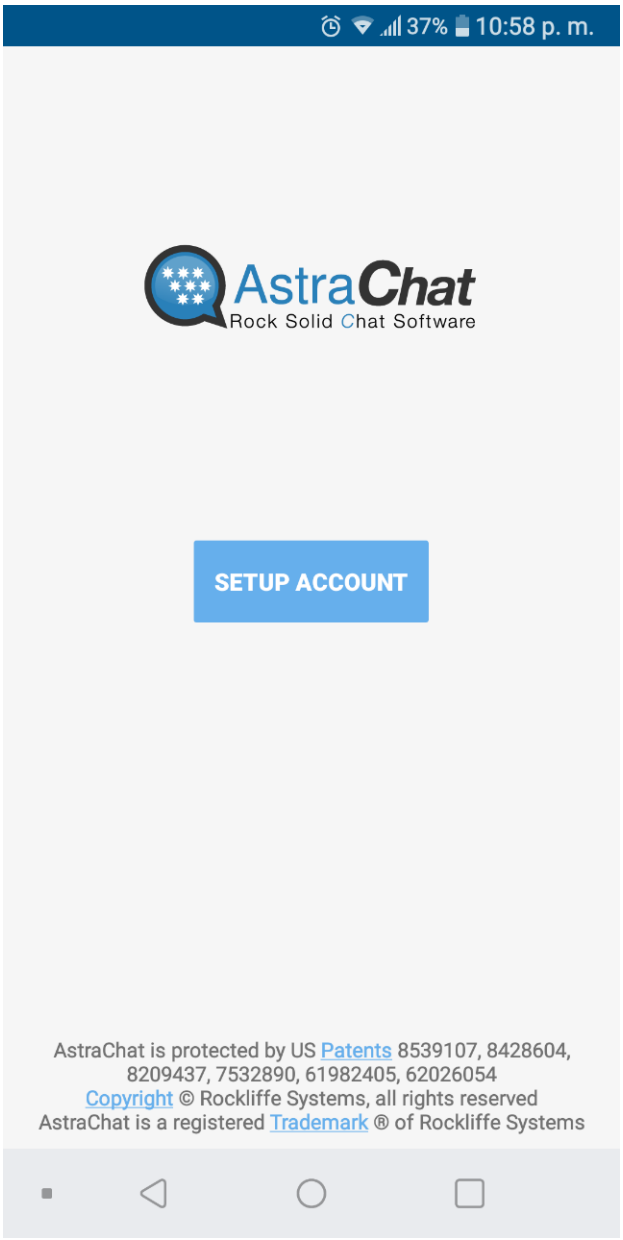


Fig. 4.8 Inicio de sesión en Astrachat

Si los datos son correctos la sesión iniciara, se puede notar que los contactos agregados en el cliente PC aparecen aquí (Fig. 4.9), esto es debido a que los datos son guardados en el servidor XMPP que alberga la cuenta.

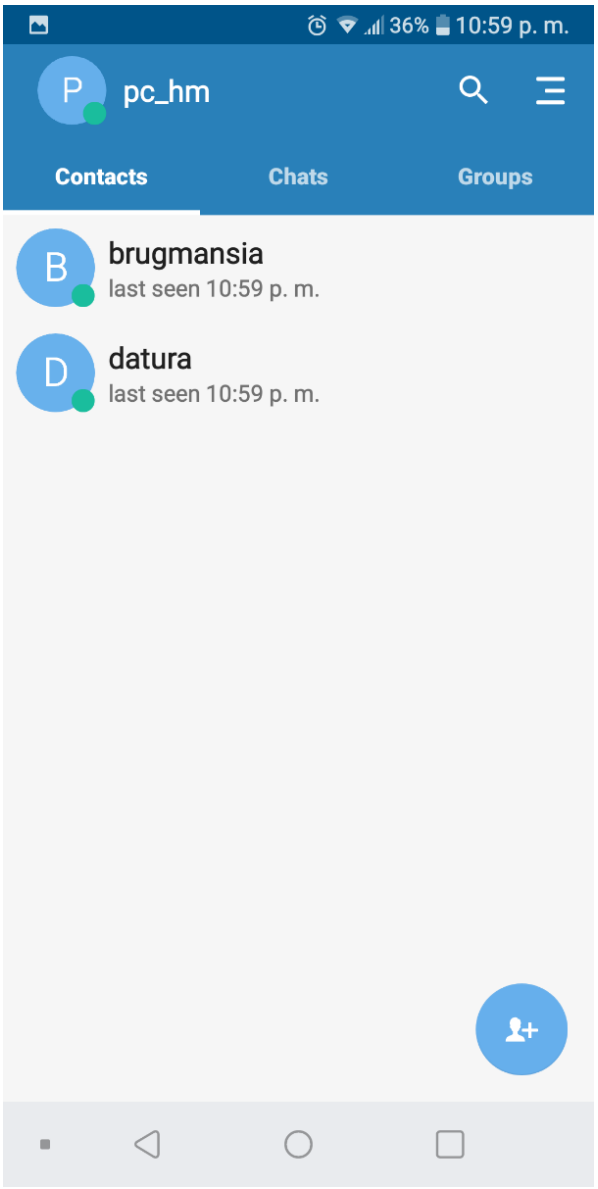


Fig. 4.9 Lista de contactos en Astrachat

En este punto la sesión esta activa y se procederá con el siguiente paso. Cabe mencionar que se puede tener la sesión activa en el cliente PC y Android a la vez.

4.1.3 Inicio Nodo 1

Una vez que se tiene el código del programa en la Raspberry Pi, se selecciona la opción de "Run Module" o se oprime la tecla F5. Enseguida

comenzara el proceso de inicio de sesión con el servidor que consta de varios pasos y tardara aproximadamente un minuto. El proceso de inicio será descrito a continuación.

- Proceso de carga de las bibliotecas y funciones del protocolo XMPP en la memoria RAM (Fig. 4.10)
- Se solicita la dirección ip del servidor ubicado en "jabber.at" al servidor DNS correspondiente (Fig. 4.10)
- Petición de conexión a la ip del servidor obtenida (Fig. 4.10)
- Se completa la conexión y se genera el evento de "conectando" (Fig. 4.10)
- Comienza la transición de "desconectado" a "conectado" (Fig. 4.10)
- Inicia un hilo de ejecución para el administrador de eventos del protocolo (Fig. 4.10)
- Se envían al servidor los parámetros de la sesión XMPP actual (Fig. 4.10 y Fig. 4.11)
- Inicia la negociación para cifrado de canal TLS, enviando el certificado SSL (Fig. 4.11 y Fig. 4.12)
- Negociación del cifrado de canal (Fig. 4.12 y Fig. 4.13)
- Se completa la negociación del cifrado (Fig. 4.13)
- Acuerdo sobre el formato de los mensajes que se intercambiaran en la comunicación (Fig. 4.13 y Fig. 4.14)
- Se solicita la lista de contactos actualizada o "roster" (Fig. 4.14 y Fig. 4.15)
- Finaliza el intercambio de datos sobre las características de los mensajes que se intercambiaran. (Fig. 4.15)
- La sesión se establece bajo los parámetros acordados. (Fig. 4.15)

- Se envía la notificación de que se inició sesión y se avisa a los contactos. (Fig. 4.15)
- Se solicita información de los contactos y el estado de los mismos. (Fig. 4.16)
- Cambia el estado a "disponible" (Fig. 4.17)
- Se envían mensajes cada determinado tiempo para mantener la sesión iniciada (Fig. 4.17)

```
====================== RESTART: /home/pi/xmpp_sensor.py ======================
DEBUG    Loaded Plugin: RFC 6120: Stream Feature: STARTTLS
DEBUG    Loaded Plugin: RFC 6120: Stream Feature: Resource Binding
DEBUG    Loaded Plugin: RFC 3920: Stream Feature: Start Session
DEBUG    Loaded Plugin: RFC 6121: Stream Feature: Roster Versioning
DEBUG    Loaded Plugin: RFC 6121: Stream Feature: Subscription Pre-Approval
DEBUG    Loaded Plugin: RFC 6120: Stream Feature: SASL
DEBUG    Waiting 2.307865917863965 seconds before connecting.
WARNING  DNS: dnspython not found. Can not use SRV lookup.
DEBUG    DNS: Querying jabber.at for AAAA records.
DEBUG    DNS: Error retreiving AAAA address info for jabber.at.
DEBUG    DNS: Querying jabber.at for A records.
DEBUG    Connecting to 138.201.246.149:5222
DEBUG    Event triggered: connected
DEBUG     ==== TRANSITION disconnected -> connected
DEBUG    Starting HANDLER THREAD
DEBUG    Loading event runner
DEBUG    SEND (IMMED): <stream:stream to='jabber.at' xmlns:stream='http://ethe
rx.jabber.org/streams' xmlns='jabber:client' xml:lang='en' version='1.0'>
```

Fig. 4.10 Inicio de sesión nodo 1 (parte 1)

```
DEBUG    RECV: <stream:stream id="15672641963668252673" xml:lang="en" from="ja
bber.at" version="1.0">
DEBUG    RECV: <stream:features xmlns="http://etherx.jabber.org/streams"><star
ttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"><required /></starttls></stream:f
eatures>
DEBUG    SEND (IMMED): <starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"><requ
ired /></starttls>
DEBUG    RECV: <proceed xmlns="urn:ietf:params:xml:ns:xmpp-tls" />
DEBUG    Starting TLS
INFO     Negotiating TLS
INFO     Using SSL version: TLSv1
DEBUG    CERT: -----BEGIN CERTIFICATE-----
MIIHZjCCBk6gAwIBAgISA8coOmd9Ct6hr7MriLi/dWyVMA0GCSqGSIb3DQEBCwUA
MEoxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MSMwIQYDVQQD
ExpMZXQncyBFbmNyeXB0IEF1dGhvcml0eSBYMzAeFw0xODExMDExMDA2NDVaFw0x
OTAxMzAxMDA2NDVaMBQxEjAQBgNVBAMTCWphYmJlci5hdDCCAiIwDQYJKoZIhvcN
AQEBBQADggIPADCCAgoCggIBAJvZTeQ2Mec7oKmcQWCXOnSRO3L1OO9vxnKqowDw
```

Fig. 4.11 Inicio de sesión nodo 1 (parte 2)

```
jB3IBjFJrL9TL2U6qyg7XoSBrAKwdc7aMndzQ+5trCNbhyVHywpchde2ZXYHNzYo
YZeSjiqk0tYTwhL3nVHl4kKM/uR+qjsVCN5WZBXxv2n/zMwGpmS18TSA/PCRfjTS
ueMc5USDokQcuwRNKTOcPoxrJQDEC/Hndg8CfkFQop3SebcMF4ryn5ZZek0n/qco
KYCgxQOf1LMvB7lQ+RDEJ435xou/n5NNR2k=
-----END CERTIFICATE-----

DEBUG    Event triggered: ssl_cert
WARNING  Could not find pyasn1 and pyasn1_modules. SSL certificate COULD NOT B
E VERIFIED.
DEBUG    SEND (IMMED): <stream:stream to='jabber.at' xmlns:stream='http://ethe
rx.jabber.org/streams' xmlns='jabber:client' xml:lang='en' version='1.0'>
DEBUG    RECV: <stream:stream id="15072313247514275841" xml:lang="en" from="ja
bber.at" version="1.0">
DEBUG    RECV: <stream:features xmlns="http://etherx.jabber.org/streams"><mech
anisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl"><mechanism>PLAIN</mechanism><m
echanism>SCRAM-SHA-1</mechanism></mechanisms><register xmlns="http://jabber.or
g/features/iq-register" /></stream:features>
```

Fig. 4.12 Inicio de sesión nodo 1 (parte 3)

```
DEBUG    SEND (IMMED): <auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanis
m="SCRAM-SHA-1">eSwsbj1kYXR1cmEscj0zNzUxMzUwMjM1NjY2OTU5</auth>
DEBUG    RECV: <challenge xmlns="urn:ietf:params:xml:ns:xmpp-sasl">cj0zNzUxMzU
wMjM1NjY2OTU5dFU2d0FHditya0o5dWIwSG8vUDZ1QT09LHM9QzBYSExJUXhIV2pKVGl5QXJLT1p1Z
z09LGk9NDA5Ng==</challenge>
DEBUG    SEND (IMMED): <response xmlns="urn:ietf:params:xml:ns:xmpp-sasl">Yz1l
U3dzLHI9Mzc1MTM1MDIzNTY2Njk1OXRVNndBR3YrcmtKOXViMEhvL1A2dUE9PSxwPXU2QUM0ZFkxND
FDclVoNm5LMHZnaWZDZUkyRT0=</response>
DEBUG    RECV: <success xmlns="urn:ietf:params:xml:ns:xmpp-sasl">dj1adkJILzlMb
ENzMGc1K0Z1VHMzazJ6MzNFUTA9</success>
DEBUG    Event triggered: auth_success
DEBUG    SEND (IMMED): <stream:stream to='jabber.at' xmlns:stream='http://ethe
rx.jabber.org/streams' xmlns='jabber:client' xml:lang='en' version='1.0'>
DEBUG    RECV: <stream:stream id="9457098449969170433" xml:lang="en" from="jab
ber.at" version="1.0">
```

Fig. 4.13 Inicio de sesión nodo 1 (parte 4)

```
DEBUG    RECV: <stream:features xmlns="http://etherx.jabber.org/streams"><comp
ression xmlns="http://jabber.org/features/compress"><method>zlib</method></com
pression><bind xmlns="urn:ietf:params:xml:ns:xmpp-bind" /><session xmlns="urn:
ietf:params:xml:ns:xmpp-session"><optional /></session><c xmlns="http://jabber
.org/protocol/caps" node="http://www.process-one.net/en/ejabberd/" ver="u6Raqu
DQRHZ1CvXImtPmkgpmaj4=" hash="sha-1" /><sm xmlns="urn:xmpp:sm:2" /><sm xmlns="
urn:xmpp:sm:3" /><ver xmlns="urn:xmpp:features:rosterver" /><csi xmlns="urn:xm
pp:csi:0" /></stream:features>
DEBUG    Enabling roster versioning.
DEBUG    Requesting resource:
DEBUG    SEND (IMMED): <iq type="set" id="99f9e290-f525-4dbe-a212-ea97f4bb0d9f
-1"><bind xmlns="urn:ietf:params:xml:ns:xmpp-bind" /></iq>
DEBUG    RECV: <iq type="result" id="99f9e290-f525-4dbe-a212-ea97f4bb0d9f-1"><
bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"><jid>datura@jabber.at/1101378873
802815692947774899</jid></bind></iq>
DEBUG    Event triggered: session_bind
INFO     JID set to: datura@jabber.at/1101378873802815692947774899
DEBUG    SEND (IMMED): <iq type="set" id="99f9e290-f525-4dbe-a212-ea97f4bb0d9f
-2"><session xmlns="urn:ietf:params:xml:ns:xmpp-session" /></iq>
```

Fig. 4.14 Inicio de sesión nodo 1 (parte 5)

```
DEBUG    RECV: <iq type="result" id="99f9e290-f525-4dbe-a212-ea97f4bb0d9f-2" t
o="datura@jabber.at/11013788738028156929477748 99" from="datura@jabber.at" xml:
lang="en" />
DEBUG    Established Session
DEBUG    Event triggered: session_start
DEBUG    Finished processing stream features.
DEBUG    Event triggered: stream_negotiated
WARNING  Could not find pyasn1 and pyasn1_modules. SSL certificate expiration
COULD NOT BE VERIFIED.
DEBUG    Event triggered: sent_presence
DEBUG    SEND: <presence xml:lang="en" />
DEBUG    SEND: <iq type="get" id="99f9e290-f525-4dbe-a212-ea97f4bb0d9f-3"><que
ry xmlns="jabber:iq:roster" ver="" /></iq>
DEBUG    RECV: <presence to="datura@jabber.at/11013788738028156929477748 99" fr
om="datura@jabber.at/11013788738028156929477748 99" xml:lang="en"><x xmlns="vca
rd-temp:x:update" /></presence>
```

Fig. 4.15 Inicio de sesión nodo 1 (parte 6)

```
DEBUG    RECV: <presence id="c368db1f-828e-4d71-b952-be3e92a2c616" to="datura@
jabber.at/11013788738028156929477748 99" from="pronnus@jabber.at/gajim.MH7JE29Q
" xml:lang="es"><c xmlns="http://jabber.org/protocol/caps" node="http://gajim.
org" ver="EhDgXYarwDkGz8n/wbp2z37FJWE=" hash="sha-1" /><x xmlns="vcard-temp:x:
update"><photo /></x><delay xmlns="urn:xmpp:delay" stamp="2019-01-11T01:55:06.
645911Z" from="pronnus@jabber.at/gajim.MH7JE29Q" /><priority>50</priority></pr
esence>
DEBUG    RECV: <iq type="result" id="99f9e290-f525-4dbe-a212-ea97f4bb0d9f-3" t
o="datura@jabber.at/11013788738028156929477748 99" from="datura@jabber.at" xml:
lang="en"><query xmlns="jabber:iq:roster" ver="f7a88f005d0de03fb41d82d607d607e
a11ad76df"><item jid="pronnus@jabber.at" subscription="both" name="pronnus" />
<item jid="brugmansia@jabber.at" subscription="both" name="brugmansia" /></que
ry></iq>
DEBUG    Event triggered: roster_update
DEBUG    Event triggered: presence
DEBUG    Event triggered: presence_available
DEBUG    Event triggered: presence
DEBUG    Event triggered: presence_available
DEBUG    Event triggered: presence_available
```

Fig. 4.16 Inicio de sesión nodo 1 (parte 7)

```
DEBUG    Event triggered: got_online
DEBUG    Event triggered: changed_status
DEBUG    Event triggered: got_online
DEBUG    Event triggered: changed_status
DEBUG    Scheduled event: Session timeout check: ()
DEBUG    Scheduled event: Whitespace Keepalive: (' ',)
```

Fig. 4.17 Inicio de sesión nodo 1 (parte 8)

Si el proceso anterior fue exitoso, en el cliente PC aparecerá el siguiente mensaje (Fig. 4.18) en la parte inferior de la pantalla.

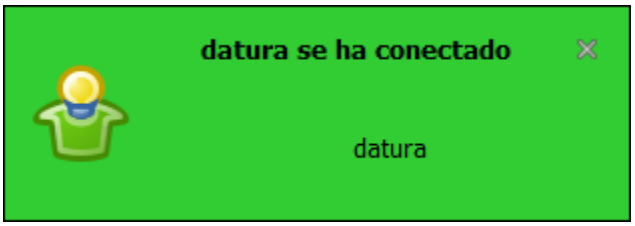


Fig. 4.18 Notificación de inicio de sesión del nodo 1

El mensaje indica que la Raspberry Pi está en línea, para comprobar que la comunicación es exitosa, se envía el siguiente mensaje de prueba (Fig. 4.19).

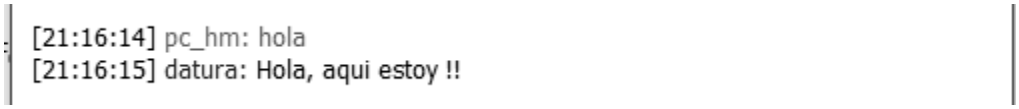


Fig. 4.19 Mensaje de prueba de PC a nodo 1

Cuando se recibe el mensaje en la Raspberry Pi, se genera un evento y la respuesta programada se envía (Fig. 4.20).

```
DEBUG    RECV: <message type="chat" id="fcd51538-88a4-44c7-8233-8006a920a35d"
to="datura@jabber.at/110137887380281569294774899" from="pronnus@jabber.at/gaj
im.MH7JE29Q" xml:lang="es"><archived xmlns="urn:xmpp:mam:tmp" by="datura@jabbe
r.at" id="1547176584518029" /><stanza-id xmlns="urn:xmpp:sid:0" by="datura@jab
ber.at" id="1547176584518029" /><origin-id xmlns="urn:xmpp:sid:0" id="fcd51538
-88a4-44c7-8233-8006a920a35d" /><request xmlns="urn:xmpp:receipts" /><body>hol
a</body><thread>qUGzVlZheixXMraylceZAegsMEhSaEed</thread></message>
DEBUG    Event triggered: message
DEBUG    SEND: <message type="chat" to="pronnus@jabber.at/gajim.MH7JE29Q" xml:
lang="es"><thread>qUGzVlZheixXMraylceZAegsMEhSaEed</thread></message>
DEBUG    SEND: <message type="chat" to="pronnus@jabber.at/gajim.MH7JE29Q" xml:
lang="es"><thread>qUGzVlZheixXMraylceZAegsMEhSaEed</thread><body>Hola, aqui es
toy !!</body></message>
```

Fig. 4.20 Mensaje recibido en nodo 1

4.1.4 Inicio Nodo 2

Una vez que se tiene el código del programa en la Raspberry Pi, se selecciona la opción de "Run Module" o se oprime la tecla F5. Enseguida comenzara el proceso de inicio de sesión con el servidor que consta de varios pasos y tardara aproximadamente un minuto. El proceso de inicio será descrito a continuación.

- Proceso de carga de las bibliotecas y funciones del protocolo XMPP en la memoria RAM (Fig. 4.21)

- Se solicita la dirección ip del servidor ubicado en "jabber.at" al servidor DNS correspondiente (Fig. 4.21)
- Petición de conexión a la ip del servidor obtenida (Fig. 4.21)
- Se completa la conexión y se genera el evento de "conectando" (Fig. 4.21)
- Comienza la transición de "desconectado" a "conectado" (Fig. 4.21)
- Inicia un hilo de ejecución para el administrador de eventos del protocolo (Fig. 4.21)
- Se envían al servidor los parámetros de la sesión XMPP actual (Fig. 4.21)
- Inicia la negociación para cifrado de canal TLS, enviando el certificado SSL (Fig. 4.21 y Fig. 4.22)
- Negociación del cifrado de canal (Fig. 4.22 y Fig. 4.23)
- Se completa la negociación del cifrado (Fig. 4.23)
- Acuerdo sobre el formato de los mensajes que se intercambiaran en la comunicación (Fig. 4.23 y Fig. 4.24)
- Se solicita la lista de contactos actualizada o "roster" (Fig. 4.24 y Fig. 4.25)
- Finaliza el intercambio de datos sobre las características de los mensajes que se intercambiaran. (Fig. 4.25)
- La sesión se establece bajo los parámetros acordados. (Fig. 4.25)
- Se envía la notificación de que se inició sesión y se avisa a los contactos. (Fig. 4.25)
- Se solicita información de los contactos y el estado de los mismos. (Fig. 4.26)

- Cambia el estado a "disponible" (Fig. 4.27)
- Se envían mensajes cada determinado tiempo para mantener la sesión iniciada (Fig. 4.27)

```
====================== RESTART: /home/pi/xmpp_sensor.py ======================
DEBUG    Loaded Plugin: RFC 6120: Stream Feature: STARTTLS
DEBUG    Loaded Plugin: RFC 6120: Stream Feature: Resource Binding
DEBUG    Loaded Plugin: RFC 3920: Stream Feature: Start Session
DEBUG    Loaded Plugin: RFC 6121: Stream Feature: Roster Versioning
DEBUG    Loaded Plugin: RFC 6121: Stream Feature: Subscription Pre-Approval
DEBUG    Loaded Plugin: RFC 6120: Stream Feature: SASL
DEBUG    Waiting 1.9442618918074155 seconds before connecting.
WARNING  DNS: dnspython not found. Can not use SRV lookup.
DEBUG    DNS: Querying jabber.at for AAAA records.
DEBUG    DNS: Error retreiving AAAA address info for jabber.at.
DEBUG    DNS: Querying jabber.at for A records.
DEBUG    Connecting to 138.201.246.149:5222
DEBUG    Event triggered: connected
DEBUG     ==== TRANSITION disconnected -> connected
DEBUG    Starting HANDLER THREAD
DEBUG    Loading event runner
```

Fig. 4.21 Inicio de sesión nodo 2 (parte 1)

```
DEBUG    SEND (IMMED): <stream:stream to='jabber.at' xmlns:stream='http://ethe
rx.jabber.org/streams' xmlns='jabber:client' xml:lang='en' version='1.0'>
DEBUG    RECV: <stream:stream xml:lang="en" id="7119644207145929729" from="jab
ber.at" version="1.0">
DEBUG    RECV: <stream:features xmlns="http://etherx.jabber.org/streams"><star
ttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"><required /></starttls></stream:f
eatures>
DEBUG    SEND (IMMED): <starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"><requ
ired /></starttls>
DEBUG    RECV: <proceed xmlns="urn:ietf:params:xml:ns:xmpp-tls" />
DEBUG    Starting TLS
INFO     Negotiating TLS
INFO     Using SSL version: TLSv1
DEBUG    CERT: -----BEGIN CERTIFICATE-----
MIIHZjCCBk6gAwIBAgISA8coOmd9Ct6hr7MriLi/dWyVMA0GCSqGSIb3DQEBCwUA
MEoxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MSMwIQYDVQQD
ExpMZXQncyBFbmNyeXB0IEF1dGhvcml0eSBYMzAeFw0xODExMDExMDA2NDVaFw0x
OTAxMzAxMDA2NDVaMBQxEjAQBgNVBAMTCWphYmJlci5hdDCCAiIwDQYJKoZIhvcN
AQEBBQADggIPADCCAgoCggIBAJvZIeO2Mec7oKmcQWCXQnSR03L1QO9vxnKqowDw
```

Fig. 4.22 Inicio de sesión nodo 2 (parte 2)

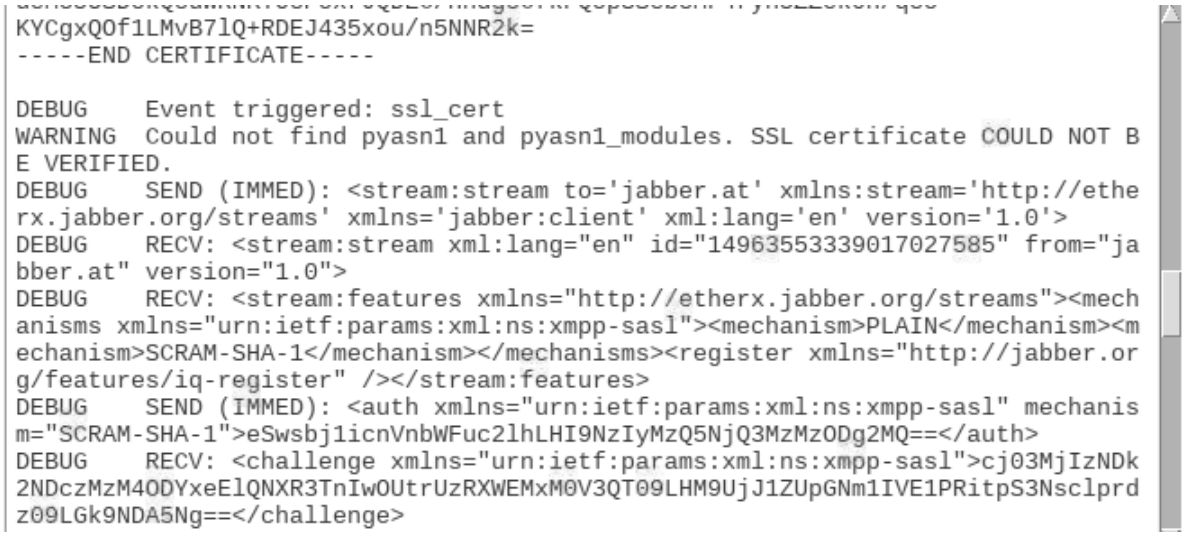


Fig. 4.23 Inicio de sesión nodo 2 (parte 3)

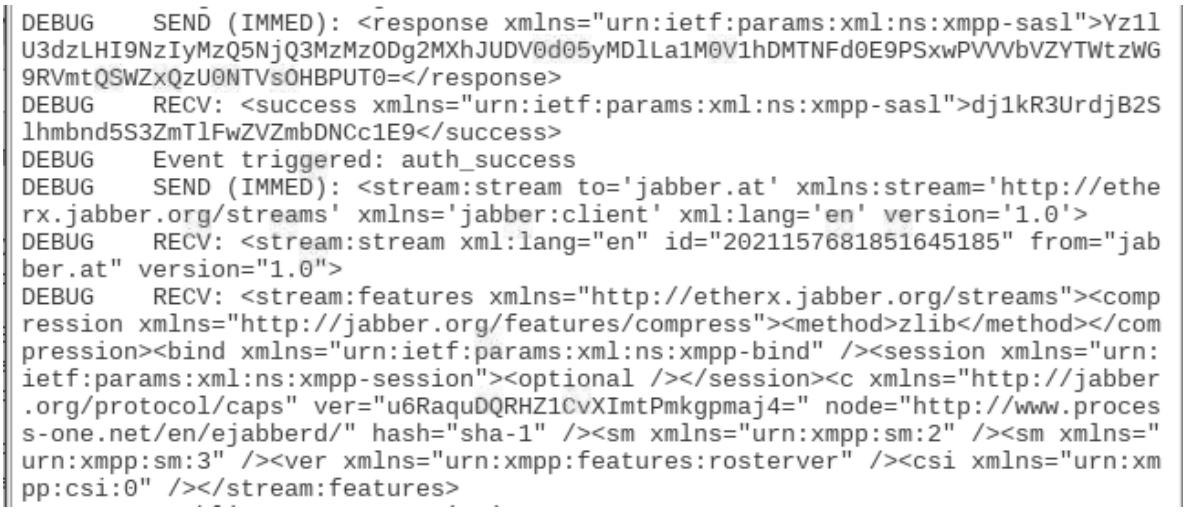
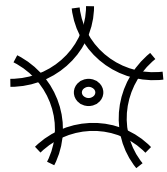


Fig. 4.24 Inicio de sesión nodo 2 (parte 4)



```
DEBUG    Enabling roster versioning.
DEBUG    Requesting resource:
DEBUG    SEND (IMMED): <iq type="set" id="158731ce-3fc7-4a30-9c8e-2649daa848c9
-1"><bind xmlns="urn:ietf:params:xml:ns:xmpp-bind" /></iq>
DEBUG    RECV: <iq id="158731ce-3fc7-4a30-9c8e-2649daa848c9-1" type="result"><
bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"><jid>brugmansia@jabber.at/144902
01462598072321318910 45</jid></bind></iq>
DEBUG    Event triggered: session_bind
INFO     JID set to: brugmansia@jabber.at/144902014625980723213189104 5
DEBUG    SEND (IMMED): <iq type="set" id="158731ce-3fc7-4a30-9c8e-2649daa848c9
-2"><session xmlns="urn:ietf:params:xml:ns:xmpp-session" /></iq>
DEBUG    RECV: <iq to="brugmansia@jabber.at/144902014625980723213189104 5" xml:
lang="en" id="158731ce-3fc7-4a30-9c8e-2649daa848c9-2" type="result" from="brug
mansia@jabber.at" />
DEBUG    Established Session
DEBUG    Event triggered: session_start
DEBUG    Finished processing stream features.
DEBUG    Event triggered: stream_negotiated
WARNING  Could not find pyasn1 and pyasn1_modules. SSL certificate expiration
COULD NOT BE VERIFIED.
```

Fig. 4.25 Inicio de sesión nodo 2 (parte 5)

```
DEBUG    Event triggered: sent_presence
DEBUG    SEND: <presence xml:lang="en" />
DEBUG    SEND: <iq type="get" id="158731ce-3fc7-4a30-9c8e-2649daa848c9-3"><que
ry xmlns="jabber:iq:roster" ver="" /></iq>
DEBUG    RECV: <presence to="brugmansia@jabber.at/144902014625980723213189104 5
" xml:lang="en" from="brugmansia@jabber.at/144902014625980723213189104 5"><x xm
lns="vcard-temp:x:update" /></presence>
DEBUG    RECV: <iq to="brugmansia@jabber.at/144902014625980723213189104 5" xml:
lang="en" id="158731ce-3fc7-4a30-9c8e-2649daa848c9-3" type="result" from="brug
mansia@jabber.at"><query xmlns="jabber:iq:roster" ver="032fa62e7f33e571ebfbcf4
07f9a6198ebba7ce1"><item jid="datura@jabber.at" subscription="both" /><item ji
d="pronnus@jabber.at" subscription="from" ask="subscribe" /></query></iq>
```

Fig. 4.26 Inicio de sesión nodo 2 (parte 6)

```
DEBUG    Event triggered: roster_update
DEBUG    Event triggered: presence
DEBUG    Event triggered: presence_available
DEBUG    Event triggered: got_online
DEBUG    Event triggered: changed_status
DEBUG    Scheduled event: Session timeout check: ()
DEBUG    Scheduled event: Whitespace Keepalive: (' ',)
DEBUG    SEND (IMMED):
```

Fig. 4.27 Inicio de sesión nodo 2 (parte 7)

Si el proceso anterior fue exitoso, en el cliente PC aparecerá el siguiente mensaje (Fig. 4.28) en la parte inferior de la pantalla.

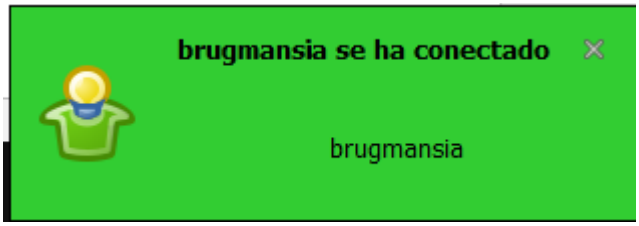


Fig. 4.28 Notificación de inicio de sesión del nodo 2

El mensaje indica que la Raspberry Pi está en línea, para comprobar que la comunicación es exitosa, se enviara el siguiente mensaje de prueba (Fig. 4.29).

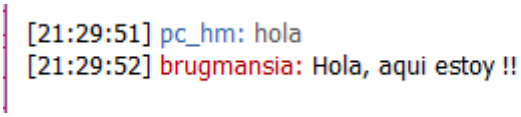


Fig. 4.29 Mensaje de prueba de PC a nodo 2

Cuando se recibe el mensaje en la Raspberry Pi, se genera un evento y la respuesta programada se envía (Fig. 4.30).

```
DEBUG    RECV: <message id="8ad1ec03-9587-44cc-b1de-5638b2db6728" to="brugmans
ia@jabber.at/402590104688973260956379074" type="chat" xml:lang="es" from="pron
nus@jabber.at/gajim.MH7JE29Q"><origin-id xmlns="urn:xmpp:sid:0" id="8ad1ec03-9
587-44cc-b1de-5638b2db6728" /><request xmlns="urn:xmpp:receipts" /><body>hola<
/body><thread>mLflfjzsvUaEDSCccmTSoDmdUExboOIN</thread></message>
DEBUG    Event triggered: message
DEBUG    SEND: <message to="pronnus@jabber.at/gajim.MH7JE29Q" type="chat" xml:
lang="es"><thread>mLflfjzsvUaEDSCccmTSoDmdUExboOIN</thread></message>
DEBUG    SEND: <message to="pronnus@jabber.at/gajim.MH7JE29Q" type="chat" xml:
lang="es"><thread>mLflfjzsvUaEDSCccmTSoDmdUExboOIN</thread><body>Hola, aqui es
toy !!</body></message>
```

Fig. 4.30 Mensaje recibido en nodo 2

4.1.5 El sistema está operativo

Si los procesos de inicio de sesión anteriores fueron exitosos, en el cliente aparecerán ambas Raspberry Pi como disponibles (Fig. 4.31).

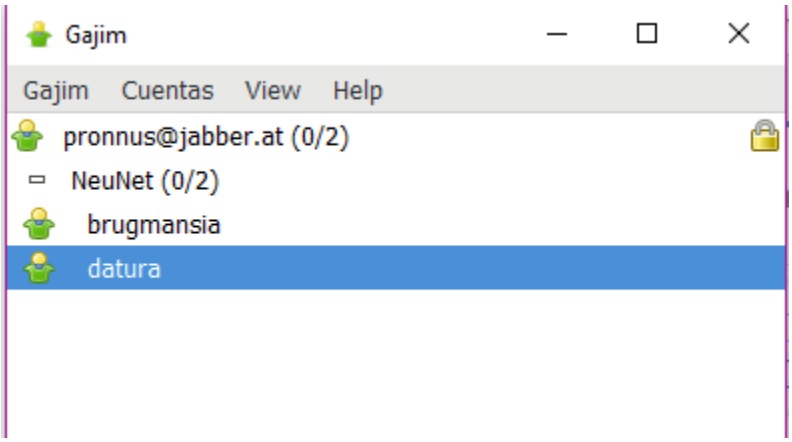


Fig. 4.31 Aplicación Gajim mostrando ambos nodos disponibles

Ahora el sistema está activo y se pueden comenzar las pruebas. El orden de inicio de sesión de los dispositivos no afecta el sistema, siempre y cuando al finalizar estén activos ambos nodos Raspberry Pi y almenos un cliente, ya sea PC o Android.

En el caso de que algún dispositivo cierre la sesión con el servidor, los mensajes se acumularan y los recibirá cuando inicie sesión de nuevo. Los mensajes que le sean enviados cuando no este activo no serán analizados y las instrucciones no serán recibidas.

Se puede saber que un cliente no está disponible porque aparecerá como desconectado (Fig. 4.32).

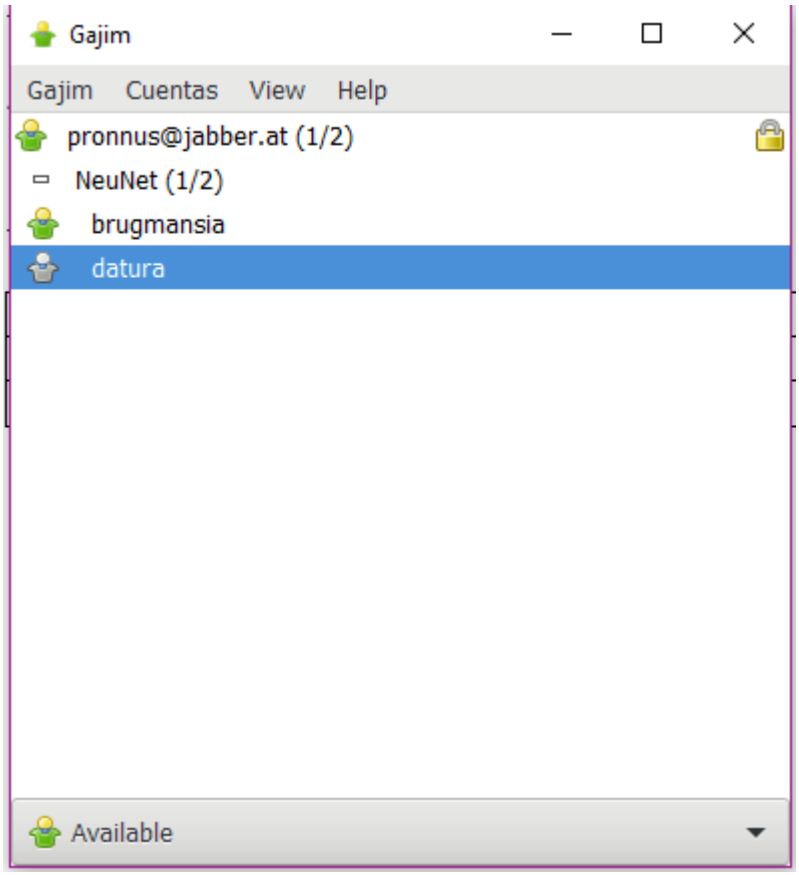


Fig. 4.32 Aplicación Gajim después de que un nodo se desconectó inesperadamente

4.2 Análisis de paquetes y rutas de comunicación.

Una vez activo el sistema, se procederá a analizar los paquetes y las rutas de comunicación.

4.2.1 Análisis de paquetes

Para este análisis se utiliza el software "Wireshark" que capturará los paquetes entrantes y salientes del protocolo XMPP.

En la (Fig. 4.33) se puede notar que los paquetes de comunicación son "TLSv1.2" que es una derivación del protocolo SSL, se puede comprobar que

los paquetes intercambiados por el protocolo XMPP están cifrados de extremo a extremo.

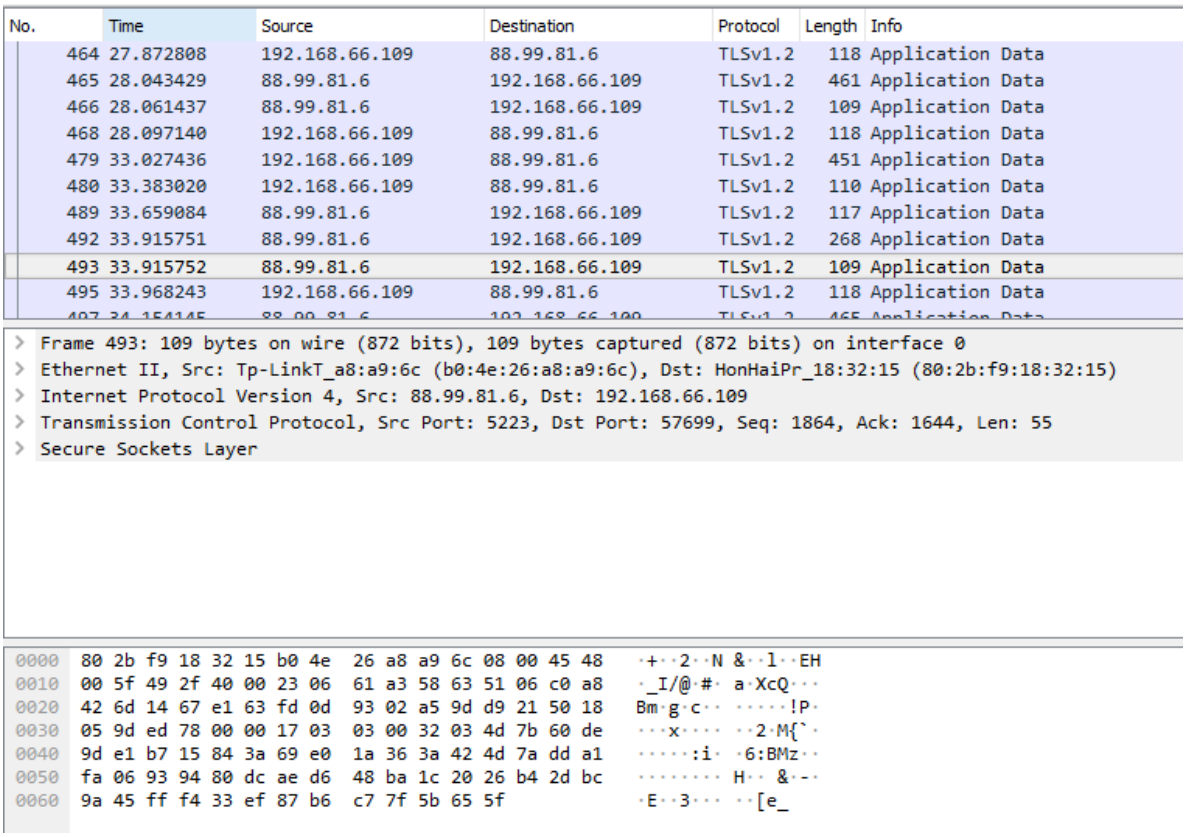


Fig. 4.33 Análisis con Wireshark de paquete recibido (parte 1)

Se analizará otro paquete (Fig. 4.34), es similar al anterior, los datos están cifrados y no se puede visualizar el contenido del mensaje.

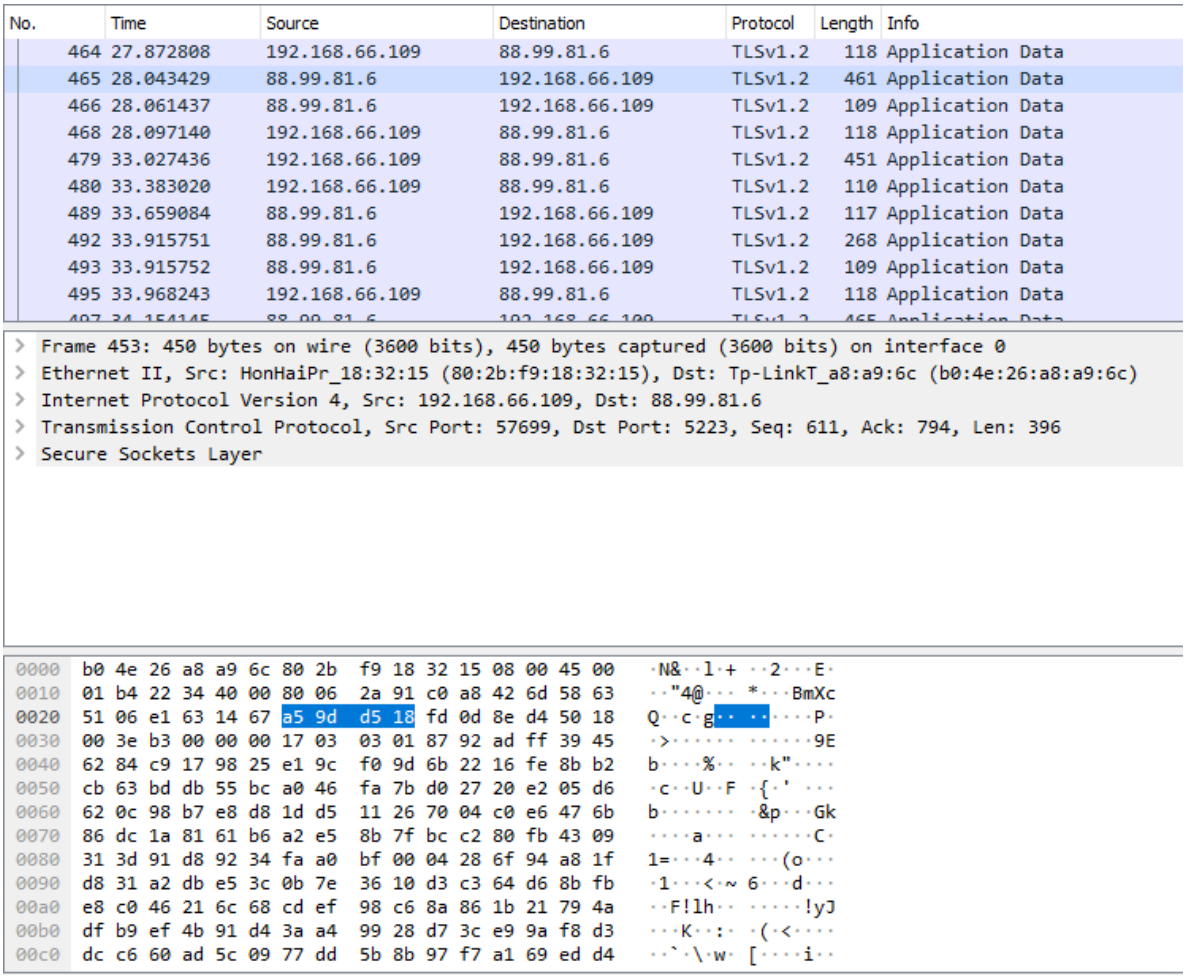


Fig. 4.34 Análisis con Wireshark de paquete recibido (parte 2)

No se hizo énfasis en la dirección ip del servidor pues se está utilizando un servidor público que engloba varios servidores encargados de gestionar las comunicaciones por lo que en cada inicio de sesión la dirección del servidor puede ser diferente.

4.2.2 Rutas de comunicación

Como se mencionó anteriormente la arquitectura de XMPP es cliente servidor y cada comunicación pasa por el servidor. En este proyecto se tienen tres tipos de comunicación posibles.

4.2.2.1 Comunicación directa

La ruta de comunicación mostrada en la (Fig. 4.35) se da cuando un cliente le envía una petición (Ruta en color rojo) a un nodo (Nodo 1) sobre un actuador (Actuador 1, 2, 3) o sensor (Sensor 1) que tiene directamente conectado, por lo que el nodo responde (Ruta en color verde) directamente a la petición.

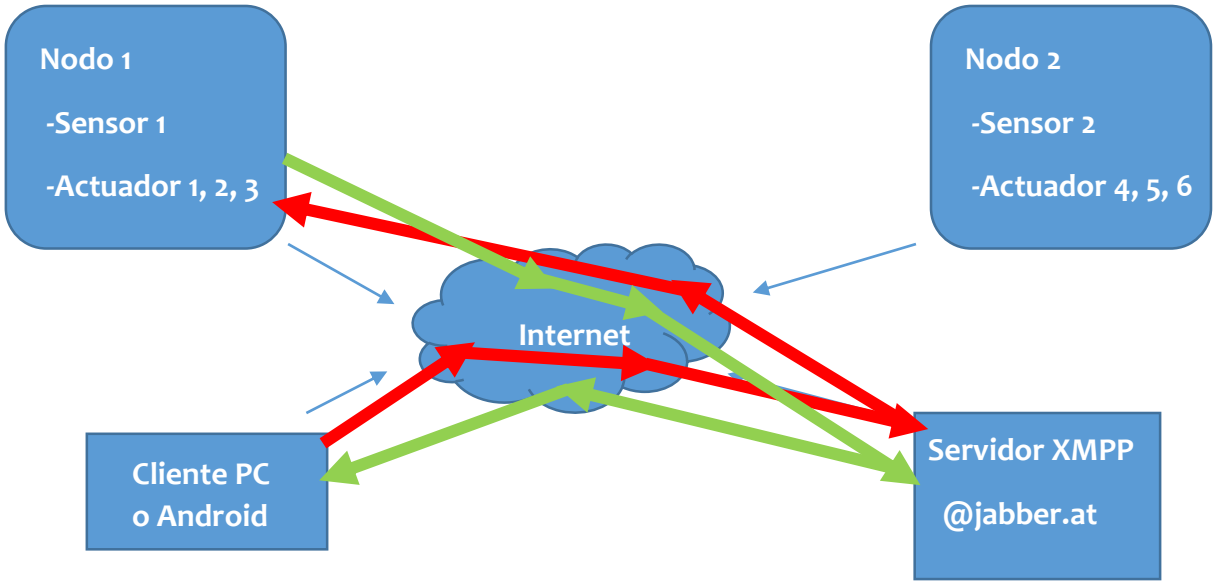


Fig. 4.35 Diagrama de comunicación directa

4.2.2.2 Comunicación redireccionada

La ruta de comunicación mostrada en (Fig. 4.36, Fig. 4.37) se da cuando un cliente le envía una petición (Ruta en color rojo) a un nodo (Nodo 1) sobre un actuador (Actuador 3, 4, 5) o sensor (Sensor 2) que están conectados en otro nodo, por lo que el nodo 1 revisa en su base de contactos buscando información sobre el nodo en el que están conectados (Nodo 2) y lo contacta. La respuesta se genera siguiendo la misma ruta (Ruta en color verde). Si el actuador o sensor solicitado no se encuentre en las bases de datos se responderá al cliente con un mensaje de error.

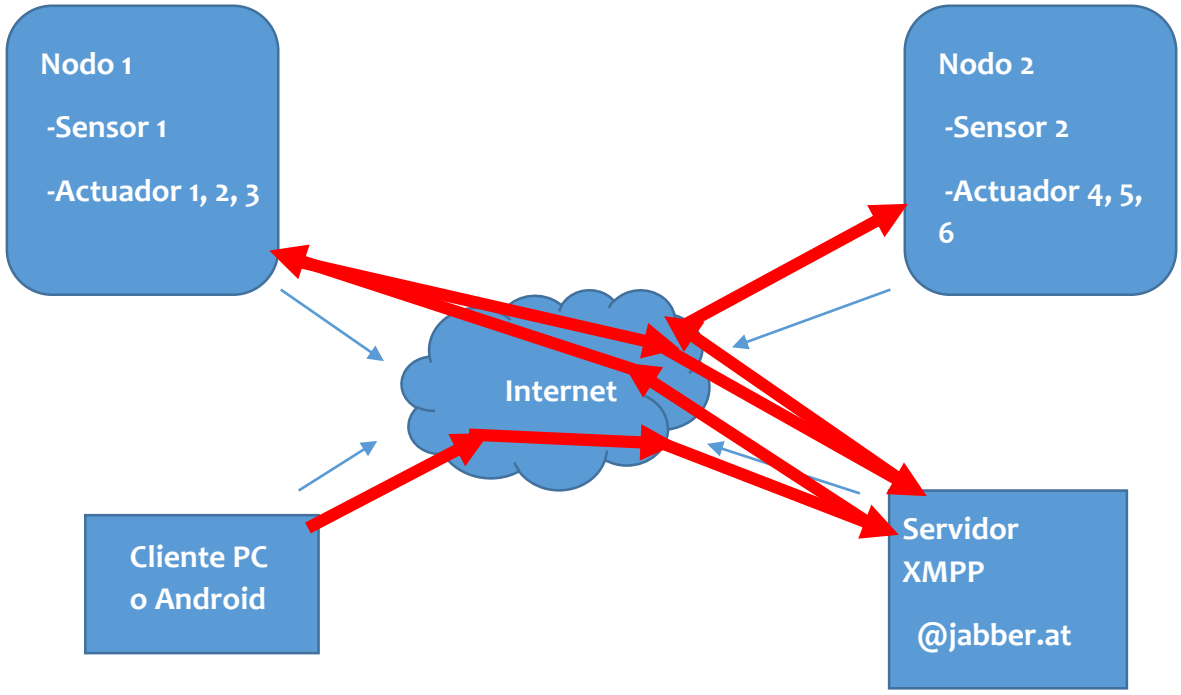


Fig. 4.36 Diagrama de comunicación redireccionada (parte 1)

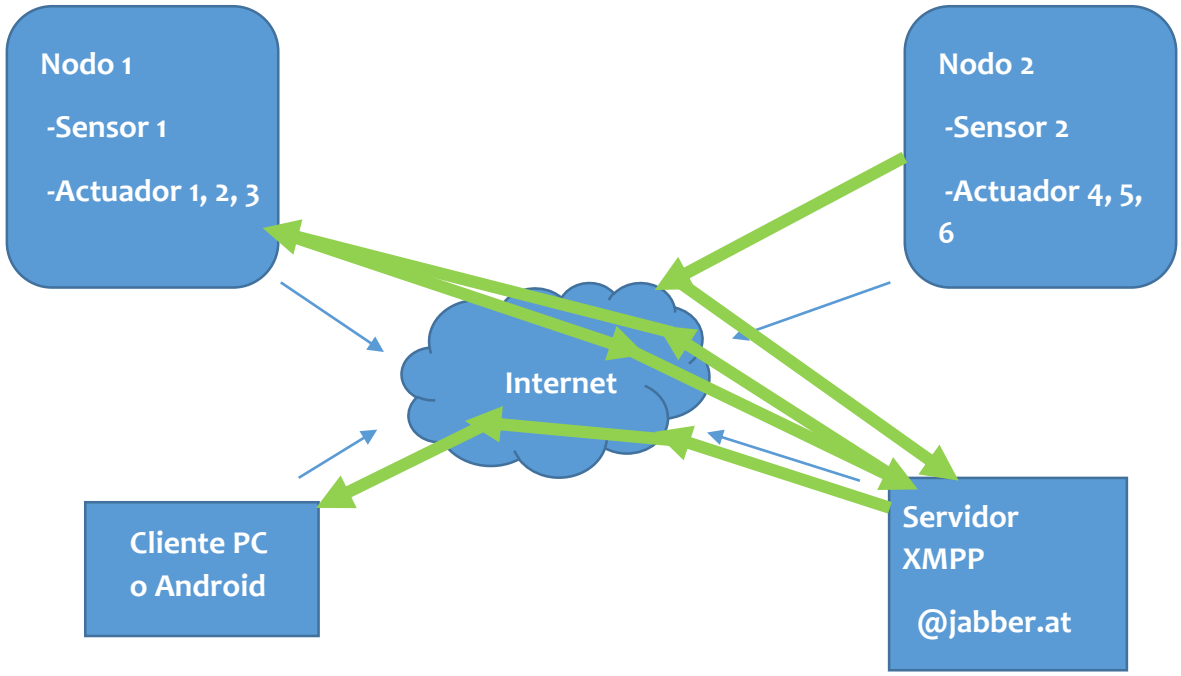


Fig. 4.37 Diagrama de comunicación redireccionada (parte 2)

4.2.2.3 Comunicación con dos o más clientes activos

La ruta de comunicación (Fig. 4.38) se da cuando existen dos o más clientes activos usando la misma cuenta XMPP, si un cliente le envía una petición (Ruta en color rojo) a un nodo (Nodo 1) sobre un actuador (Actuador 1, 2, 3) o sensor (Sensor 1) que tiene directamente conectado, el nodo responde (Ruta en color verde) directamente a la petición, pero desde el servidor la respuesta llega a todos los clientes activos.

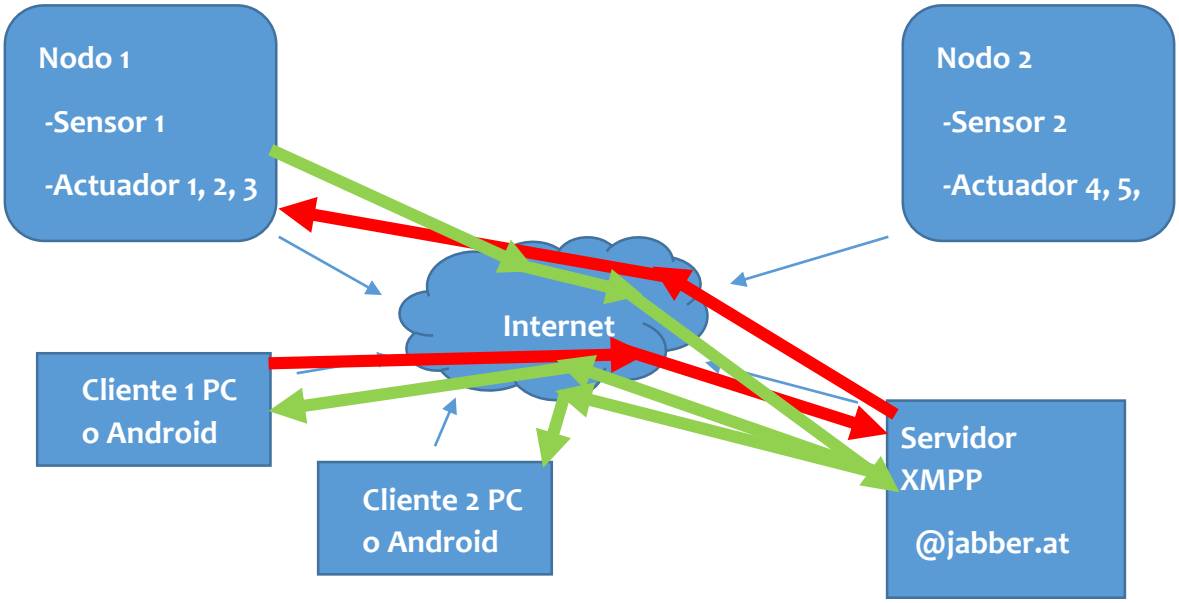


Fig. 4.38 Diagrama de comunicación con dos clientes activos

4.3 Instrucciones y tiempo de respuesta

Ahora se probará el tiempo de respuesta del sistema, desde el momento en que se envía una instrucción hasta recibir la respuesta. El tiempo considerado incluye el tiempo que tarda el mensaje en llegar al dispositivo destino, el tiempo de procesamiento (Lo que tarda el sistema en completar la instrucción que recibe y generar la respuesta) y el tiempo que tarda la respuesta en llegar al usuario que envió la petición.

Se realizaron pruebas de comunicación con los dispositivos en diferentes redes públicas, los nodos conectados a un ISP de la compañía "IZZI" y los clientes desde la red 4G de la compañía "Telcel".

4.3.1 Cliente PC hacia nodo 1

Se probará la comunicación hacia el Nodo 1 desde el cliente PC.

4.3.1.1 Comunicación directa

En este tipo de comunicación se envían mensajes que atenderá directamente el Nodo 1

- Prueba 1

  El saludo es únicamente un indicador de presencia, para confirmar que el nodo está disponible y atiende los mensajes.

  - **Tipo de instrucción:** Saludo
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 1 seg.

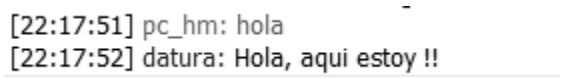


Fig. 4.39 Ejemplo de comunicación (parte 1)

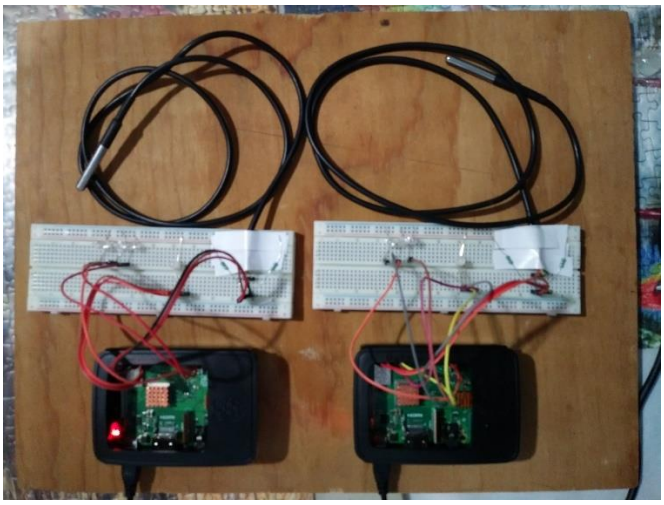
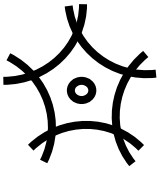


Fig. 4.40 Estado físico del proyecto (parte 1)

- Prueba 2
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Cocina, apagado -> encendido



- **Tiempo de respuesta:** 1 seg.

[22:18:12] pc_hm: encender cocina
[22:18:13] datura: Y ahora ya encendi la luz de la cocina

Fig. 4.41 Ejemplo de comunicación (parte 2)

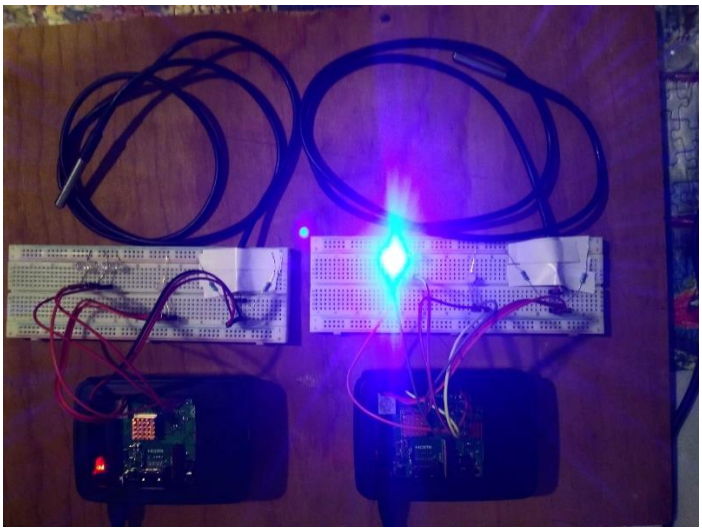


Fig. 4.42 Estado físico del proyecto (parte 2)

- Prueba 3
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Sala, apagado -> encendido
  - **Tiempo de respuesta:** 1 seg.

[22:18:39] pc_hm: encender sala
[22:18:40] datura: Y ahora ya encendi la luz de la sala

Fig. 4.43 Ejemplo de comunicación (parte 3)

- Prueba 4

  En esta instrucción se solicita encender un actuador que esta encendido, por lo que el nodo informa que es imposible realizar la acción y no hace ningún cambio de estado.

  - **Tipo de instrucción:** Encender actuador encendido
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 1 seg.

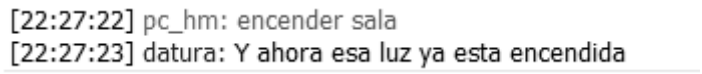


Fig. 4.44 Ejemplo de comunicación (parte 4)

- Prueba 5
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Sala, encendido -> apagado
  - **Tiempo de respuesta:** 1 seg.

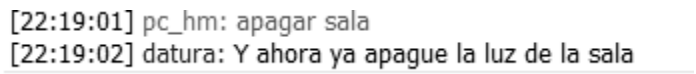


Fig. 4.45 Ejemplo de comunicación (parte 5)

- Prueba 6
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Cocina, encendido -> apagado
  - **Tiempo de respuesta:** 1 seg.

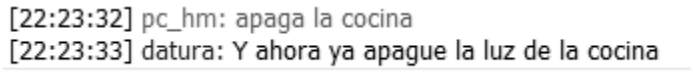


Fig. 4.46 Ejemplo de comunicación (parte 6)

- Prueba 7

  En esta instrucción se solicita apagar un actuador que está apagado, por lo que el nodo informa que es imposible realizar la acción y no hace ningún cambio de estado.

  - **Tipo de instrucción:** Apagar actuador apagado
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 1 seg.

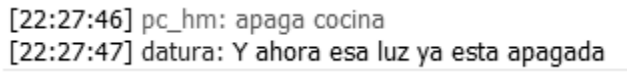
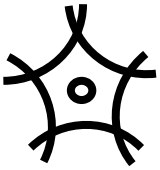


Fig. 4.47 Ejemplo de comunicación (parte 7)

- Prueba 8
  - **Tipo de instrucción:** Solicitar valor de sensor
  - **Cambio de estado:** Ninguno



- **Tiempo de respuesta:** 7 seg.

[22:24:16] pc_hm: que temperatura esta la sala
[22:24:17] datura: Y ahora le informo la temperatura
[22:24:23] datura: Y ahora la temperatura es 19.25 grados centigrados

Fig. 4.48 Ejemplo de comunicación (parte 8)

4.3.1.2 Comunicación redireccionada

En este tipo de comunicación se envían mensajes al Nodo 1, solicitando acciones que corresponden al Nodo 2, por lo que reenviara los mensajes al nodo correspondiente, como se analizó en la sección anterior "Rutas de comunicación".

- Prueba 1
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Baño, apagado -> encendido
  - **Tiempo de respuesta:** 3 seg.

[22:20:47] pc_hm: encender baño
[22:20:50] datura: y ahora ya encendi la luz del baño

Fig. 4.49 Ejemplo de comunicación (parte 9)

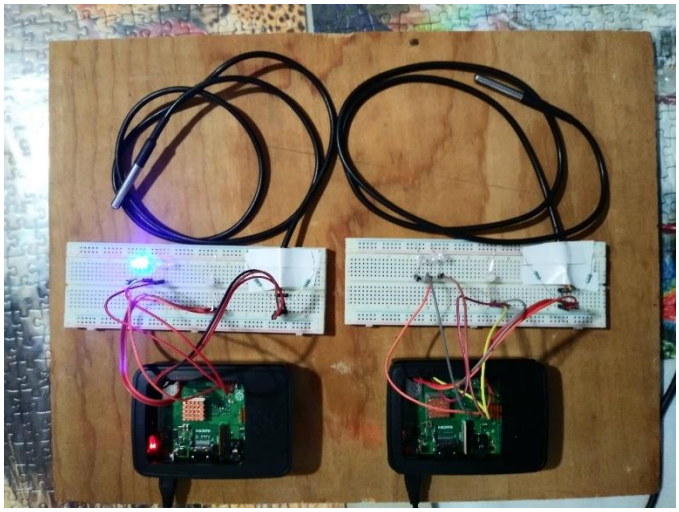


Fig. 4.50 Estado físico del proyecto (parte 3)

- Prueba 2
  - **Tipo de instrucción:** Encender actuador apagado

- **Cambio de estado:** Recamara, apagado -> encendido
- **Tiempo de respuesta:** 3 seg.

[22:21:24] pc_hm: enciende la luz de mi recamara
[22:21:27] datura: y ahora ya encendi la luz de tu recamara

Fig. 4.51 Ejemplo de comunicación (parte 10)

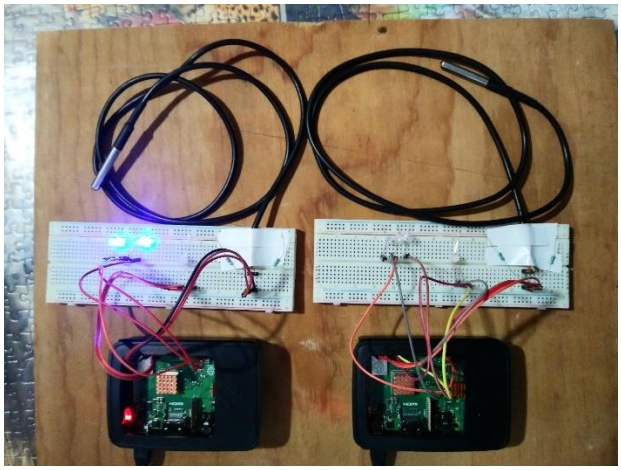


Fig. 4.52 Estado físico del proyecto (parte 4)

- Prueba 3
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Baño, encendido -> apagado
  - **Tiempo de respuesta:** 5 seg.

[22:22:22] pc_hm: apaga la luz del baño porfavor
[22:22:27] datura: y ahora ya apague la luz del baño

Fig. 4.53 Ejemplo de comunicación (parte 11)

- Prueba 4
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Recamara, encendido -> apagado
  - **Tiempo de respuesta:** 4 seg.

[22:22:57] pc_hm: apaga mi recamara
[22:23:01] datura: y ahora ya apague la luz de tu recamara

Fig. 4.54 Ejemplo de comunicación (parte 12)

- Prueba 5

- **Tipo de instrucción:** Solicitar valor de sensor
- **Cambio de estado:** Ninguno
- **Tiempo de respuesta:** 8 seg.

[22:26:44] pc_hm: a que temperatura esta mi recamara
[22:26:46] datura: y ahora le informo la temperatura
[22:26:52] datura: y ahora la temperatura es 19.125 grados centigrados

Fig. 4.55 Ejemplo de comunicación (parte 13)

4.3.2 Cliente PC hacia nodo 2

Se probará la comunicación hacia el Nodo 2 desde el cliente PC.

4.3.2.1 Comunicación directa

En este tipo de comunicación se envían mensajes que atenderá directamente el Nodo 2

- Prueba 1

  El saludo es únicamente un indicador de presencia, para confirmar que el nodo está disponible y atiende los mensajes.

  - **Tipo de instrucción:** Saludo
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 1 seg.

[22:44:03] pc_hm: hola
[22:44:04] brugmansia: Hola, aqui estoy !!

Fig. 4.56 Ejemplo de comunicación (parte 14)

- Prueba 2
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Recamara, apagado -> encendido
  - **Tiempo de respuesta:** 1 seg.

[22:44:27] pc_hm: enciende mi recamara
[22:44:28] brugmansia: Y ahora ya encendi la luz de tu recamara

Fig. 4.57 Ejemplo de comunicación (parte 15)

- Prueba 3
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Recamara 2, apagado -> encendido
  - **Tiempo de respuesta:** 1 seg.

[22:45:10] pc_hm: enciende la otra recamara tambien
[22:45:11] brugmansia: Y ahora ya encendi la luz de esa recamara

Fig. 4.58 Ejemplo de comunicación (parte 16)

- Prueba 4

  En esta instrucción se solicita encender un actuador que esta encendido, por lo que el nodo informa que es imposible realizar la acción y no hace ningún cambio de estado.

  - **Tipo de instrucción:** Encender actuador encendido
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 1 seg.

[22:45:51] pc_hm: enciende la otra recamara
[22:45:52] brugmansia: Y ahora esa luz ya esta encendida

Fig. 4.59 Ejemplo de comunicación (parte 17)

- Prueba 5
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Recamara 2, encendido -> apagado
  - **Tiempo de respuesta:** 1 seg.

[22:46:14] pc_hm: apaga la luz de la otra recamara
[22:46:15] brugmansia: Y ahora ya apague la luz de esa recamara

Fig. 4.60 Ejemplo de comunicación (parte 18)

- Prueba 6

  En esta instrucción se solicita apagar un actuador que está apagado, por lo que el nodo informa que es imposible realizar la acción y no hace ningún cambio de estado.

  - **Tipo de instrucción:** Apagar actuador apagado

- **Cambio de estado:** Ninguno
- **Tiempo de respuesta:** 1 seg.

[22:46:41] pc_hm: apaga la otra recamara
[22:46:42] brugmansia: Y ahora esa luz ya esta apagada

Fig. 4.61 Ejemplo de comunicación (parte 19)

- Prueba 7
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Recamara, encendido -> apagado
  - **Tiempo de respuesta:** 1 seg.

[22:47:01] pc_hm: apaga mi recamara porfaz
[22:47:02] brugmansia: Y ahora ya apague la luz de tu recamara

Fig. 4.62 Ejemplo de comunicación (parte 20)

- Prueba 8
  - **Tipo de instrucción:** Solicitar valor de sensor
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 5 seg.

[22:48:18] pc_hm: que temperatura esta el baño
[22:48:19] brugmansia: Y ahora le informo la temperatura
[22:48:24] brugmansia: Y ahora la temperatura es 19.25 grados centigrados

Fig. 4.63 Ejemplo de comunicación (parte 21)

4.3.2.2 Comunicación redireccionada

En este tipo de comunicación se envían mensajes al Nodo 2, solicitando acciones que corresponden al Nodo 1, por lo que reenviara los mensajes al nodo correspondiente, como se analizó en la sección anterior "Rutas de comunicación".

- Prueba 1
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Comedor, apagado -> encendido
  - **Tiempo de respuesta:** 2 seg.

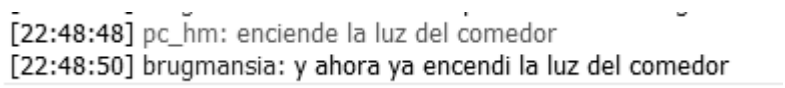


Fig. 4.64 Ejemplo de comunicación (parte 22)

- Prueba 2
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Sala, apagado -> encendido
  - **Tiempo de respuesta:** 3 seg.

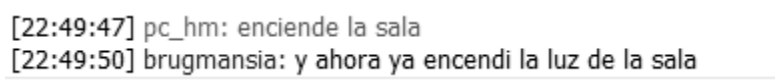


Fig. 4.65 Ejemplo de comunicación (parte 23)

- Prueba 3
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Sala, encendido -> apagado
  - **Tiempo de respuesta:** 3 seg.

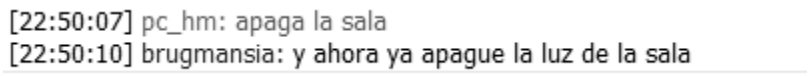


Fig. 4.66 Ejemplo de comunicación (parte 24)

- Prueba 4
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Comedor, encendido -> apagado
  - **Tiempo de respuesta:** 5 seg.

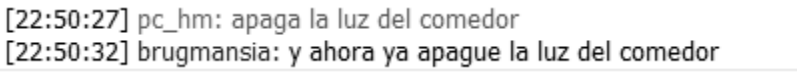


Fig. 4.67 Ejemplo de comunicación (parte 25)

- Prueba 5
  - **Tipo de instrucción:** Solicitar valor de sensor
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 9 seg.

[22:53:19] pc_hm: que temperatura esta la cocina
[22:53:23] brugmansia: y ahora le informo la temperatura
[22:53:28] brugmansia: y ahora la temperatura es 19.312 grados centigrados

Fig. 4.68 Ejemplo de comunicación (parte 26)

4.3.3 Cliente PC en comunicación mixta

En este tipo de comunicación la primera instrucción (Prueba 1) se envía al Nodo 1 que atenderá directamente la petición. La segunda instrucción (Prueba 2) se envía al Nodo 2, pero corresponde al Nodo 1, por lo que el Nodo 2 la reenviara.

- Prueba 1
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Sala, apagado -> encendido
  - **Tiempo de respuesta:** 1 seg.

[22:55:05] pc_hm: encender sala
[22:55:06] datura: Y ahora ya encendi la luz de la sala

Fig. 4.69 Ejemplo de comunicación (parte 27)

- Prueba 2
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Sala, encendido -> apagado
  - **Tiempo de respuesta:** 3 seg.

[22:55:24] pc_hm: apagar sala
[22:55:27] brugmansia: y ahora ya apague la luz de la sala

Fig. 4.70 Ejemplo de comunicación (parte 28)

4.3.4 Cliente Android hacia nodo 1

Se probará la comunicación hacia el Nodo 1 desde el cliente Android.

4.3.4.1 Conexión directa

En este tipo de comunicación se envían mensajes que atenderá directamente el Nodo 1

- Prueba 1

  El saludo es únicamente un indicador de presencia, para confirmar que el nodo está disponible y atiende los mensajes.

  - **Tipo de instrucción:** Saludo
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 1 seg.

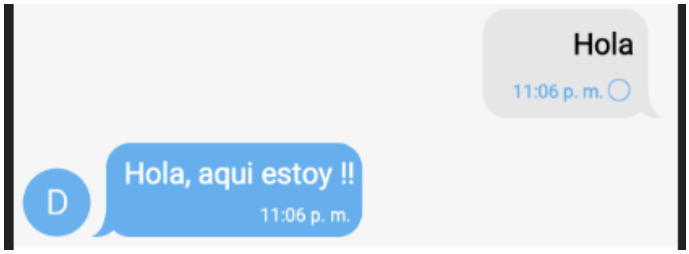


Fig. 4.71 Ejemplo de comunicación (parte 29)

- Prueba 2
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Sala, apagado -> encendido
  - **Tiempo de respuesta:** 1 seg.

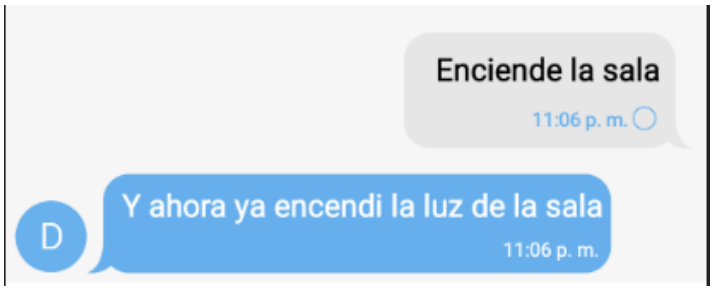


Fig. 4.72 Ejemplo de comunicación (parte 30)

- Prueba 3
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Sala, encendido -> apagado
  - **Tiempo de respuesta:** 1 seg.

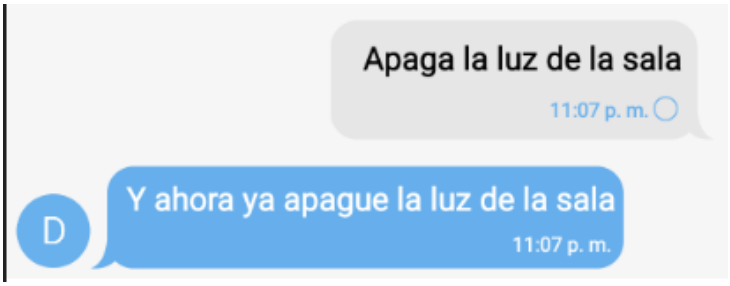


Fig. 4.73 Ejemplo de comunicación (parte 31)

- Prueba 4
  - **Tipo de instrucción:** Solicitar valor de sensor
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 7 seg.

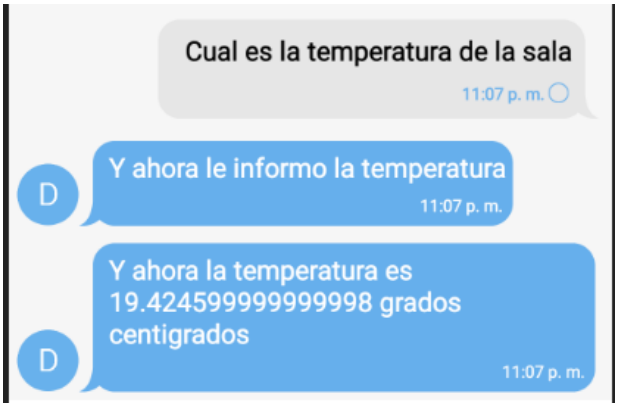


Fig. 4.74 Ejemplo de comunicación (parte 32)

4.3.4.2 Conexión redireccionada

En este tipo de comunicación se envían mensajes al Nodo 1, solicitando acciones que corresponden al Nodo 2, por lo que reenviara los mensajes al nodo correspondiente, como se analizó en la sección anterior "Rutas de comunicación".

- Prueba 1
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Baño, apagado -> encendido
  - **Tiempo de respuesta:** 3 seg.

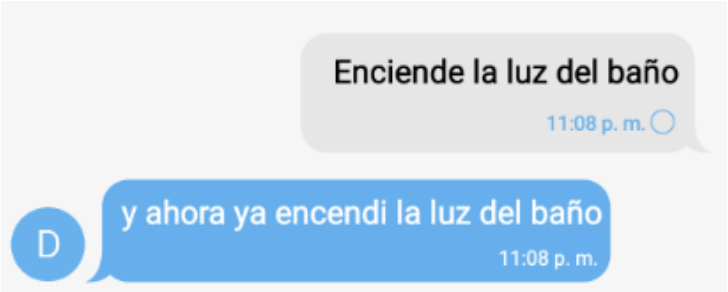


Fig. 4.75 Ejemplo de comunicación (parte 33)

- Prueba 2
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Baño, encendido -> apagado
  - **Tiempo de respuesta:** 3 seg.

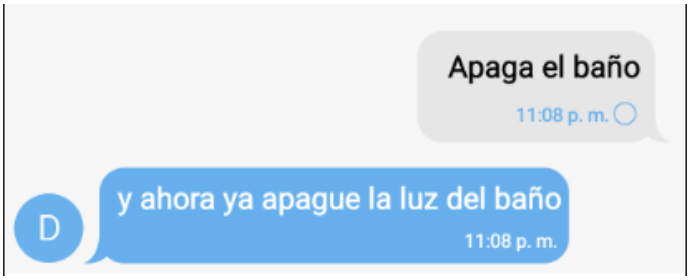


Fig. 4.76 Ejemplo de comunicación (parte 34)

- Prueba 3
  - **Tipo de instrucción:** Solicitar valor de sensor
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 9 seg.

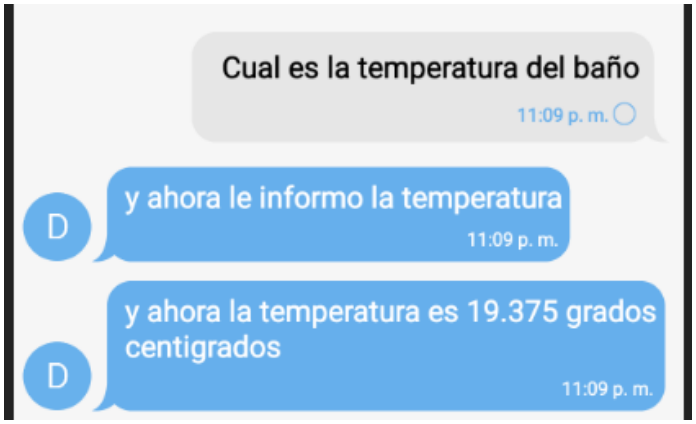
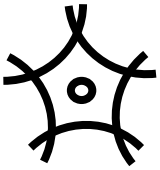


Fig. 4.77 Ejemplo de comunicación (parte 35)



4.3.5 Cliente Android hacia nodo 2

Se probará la comunicación hacia el Nodo 1 desde el cliente Android.

4.3.5.1 Conexión directa

En este tipo de comunicación se envían mensajes que atenderá directamente el Nodo 2

- Prueba 1

  El saludo es únicamente un indicador de presencia, para confirmar que el nodo está disponible y atiende los mensajes.

  - **Tipo de instrucción:** Saludo
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 1 seg.

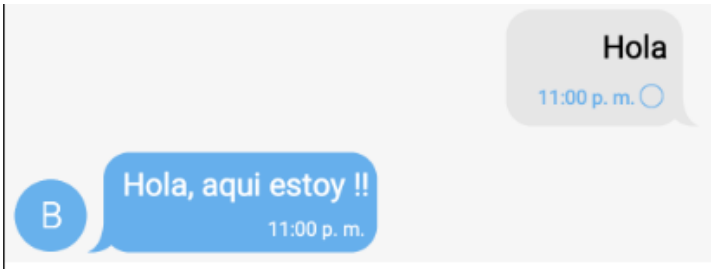


Fig. 4.78 Ejemplo de comunicación (parte 36)

- Prueba 2
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Baño, apagado -> encendido
  - **Tiempo de respuesta:** 1 seg.

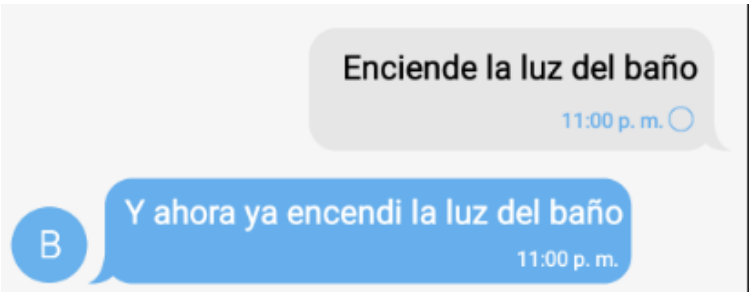


Fig. 4.79 Ejemplo de comunicación (parte 37)

- Prueba 3
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Recamara, apagado -> encendido
  - **Tiempo de respuesta:** 1 seg.

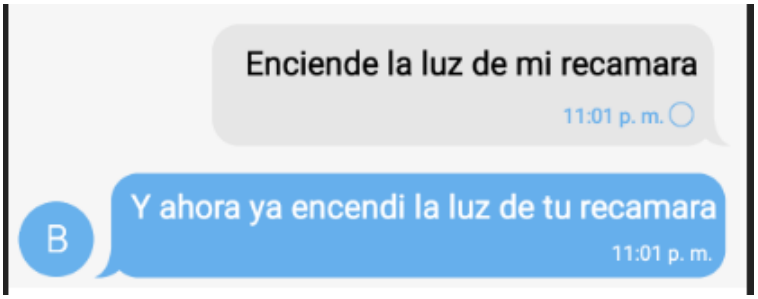


Fig. 4.80 Ejemplo de comunicación (parte 38)

- Prueba 4
  - **Tipo de instrucción:** Solicitar valor de sensor
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 7 seg.

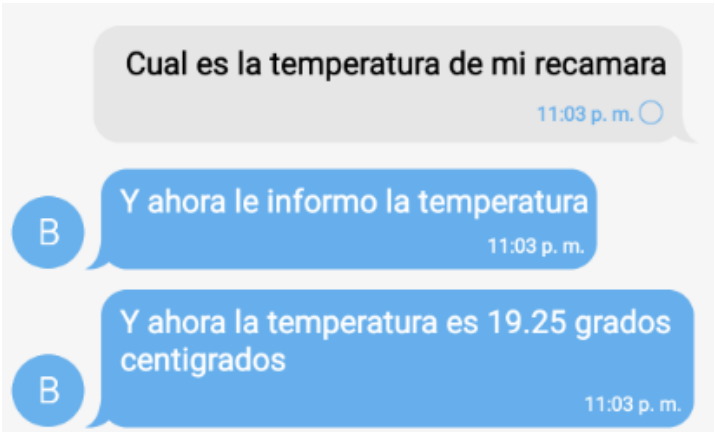


Fig. 4.81 Ejemplo de comunicación (parte 39)

4.3.5.2 Conexión redireccionada

En este tipo de comunicación se envían mensajes al Nodo 2, solicitando acciones que corresponden al Nodo 1, por lo que reenviara los mensajes al nodo correspondiente, como se analizó en la sección anterior "Rutas de comunicación".

- Prueba 1
  - **Tipo de instrucción:** Encender actuador apagado
  - **Cambio de estado:** Cocina, apagado -> encendido
  - **Tiempo de respuesta:** 3 seg.

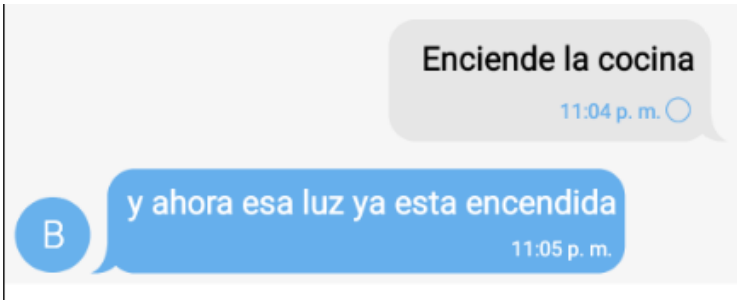


Fig. 4.82 Ejemplo de comunicación (parte 40)

- Prueba 2
  - **Tipo de instrucción:** Apagar actuador encendido
  - **Cambio de estado:** Cocina, encendido -> apagado
  - **Tiempo de respuesta:** 3 seg.

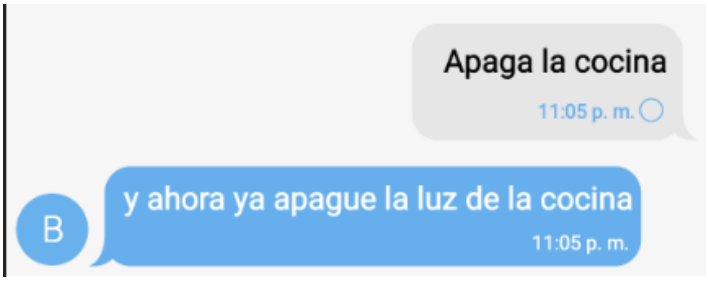


Fig. 4.83 Ejemplo de comunicación (parte 41)

- Prueba 3
  - **Tipo de instrucción:** Solicitar valor de sensor
  - **Cambio de estado:** Ninguno
  - **Tiempo de respuesta:** 11 seg.

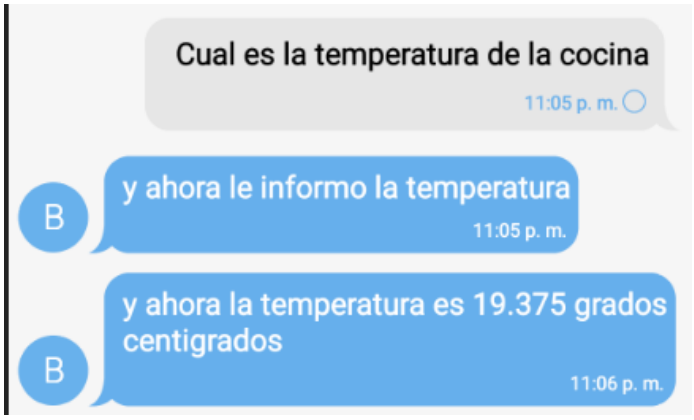


Fig. 4.84 Ejemplo de comunicación (parte 42)

4.3.6 Tabla de resultados

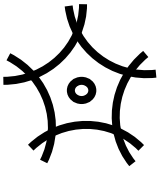
En la Tabla 4 se muestra un resumen de los resultados obtenidos.

Origen	Destino directo	Destino indirecto	Instrucción	Repeticiones	Tiempo Promedio
PC	Nodo 1	-	Saludo	40	1 seg
PC	Nodo 1	-	Encender/Apagar Actuador	100	1 seg
PC	Nodo 1	-	Solicitar Datos	60	6 seg
PC	Nodo 1	Nodo 2	Encender/Apagar Actuador	100	3 seg
PC	Nodo 1	Nodo 2	Solicitar Datos	60	8 seg
PC	Nodo 2	-	Saludo	40	1 seg
PC	Nodo 2	-	Encender/Apagar Actuador	100	1 seg
PC	Nodo 2	-	Solicitar Datos	60	6 seg
PC	Nodo 2	Nodo 1	Encender/Apagar Actuador	100	3 seg
PC	Nodo 2	Nodo 1	Solicitar Datos	60	8 seg

Android	Nodo 1	-	Saludo	40	1 seg
Android	Nodo 1	-	Encender/Apagar Actuador	100	1 seg
Android	Nodo 1	-	Solicitar Datos	60	6 seg
Android	Nodo 1	Nodo 2	Encender/Apagar Actuador	100	3 seg
Android	Nodo 1	Nodo 2	Solicitar Datos	60	8 seg
Android	Nodo 2	-	Saludo	40	1 seg
Android	Nodo 2	-	Encender/Apagar Actuador	100	1 seg
Android	Nodo 2	-	Solicitar Datos	60	6 seg
Android	Nodo 2	Nodo 1	Encender/Apagar Actuador	100	3 seg
Android	Nodo 2	Nodo 1	Solicitar Datos	60	8 seg

Tabla 4.1 Tiempos de respuesta

5. CONCLUSIONES



Conclusiones

Después de concluir la implementación y las pruebas del sistema, resalta la eficiencia del protocolo XMPP, con una comunicación prácticamente en tiempo real como se puede cotejar en la (Tabla 5.1) que muestra de forma resumida los resultados obtenidos en las pruebas del "tiempo de respuesta del sistema" mostrados en la (Tabla 4.1).

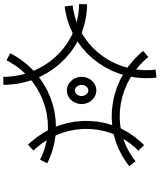
Tipo de comunicación	Tipo de instrucción	Tiempo Promedio
Directo	Saludo	1 seg
	Encender/Apagar Actuador	1 seg
	Solicitar Datos	6 seg
Redireccionada	Saludo	-
	Encender/Apagar Actuador	3 seg
	Solicitar Datos	8 seg

Tabla 5.1 Resumen de tiempos de respuesta

Analizando las rutas de comunicación, se comprobó que es un protocolo con arquitectura cliente-servidor, pues todos los mensajes de la comunicación tienen que pasar por el servidor antes de ser entregados el destinatario, la ventaja de esta arquitectura es que se evita el problema de firewalls y se puede crear una comunicación entre dispositivos conectados a internet sin importar su ubicación física o a que proveedor de servicios estén conectados. En este trabajo se utilizaron dos proveedores diferentes "izzi" y "Telcel" simultáneamente para realizar las pruebas.

Una ventaja más del protocolo es que la comunicación está cifrada de extremo a extremo como se comprobó al analizar con Wireshark los paquetes XMPP intercambiados en la comunicación, este cifrado agrega seguridad al sistema y dificulta que personas no autorizadas visualicen el contenido de los mensajes intercambiados.

En la implementación del protocolo, se cuenta con gran cantidad de información de ayuda disponible, además, existen servidores que ofrecen



registro gratuito de cuentas, como los mostrados en la página oficial de XMPP (Fig. 5.1). Las cuentas se pueden crear en pocos minutos y ofrecen todos los beneficios del protocolo.

- xmpp.jp
- jabber.ru
- JWChat
- jabber.at
- yax.im

Fig. 5.1 Servidores XMPP gratuitos

Adicionalmente, con XMPP existe la posibilidad de crear un servidor propio, algunos programas creados para este propósito se pueden consultar en la página oficial de XMPP (Fig. 5.2).

Nombre del Proyecto	Sistema Operativo
AstraChat	Linux / macOS / Solaris / Windows
ejabberd	Linux / macOS / Windows
IoT Broker	Windows
Isode M-Link	Linux / Windows
Metronome IM	Linux
MongooseIM	Linux / macOS
Openfire	Linux / macOS / Solaris / Windows
Prosody IM	BSD / Linux / macOS
Tigase XMPP Server	Linux / macOS / Solaris / Windows

Fig. 5.2 Lista de programas servidor xmpp

También se pudo comprobar la compatibilidad del protocolo con pequeños dispositivos usados en IoT, como es la Raspberry Pi, pues el protocolo no demanda una gran cantidad de recursos computacionales y existen bibliotecas con funciones XMPP para la mayoría de lenguajes de

programación lo que permite compatibilidad con la mayoría de sistemas (Fig. 5.3).

Nombre del Proyecto	Lenguaje de Programación
aioxmpp	Python
Babbler	Java
Erlang/Elixir XMPP	Elixir / Erlang
gloox	C++
MatriX	.net / C# / Mono
QXmpp	C++
Slixmpp	Python
Smack	Java (Java SE and Android)
Stroke	Java
Strophe.js	JavaScript
Swiften	C++
Tigase JaXMPP	Android / Google Web Toolkit / Java
Waher Networking	.NET Core / .NET Standard / C#

Fig. 5.3 Lista de librerías xmpp

Además de bibliotecas, existen programas clientes gratuitos, en la página oficial de XMPP se encuentra una lista de clientes y el sistema operativo para el que fueron diseñados (Fig. 5.4).

Nombre del Proyecto	Sistema Operativo
AstraChat	Android / iOS / Linux / macOS / Windows
Bruno the Jabber™ Bear	Android
Conversations	Android
Converse	Browser
Dino	Linux
Gajim	Linux / Windows
Jitsi Desktop	Linux / macOS / Windows
JSXC	Browser
Kaidan	Android / iOS / Linux / macOS / Other / Windows

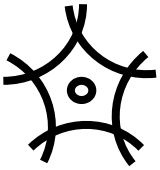
Fig. 5.4 Lista de clientes XMP.

Finalmente, se debe mencionar que la mayor ventaja que proporciona el protocolo XMPP es que es libre, se puede hacer uso del mismo sin necesidad de pagar por una licencia.

5.1 Posibilidades para trabajos a futuro

- Poner a prueba redes con mayor cantidad de nodos y clientes activos.
- Implementar el protocolo en otras tarjetas de desarrollo similares a la Raspberry Pi.
- Darle un uso diferente a este tipo de redes además de la domótica, como puede ser monitoreo de instalaciones industriales, seguridad, etc.
- Implementar algoritmos de inteligencia artificial en este tipo de redes para reconocer patrones de comportamiento.
- Implementar servidores XMPP y de análisis de datos propios con características personalizadas.

6. BIBLIOGRAFIA



Bibliografía

[1] R. L. A. F. Michael Blackstock, Uniting online social networks with places and things, San Francisco, California, USA: WoT '11 Proceedings of the Second International Workshop on Web of Things, Article No. 5 , 2011.

[2] Z. Y. X. Z. D. Z. Bin Guo, Opportunistic IoT: Exploring the Social Side of the Internet of Things, 16th International Conference on Computer Supported Cooperative Work in Design, 2012.

[3] P. C. F. V.-G. J. A.-Z. Vasileios Karagiannis, A Survey on Application Layer Protocols for the Internet of Things, Transaction on IoT and Cloud Computing 2015, 2015.

[4] R. Klauck y M. Kirsche, Chatty things - Making the Internet of Things readily usable for the masses with XMPP, Pittsburgh, PA, USA: 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012.

[5] T. S. D. S. A. S. R. A. M. A. Sven Bendel, A Service Infrastructure for the Internet of Things based on XMPP, San Diego: Work in Progress session at PerCom 2013, 2013.

[6] D. T. B. P. B. A. L. R. T. a. M. A. S. Conzon, The VIRTUS Middleware: An XMPP Based Architecture for Secure IoT Communications, Munich, Germany: 2012 21st International Conference on Computer Communications and Networks (ICCCN), 2012.

[7] R. W. Adrian Hornsby, From instant messaging to cloud computing, an XMPP review, Braunschweig, Germany: IEEE International Symposium on Consumer Electronics (ISCE 2010), 2010.

[8] D. J. Á. N. Arnaldos, Sistema de Diálogo Basado en Mensajería Instantánea para el Control de Dispositivos en el Internet de las

cosas, Universidad de Murcia: Tesis Doctoral, Departamento de Informatica y Sistemas, 2016.

[9] Ramírez, Jesús A. Romualdo, Enrique Mendéz Franco, and David Tinoco Varela. "Fuzzification of facial movements to generate human-machine interfaces in order to control robots by XMPP internet protocol." *MATEC Web of Conferences*. Vol. 125. EDP Sciences, 2017.

[10] D. (. Jun Kyun Choi y D. (. Kyu Yeong Jeon, Method and System for Controlling internet of Things (IoT) Device, Korea Advanced Institute of Science and Technology, Daejeon (KR) , 2017.

[11] «Documentacion de Python,» Python.org, [En línea]. Available: https://www.python.org/doc/. [Último acceso: 21 Enero 2019].

[12] xmpp.org, «xmpp About,» [En línea]. Available: https://xmpp.org/. [Último acceso: 2018 Agosto 15].

[13] K. S. a. R. T. Peter Saint-Andre, XMPP: The Definitive Guide, Building Real-Time Applications with Jabber, United States of America, Gravenstein Highway North, Sebastopol: O'REILLY, 2009.

[14] SleekXMPP.org, «SleekXMPP Documentation,» [En línea]. Available: http://sleekxmpp.com/index.html. [Último acceso: 12 Agosto 2018].

[15] J. PASTOR, «Raspberry Pi 3 Model B+, análisis: más potencia y mejor WiFi para un miniPC que sigue asombrando,» xataka, 16 Abril 2018. [En línea]. Available: https://www.xataka.com/ordenadores/raspberry-pi-3-model-b-analisis-mas-potencia-y-mejor-wifi-para-un-minipc-que-sigue-asombrando. [Último acceso: 23 Diciembre 2018].

[16] M. Santos, «Raspberry Pi 3 Model B+: ¿cómo rinde respecto a un procesador de Intel?,» Hardzone, 2 Abril 2018. [En línea]. Available: https://hardzone.es/2018/04/02/raspberry-pi-3-model-b-plus-benchmarks/. [Último acceso: 17 Diciembre 2018].

[17] G. Halfacree, «Benchmarking the Raspberry Pi 3 B+,» Medium Technology, 13 Marzo 2018. [En línea]. Available: https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-3-b-plus-44122cf3d806. [Último acceso: 8 Octubre 2018].

[18] C. d. s. i. i. e. p. d. s. d. d. e. f. e. i. I. D. Luis Delgado, «Curso de privacidad y protección de comunicaciones digitales,» Criptored, 21 Marzo 2013. [En línea]. Available: http://www.criptored.upm.es/crypt4you/temas/privacidad-proteccion/leccion3/leccion3.html. [Último acceso: 2018 Noviembre 3].

[19] E. M. N. D. P. O. D. S. Z.-Y. S. a. C. W. Raymond B. Jennings III, A Study of Internet Instant Messaging and Chat Protocols, Watson Research Center: IBM T.J, 2016.

[20] B. I. A. B. a. F. M. Tom, INSTANT MESSAGING: STANDARDS, PROTOCOLS, APPLICATIONS, AND RESEARCH DIRECTIONS, University of Khartoum - Sudan: Mathematical Sciences and Information Technology Research Unit (MITRU), Faculty of Mathematical Sciences, 2009.

[21] M. W. M. W. S. M. M.U. Farooq, A Critical Analysis on the Security Concerns of Internet of Things (IoT), International Journal of Computer Applications , 2015.

[22] J. B. W. a. M. P. Teng Xu, Security of IoT Systems: Design Challenges and Opportunities, Los Angeles: University of California.

[23] RaspberryPi.org, «Raspberry Pi Documentation,» [En línea]. Available: https://www.raspberrypi.org/. [Último acceso: 13 Agosto 2018].

[24] CISCO, «CISCO - Internet of Things (IoT),» [En línea]. Available: https://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html#~stickynav=4. [Último acceso: 2018 Diciembre 23].

[25] D. Evans, Internet de las cosas Cómo la próxima evolución de Internet lo cambia todo, Cisco Internet Business Solutions Group (IBSG), 2011.

[26] R. K. a. D. C. Neil Gershenfeld, The Internet of Things, SCIENTIFIC AMERICAN.

7. ANEXOS

7.1 Glosario

actuadores
Un ACTUADOR es un dispositivo inherentemente mecánico cuya función es proporcionar fuerza para mover o "actuar" otro dispositivo mecánico. La fuerza que provoca el actuador proviene de tres fuentes posibles
Presión neumática, presión hidráulica, y fuerza motriz eléctrica (motor eléctrico o solenoide).

cliente
Un cliente de red o cliente software, en una red de computadoras, es la entidad de software que realiza las peticiones de servicio a los proveedores del mismo.

código abierto
La idea del código abierto se centra en la premisa de que al compartir el código, el programa resultante tiende a ser de calidad superior al software propietario, es una visión técnica.

consultas
Su funcion es recuperar datos de la fuente que los produce o los almacena.

nodos de red
En redes de computadoras cada una de las máquinas es un nodo, y si la red es Internet, cada servidor constituye también un nodo.

nodos IoT
Dispositivo de gestión ubicado en el edificio que concentra las medidas de los diferentes sensores, las procesa y envía a la plataforma.

paradigma
Es una propuesta tecnológica adoptada por una comunidad de programadores, y desarrolladores cuyo núcleo central es incuestionable en cuanto que únicamente trata de resolver uno o varios problemas claramente delimitados.

protocolos
Un protocolo de red designa el conjunto de reglas que rigen el intercambio de información.

retardo
Retardo de propagación, en lo que a la electrónica se refiere, es el tiempo que tarda una señal para atravesar un canal.

servidor
Un servidor de red es un ordenador que ofrece el acceso a los recursos compartidos entre las estaciones de trabajo u otros servidores conectados en una red informática. Los recursos compartidos pueden incluir acceso a hardware, como discos duros, impresoras, etc, sofware, servicios de email o acceso a internet.

tiempo real
Un sistema en tiempo real es aquel sistema digital que interactúa activamente con un entorno con dinámica conocida en relación con sus entradas, salidas y restricciones temporales.

topología
Las topologías pueden ser de dos tipos
a) Topología física

Se refiere al diseño actual del medio de transmisión de la red. b) Topología lógica

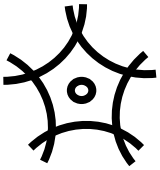
Se refiere a la trayectoria lógica que una señal a su paso por los nodos de la red.

variables físicas

Una variable es algo que cambia respecto a algo, Una VARIABLE FÍSICA es la magnitud que puede influir en el estado de un sistema físico. Por ejemplo

peso, velocidad, fuerza, etc.

7.2 Lista de figuras



7.3 Lista de tablas