Proyecto de Grado

Presentado ante la ilustre Universidad de Los Andes como requisito parcial para obtener el Título de Ingeniero de Sistemas

Diseño e implementación de un sistema de monitoreo de redes orientado a la recolección masiva de datos.

Por

Br. Jesús Alberto Gómez Pérez

Tutor: Dr. Andrés Arcia-Moret

Febrero 2016

Diseño e implementación de un sistema de monitoreo de redes orientado a la recolección masiva de datos.

Br. Jesús Alberto Gómez Pérez

Proyecto de Grado — Sistemas Computacionales, 157 páginas

Resumen: En el presente proyecto se plantea el desarrollo de un sistema de monitoreo distribuido de enlaces de redes a través de un servicio web centralizado. Este servicio además hace énfasis en la visualización de los datos recolectados a partir de pruebas periódicas. El sistema está planteado como una herramienta para facilitar el entendimiento del funcionamiento de la red y ofrecer una solucion de poco impacto computacional, de bajo costo y que pueda operar en componentes de hardware desatendidos.

Palabras clave: Monitoreo de redes, Calidad de servicio, Big Data, Benchmarking, Cloud

Índice

Capítulo 1

Introducción

Con el objetivo de proveer alternativas para el acceso a Internet a bajo costo se han popularizado despliegues de redes alternativos tales como las Redes Comunitarias[1]. Estas redes consisten en arreglos interconectados de nodos mantenidos por la comunidad y para la comunidad, los interesados en formar parte de la red ofrecen su hardware y extienden la red, la cual crece de forma orgánica y descentralizada. A pesar de que estos despliegues sean de bajo costo, descentralizados, e incluso caóticos, los usuarios esperan una mínima calidad de servicio y velocidad de respuesta que les permita aprovechar tanto los servicios tradiciones de Internet así como servicios locales disponibles solo para los usuarios de la red comunitaria[2]. Tener un sistema de monitoreo de redes de bajo impacto sobre los recursos de red y de hardware disponibles y que ofrezca información sobre la red en tiempo real es crucial para ayudar a los interesados a detectar problemas y asegurar la calidad de servicio que los usuarios esperan.

Existen dos estrategias de monitoreo de redes: monitoreo activo o *benchmarking* que consiste en generar tráfico para realizar medidas y comprobar la respuesta de la red, y monitoreo pasivo que consiste en escanear el tráfico de la red en ciertos puntos estratégicos para censar el tráfico en la red. Para llevar a cabo el *benchmarking*, ha de inyectarse tráfico debidamente para conservar el funcionamiento normal de la red. Ejemplos de herramientas de *benchmarking* son *ping*, *iperf* y *traceroute*.

El monitoreo pasivo nos puede dar una idea del uso de la red, sin embargo,

para mayor efectividad debe realizarse en nodos intermedios a los que muchas veces no tenemos acceso. Ejemplos de herramientas de monitoreo pasivo son *tcpdump*[1] y *wireshark*[2]. En ambos casos hay que resaltar que es difícil tener una imagen completa del estado de la red.

Uno de los trabajos más importantes en el área de monitoreo de redes es el Protocolo Simple de Administración de Red o SNMP (del inglés *Simple Network Management Protocol*). Este emergió como una de las primeras soluciones al problema de manejo de redes y se ha convertido en la solución más ampliamente aceptada debido a su diseño modular e independiente de productos o redes específicas. Así como plantea Kurose y Ross[3], SNMP consiste de (1) un administrador de red, (2) una serie de dispositivos remotos monitoreados, (3) bases de información de administración (MIBs) en estos dispositivos, (4) agentes remotos que reportan la información de las MIBs al administrador de red y toman acciones si les indica, y, (5) un protocolo de comunicación entre los dispositivos.

SNMP no solo ofrece al administrador de red reportes sobre cada uno de los dispositivos administrados sino que también permite tomar acción sobre ellos de forma proactiva antes de que ocurran problemas o de forma reactiva para solucionar problemas cuando ocurren de forma inesperada.

Las redes en zonas rurales como en el caso de la red de Malawi[4], deben dejarse desatendidas durante largos periodos de tiempo ya que sus nodos son de difícil acceso o es muy costoso tener personal dedicado que se encargue del mantenimiento. Este escenario hace evidente la necesidad de una solución de monitoreo de redes a distancia y de dispositivos de mínimo mantenimiento. Además es atractivo para el centro de monitoreo de la red poder añadir, modificar o eliminar nodos de interés de manera sencilla a la interfaz de monitoreo.

Para intentar solucionar este problema y recolectar información sobre este tipo de redes, en [5] se propuso un sistema en dos partes: un monitor de red instalado en la estación base (BS) y una aplicación web remota, el monitor en la estación base se conecta a cada uno de los nodos de la red y determina el tiempo de ida y vuelta (RTT

[1] tcpdump: http://www.tcpdump.org/
[2] Wireshark: https://www.wireshark.org/

por sus siglas en ingles) de forma automatizada en ciertos intervalos de tiempo, guarda los resultados en archivos y los coloca en una carpeta que se sincroniza a través de un servicio en la nube de tipo PaaS (Plataforma como servicio) con el servidor web, que a su vez escanea la carpeta compartida y actualiza su base de datos que puede usarse para generar gráficas de RTT promedio y determinar tiempos de actividad continuos y porcentaje de disponibilidad de servicio.

Se han realizado otros trabajos en el área de monitoreo de redes como Bowlmap[6]; este es un sistema de monitoreo de redes a través de la visualización de mediciones para el Laboratorio Abierto Inalámbrico de Berlín (BOWL, por sus siglas en ingles). Este sistema permite hacer ajustes en sus pruebas existentes, así como agregar pruebas nuevas y a su vez generar las visualizaciones necesarias para el análisis de dicha información.

Bowlmap permite la observación del estado de la red en tiempo real y minimiza el uso de recursos de red transmitiendo solo la información necesaria para cada actualización; es decir, que solo envía el estado completo de la red cuando un cliente comienza a observar y las siguientes actualizaciones solo contienen la descripción de los cambios que han ocurrido desde entonces.[6].

Netradar[7] es un sistema orientado a la evaluación de la calidad de redes de telefonía y e Internet inalámbrico. Este obtiene datos como la intensidad de la señal, velocidad de descarga, posición geográfica, y latencia a partir de pruebas realizadas en dispositivos móviles. Estos datos son usados para generar mapas de calor indicando la calidad del servicio por regiones y por proveedor de servicio.

Project Bismark[8] es un proyecto de monitoreo de redes con el objetivo de investigar el rendimiento de redes en hogares, el software de Bismark se instala en enrutadores y realiza mediciones regulares (benchmarks) y capturas del tráfico de la red. Los usuarios de Bismark pueden observar los datos sobre la latencia, ancho de banda y uso de la red a través de un panel de control disponible en linea.

Ripe Atlas[9] es otro proyecto de monitoreo de redes con el objetivo de obtener un mayor entendimiento del estado de Internet en tiempo real. Ripe Atlas recolecta datos a partir de mediciones realizadas en dispositivos especializados llamados *probes* y *anchors*. Los *probes* realizan mediciones para determinar la conectividad y

alcanzabilidad con respecto a ciertos nodos de interés como servidores DNS o *anchors*. Los *anchors* son *probes* extendidos que además de ejecutar pruebas sirven como referencias regionales para la realización de mediciones. Los usuarios que colaboran alojando cualquiera de estos dispositivos pueden aprovechar toda la red de medición para realizar sus propias pruebas y observar todos los resultados obtenidos.

Tanto Netradar como Project Bismark y Ripe Atlas dependen de la colaboración de numerosos usuarios para la recolección masiva de datos. A mayor número de colaboradores es posible hacer análisis mas completos y detallados sobre el estado y calidad del servicio de Internet a nivel mundial. A cambio de su colaboración, los usuarios obtienen información valiosa sobre su propia red y pueden consultar los resultados públicos y comparar su rendimiento con respecto al total de datos disponibles.

1.1 Planteamiento del Problema

Las redes de computadoras se componen de un conjunto de nodos interconectados y generalmente no ofrecen garantías sobre el servicio que prestan. Algunas aplicaciones dependen de una alta disponibilidad y estabilidad de la red, por lo que es esencial para un administrador de red tener información del estado de la red para diagnosticar y solucionar problemas asegurando la calidad de servicio.

Otro caso de uso interesante corresponde al monitoreo de datos del estado de redes de bajo costo en las que fácilmente pueden ocurrir largas interrupciones de servicio. Así mismo, para un cliente de un servicio de alojamiento web desea saber si su sitio web está disponible y que tan rápido responde. Plataformas como Pingdom [10] o UptimeRobot [11] permiten monitorear distintos servicios en Internet y generan alertas cuando encuentran problemas, sin embargo no son gratuitas y no permiten la inclusión de nuevos tipos de pruebas o la visualización masiva de datos históricos.

Mantener estas mediciones con las herramientas existentes se vuelve una tarea compleja mientras crece el número de nodos a monitorear pues la cantidad de datos aumenta a través del tiempo. Sumado a esto, solo podemos capturar información a partir de los nodos extremos de la red.

En este trabajo, proponemos la construcción de un sistema de monitoreo de redes a bajo costo, configurable, y tolerante a fallas en las redes a monitorear. El sistema se manejará a través de una aplicación web que permitirá a usuarios autenticados manejar agentes de red remotos, para ejecutar pruebas planificadas y recoger datos de los enlaces de interés. La transferencia de datos entre la aplicación web y los monitores remotos se realizará a través de la nube. A este sistema le hemos dado el nombre de Octopus Monitor[2].

1.2 Justificación

Ante la aparición de nuevas formas y organizaciones alternativas en la distribución del acceso a Internet, tales como redes comunitarias caracterizadas por despliegues caóticos y descentralizados o redes inalámbricas con enlaces de larga distancia que pueden ser poco confiables, se evidencia la necesidad de tener herramientas de bajo costo, sencillas de desplegar y mantener que puedan ayudar a los miembros de la comunidad a tener conocimiento del estado de la red.

A pesar de que existe una gran variedad de herramientas para el monitoreo de despliegues de redes arbitrarias, se desea construir una plataforma que facilite el monitoreo a través de una interfaz clara y metáforas intuitivas que abstraigan las entidades monitoreadas y que permita extender rápidamente el sistema cuando se deseen monitorear nuevas características de la red.

1.3 Objetivos

1.3.1 Objetivos Generales

Construir un sistema de monitoreo de redes de bajo costo con almacenamiento de datos en la nube de tipo PaaS que sea de fácil instalación y permita configurar múltiples monitores remotos. Este debe ajustarse a cambios en el uso de recursos en los nodos donde vive el sistema y la carencia de personal in sitio.

1.3.2 Objetivos Específicos

- Desarrollar un servicio de monitoreo de bajo costo que de servicio a múltiples monitores de red remotos, presente visualizaciones gráficas a partir de los datos recogidos y ofrezca un marco de trabajo para agregar nuestros tipos de pruebas a los monitores de red existentes.

- Desarrollar un cliente monitor para desplegar en nodos desatendidos con dispositivos recolectores de muestra de bajo costo (ej. Raspberry PI, Alix boards, APU) para observar el comportamiento de los enlaces a través de aplicaciones de monitoreo sencillas y de consola.

- Utilizar sistemas de bajo costo y alta disponibilidad en la nube para almacenamiento y transferencia de datos.

- Desarrollar un modulo de calculo asíncrono de gráficas que permita mejorar los tiempos de interacción del usuario final con el sistema utilizando técnicas para agilizar cómputo como *caching*, *prefetching*, *threads*, etc.

1.4 Metodología

Para el desarrollo de este trabajo se siguió una metodología en espiral; el modelo en espiral es un modelo del ciclo de vida del software donde el esfuerzo del desarrollo es iterativo. Cada ciclo de la espiral representa una fase del desarrollo de software, cada uno de los ciclos consiste de los siguientes pasos:

1. Determinar o fijar los objetivos. En este paso se definen los objetivos específicos para posteriormente identificar las limitaciones del proceso y del sistema de software, además se diseña una planificación detallada de gestión y se identifican los riesgos.

2. Análisis del riesgo. En este paso se efectúa un análisis detallado para cada uno de los riesgos identificados del proyecto, se definen los pasos a seguir para reducir los riesgos y luego del análisis de estos riesgos se planean estrategias alternativas.

3. Desarrollar, verificar y validar. En este tercer paso, después del análisis de riesgo, se eligen un paradigma para el desarrollo del sistema de software.

4. Planificar. En este último paso es donde el proyecto se revisa y se toma la decisión si se debe continuar con un ciclo posterior al de la espiral. Si se decide continuar, se desarrollan los planes para la siguiente fase del proyecto.

Se realizaron cuatro ciclos, el primero correspondió a la realización de un monitor remoto básico que realice mediciones de RTT de la red con almacenamiento en la nube y visualizaciones de los datos obtenidos.

El segundo ciclo consistió en permitir el monitoreo de una cantidad arbitraria de monitores remotos permitiendo a múltiples usuarios manejar sus monitores remotos desde el servicio web y obtener las visualizaciones.

El tercer ciclo correspondió en diseñar e integrar una prueba con otras herramientas (traceroute, iperf, etc) para conseguir puntos comunes y generar un enfoque de integración sencillo de los *wrappers* futuros a las aplicaciones. (ej. Lidiar con aplicaciones que requieren enfoque cliente como ping o cliente-servidor como iperf).

El cuarto ciclo consistió en hacer análisis del rendimiento del sistema y hacer las optimizaciones necesarias para ofrecer una calidad de servicio apropiada, determinar costos, limitaciones y requisitos mínimos para implementar en plataformas de bajo costo.

1.5 Alcance

El alcance de este trabajo remite al diseño y detalles de implementación tanto de una aplicación web que funge como plataforma central de coordinación del monitoreo, y un agente monitor de redes ligero y robusto que pueda ser controlado remotamente. Se diseñó un sistema flexible y un esquema de coordinación entre sus distintos entes a través de la nube.

1.6 Estructura del Documento

El presente trabajo se estructura de la siguiente manera:

Capítulo 1. Introducción, este capítulo consiste de los antecedentes relevantes al tema a tratar, planteamiento del problema, justificación, objetivos, metodología a emplear y el alcance de este trabajo.

Capítulo 2. Marco Teórico, define brevemente los conceptos y herramientas usados durante el desarrollo de este trabajo y que son esenciales para su comprensión; tales como conceptos de alto nivel de monitoreo de redes, métricas de la calidad de un enlace, visualización de datos y los y sistemas de software usados para su implementación.

Capítulo 3. Aplicación Web, contiene el diseño general del sistema de monitoreo, así como la arquitectura de la aplicación web, sus sub-sistemas, componentes relevantes, casos de uso y finalmente análisis de rendimiento.

Capitulo 4. Monitor de Red, explica el diseño del monitor de red remoto, su arquitectura, componentes y los módulos de prueba implementados.

Capitulo 5. Framework de Integración de Pruebas, describe los requisitos del marco de trabajo para la implementación de nuevas pruebas, así como todos sus componentes genéricos, finalmente explica como usar dichos componentes para integrar nuevas características a un sistema en producción.

Capitulo 6. Conclusiones y Recomendaciones, contiene las conclusiones obtenidas a partir del diseño e implementación del sistema, así como recomendaciones y posibles mejoras a incluir en trabajos futuros.

Finalmente se presentan los apéndices y las referencias bibliográficas.

Capítulo 2

Marco Teórico

En este capitulo se presentan los conceptos necesarios para la comprensión de este trabajo y se describen las herramientas de software, métodos y formatos que se emplearán para el desarrollo del sistema propuesto.

2.1 Monitoreo de Redes

El monitoreo de redes se refiere al uso de sistemas computacionales para determinar el estado de redes de computadoras. El estado de la red puede ser visto como el conjunto de factores o métricas que la describen en un momento dado. Mientras una red tiene ciertos elementos relativamente invariables como la posición de sus nodos, la longitud de los enlaces, el tipo de enlace (fibra óptica, cable, satelital, etc), el ancho de banda de los enlaces, la velocidad de procesamiento de los enrutadores, entre otros, el estado de la red viene determinado por elementos generalmente impredecibles, como condiciones climatológicas, problemas de configuración, equipos en mal estado y congestión. Determinar todo este tipo de problemas para generar alertas, aplicar acciones correctivas o al menos tener conocimiento de ellas, es el objetivo de un sistema de monitoreo de redes.

Existen dos paradigmas fundamentales en el monitoreo de redes, el monitoreo activo que consiste en generar tráfico y observar la respuesta de la red y el monitoreo pasivo, que consiste en observar el tráfico que atraviesa un cierto enlace o nodo. Para tener

una imagen completa de la red es conveniente usar ambos paradigmas ya que ambos pueden ofrecer distintas perspectivas.

2.2 Redes Comunitarias

Las redes comunitarias se caracterizan por ser despliegues de redes a gran escalada construidos y organizados de una manera descentralizada [1], el crecimiento de la red es orgánico y está abierto a la participación de cualquiera. Los usuarios interesados en formar parte de la red colaboran aportando su propia infraestructura de red, agregando nuevos nodos que permiten que otros usuarios a su vez puedan continuar extendiendo la red.

A pesar de que el despliegue es distribuido y auto-gestionado se necesita una mínima infraestructura de gobernanza para coordinar el direccionamiento IP y enrutamiento, las redes comunitarias tienden a estar compuestas por elementos de hardware y software muy diversos, y usualmente coexisten en ellas enlaces alambrados e inalámbricos, distintos protocolos de enrutamiento o sistemas de manejo de la topología de red[1].

Las redes comunitarias sirven como una forma de distribuir el acceso a Internet de forma libre y neutral, tanto en zonas urbanas como rurales, sin embargo, también pueden servir para distribuir servicios y aplicaciones comunitarias tanto completamente gratuitas como comerciales. Cada participante sigue siendo dueño de la infraestructura que presta a la red y se benefician mutuamente al tener acceso a los servicios y aplicaciones de la red comunitaria.

2.3 Métricas de calidad de un enlace

En esta sección, se explican las métricas mas comúnmente utilizadas para determinar la calidad de un enlace. En este trabajo hemos implementado perfiles de monitoreo utilizando una o mas de ellas.

2.3.1 Latencia

La latencia es el tiempo que le toma a un paquete o mensaje viajar desde su origen hasta el punto destino. Teóricamente la latencia está relacionada directamente con la distancia entre los puntos finales de una comunicación, en la practica los paquetes viajan a través de una red de enrutadores que retransmiten el mensaje hasta su destino. Según Grigorik[12] la latencia total será la suma de cada uno de los siguientes factores para cada uno de los enrutadores que atraviese el paquete:

- **Retraso de Propagación:** es el tiempo que tarda un mensaje en viajar del emisor al receptor y es función de la distancia por la velocidad a la que la señal se propaga.
- **Retraso de Transmisión:** es el tiempo que tarda el emisor en poner todos los bits de un mensaje en el medio de transmisión y es función de la longitud del paquete y el ancho de banda del enlace.
- **Retraso de Procesamiento:** tiempo requerido para revisar la cabecera del paquete, buscar errores y determinar el destino del paquete.
- **Retraso de Colas:** Cantidad de tiempo que un paquete pasa en la cola de una interfaz esperando su turno por ser procesado.

2.3.1.1 Bufferbloat

Bufferbloat es la existencia de *buffers* excesivamente grandes y generalmente llenos presentes en Internet; puede parecer contra-intuitivo ya que *buffers* mas grandes implican que menos paquetes serán desechados al llegar a un enrutador congestionado, pero mientras la cola en la interfaz del enrutador crece también lo hace el tiempo de espera del paquete y a la vez interfiere (o invalida) los algoritmos de control de congestión de los protocolos mas comunes en la capa de transporte [13].

El *bufferbloat* puede ser mitigado configurando apropiadamente el hardware disponible, sin embargo, es difícil de diagnosticar y es confundido frecuentemente con congestión en la red.

Recientemente se ha comenzado a medir el *blufferbloat* como el tiempo adicional que toma enviar paquetes a través de un enlace congestionado, algunas pruebas disponibles en linea [14][15] intentan determinar este retraso haciendo mediciones constantes de latencia al mismo tiempo que inundan el enlace para determinar la forma en que esta varía durante la prueba.

2.3.2 Tiempo de ida y vuelta

El tiempo de ida y vuelta (RTT por sus siglas es ingles) es el tiempo que le toma a un paquete viajar desde el origen a su destino y de vuelta, podría pensarse que el RTT es aproximadamente el resultado de multiplicar la latencia por dos, sin embargo existen factores como el retraso por procesamiento en el equipo destino o diferencias en el enrutamiento del paquete en su camino de vuelta.

Uno de los métodos mas populares para medir el RTT es enviar un paquete *ICMP Echo Request* y esperar el correspondiente *ICMP Echo Reply*, sin embargo, también es posible medir el RTT pasivamente durante la transmisión de un flujo TCP usando marcas de tiempo en la cabecera de los mensajes TCP[16].

Es importante notar que no es lo mismo medir el RTT desde la capa de red con el protocolo ICMP que hacerlo en la capa de transporte con TCP o en la capa de aplicación con un protocolo como HTTP, evidentemente el RTT será mayor en las capas superiores, sin embargo estas métricas también son útiles ya que se acercan mas fielmente a la experiencia del usuario.

2.3.3 Perdida de paquetes

La pérdida de paquetes ocurre cuando los paquetes en una transmisión fallan en llegar a su destino, ya sea por interferencias en el medio de transmisión (por ejemplo obstáculos físicos en un enlace Wi-Fi), o cuando son desechados por un nodo de la red. Los paquetes pueden ser desechados si se determina que están corrompidos o mas comúnmente debido a escenarios de congestión en los que un enrutador está recibiendo paquetes a una tasa mayor de la que puede retransmitirlos y no tiene mas opción que desechar los paquetes que desborden la cola.

La perdida de paquetes afecta dramáticamente la latencia percibida en la capa de aplicación, según [16] una perdida de paquetes del 5% puede introducir un retraso de medio segundo en la aplicación.

Mientras puede parecer que la perdida de paquetes es siempre producto de un problema, esta juega un papel vital en el algoritmo de prevención de congestión (*congestion avoidance*) del protocolo TCP, ayudándolo a detectar congestión en la red el cual responderá limitando su tasa de envío de datos, esto ha sido esencial para evitar el colapso de las redes IP.

2.3.4 Jitter

Se llama *jitter* a la fluctuación de la latencia durante la transmisión de un conjunto de paquetes, estas fluctuaciones vienen dadas principalmente por la variación del retraso que los paquetes experimentan en las colas en los enrutadores[3], sin embargo todos los factores mencionados en la sección 2.3.1 pueden contribuir en menor medida.

El término *jitter* puede tener distintas connotaciones sin embargo es usado frecuentemente por científicos en el área de la computación como la variación de una métrica (comúnmente latencia) con respecto a otra métrica de referencia (como latencia mínima o promedio), a esto también se le llama variación en el retraso de los paquetes (PDV por sus siglas en ingles), este término es a veces preferido por ser mas preciso [17].

El *jitter* puede afectar considerablemente transmisiones en tiempo real como VoIP o *streaming*, sin embargo la correcta implementación de políticas de *buffering* del lado del receptor puede minimizar e incluso mitigar este efecto[3].

2.3.5 Throughput

En términos de redes de computadoras, el *throughput* es la tasa en bits por segundo a la que fluyen los datos a través de un enlace [3], el *throughput* puede compararse al caudal de un rio, donde mientras mas ancho sea el rio mas agua puede fluir a través de él, así mismo mientras mayor sea el ancho de banda de un enlace mayor será throughput.

El máximo *throughput* teórico entre un par de nodos esta determinado

principalmente por el segmento de la red con menor ancho de banda, este comportamiento se ilustra en la figura 2.1, donde el cable entre el punto de acceso Wi-Fi y el proveedor de servicio de Internet resulta ser el cuello de botella en la comunicación. En la práctica el *throughput* también viene determinado por varios factores como otros flujos de datos con los que se comparte la red o la velocidad a la que el receptor es capaz de procesar el flujo de datos entrante.

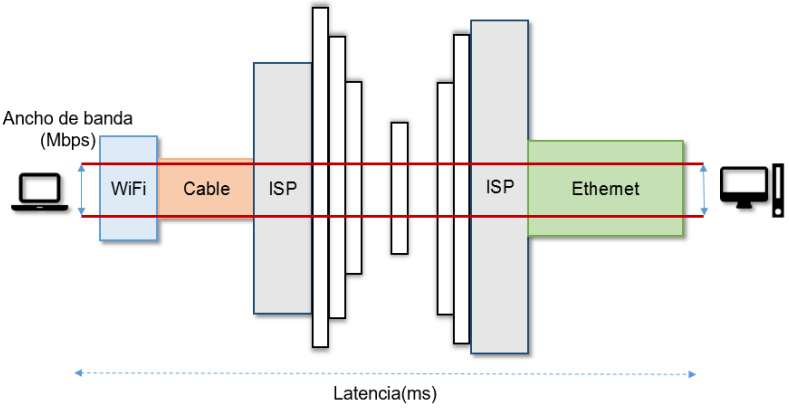

Figura 2.1: Latencia y ancho de banda

Para aplicaciones en tiempo real es esencial tener un *throughput* mínimo mayor a un cierto umbral que asegure que los datos se estén recibiendo a una tasa mayor o igual a la que se están consumiendo[3], es decir que a partir de esa tasa mínima un mayor *throughput* no implica una mejoría en la calidad de la comunicación, otras aplicaciones como transferencias de archivos se benefician del mayor *throughput* posible ya que el tiempo de transferencia es una función del tamaño del archivo entre el *throughput* durante la transferencia.

No es posible medir el *throughput* de un enlace de forma pasiva, por lo que las pruebas que existen consisten en inyectar tráfico al enlace hasta saturarlo y determinar la velocidad a la que se reciben los datos del lado del receptor. Existen múltiples maneras de saturar el enlace, por ejemplo Speedtest [18] utiliza hasta cuatro hilos paralelos transmitiendo mensajes aleatorios a través de HTTP, utilizar múltiples flujos es una forma efectiva de mitigar el efecto de la latencia sobre el *throughput*.

A diferencia de las métricas anteriores medir el *throughput* solo es posible con la colaboración de los nodos extremos del enlace, por lo que es necesario tener algún tipo

de software preparado para aceptar la prueba e informar del resultado obtenido.

2.3.6 Disponibilidad

Un servicio esta disponible para un cliente cuando dicho cliente puede comunicarse con él, análogamente un servicio no esta disponible para un cliente cuando no puede comunicarse con él, ya sea por un problema en el nodo final o en la red [19].

Generalmente la disponibilidad se mide a través de la disponibilidad promedio, que es la fracción del tiempo durante el cual un servicio esta disponible para un cliente promedio, sin embargo también se puede usar la disponibilidad continua. En este trabajo llamaremos actividad continua a un evento durante el cual un servicio está continuamente disponible para un cliente, e inactividad continua a un evento durante el cual un servicio esta continuamente no disponible para un cliente.

Hemos definido la disponibilidad como un concepto meramente binario, en el que si un servicio es alcanzable entonces está disponible. Sin embargo, hay ciertos escenarios en los que se puede considerar que un nodo alcanzable está fallando en ofrecer un servicio adecuadamente. Por ejemplo, cuando un servidor web está respondiendo a las peticiones con un código de estado 500[1], o cuando la latencia es tal que hace que una comunicación en tiempo real no sea posible.

2.3.7 Ping

Ping es un programa utilitario incluido en todos los sistemas basados en UNIX y Windows que comprueba la presencia y tiempo de respuesta de un *host* en una red IP. Ping utiliza el Protocolo de Mensajes de Control de Internet (ICMP) para enviar un paquete de solicitud ICMP (*ICMP Echo Request*) y espera el mensaje de respuesta del *host* remoto (*ICMP Echo Reply*); calculando la diferencia de tiempo entre el envío y la recepción se puede calcular el RTT, Ping también incluye funciones para enviar paquetes en ráfaga útil cuando se desea medir la perdida de paquetes.

[1]Error HTTP 500 *Internal server error* (Error interno del servidor) El servidor encontró una condición inesperada que evitó que completara la petición.

2.3.8 Traceroute

Traceroute (también llamado Tracert en sistemas Windows) es un programa utilitario de diagnostico que permite conocer los *hosts* que visita un paquete durante su transito por una red.

Al igual que Ping, Traceroute utiliza el protocolo ICMP pero envía paquetes con un valor de *Time to Live* (TTL) incremental; cada vez que un nodo de la red recibe un paquete decrementa su valor de TTL y si éste llega a cero lo descarta y envía de vuelta al *host* emisor un mensaje de control indicando que el TTL llegó a 0. De esta manera Traceroute puede generar una lista de los nodos visitados y el valor de RTT para cada uno de ellos.

Un análisis cuidadoso de la salida de Traceroute puede ayudar a diagnosticar numerosos problemas en una red como ineficiencia en el enrutamiento, presencia de enrutadores congestionados, cuellos de botella, comportamientos inesperados, etc.

2.3.9 Iperf

Iperf es una herramienta que permite medir el rendimiento (*throughput*) entre un par de nodos en un red. Al igual que muchas otras pruebas para medir velocidad de transferencia, Iperf funciona con una arquitectura cliente-servidor, donde el cliente genera un flujo de datos hacia el servidor y mide la velocidad obtenida. También es posible ejecutar una prueba "en reversa" donde el servidor es el que genera el flujo de datos.

Iperf puede ejecutar pruebas utilizando los protocolos TCP o UDP para la transferencia de los datos, sin embargo existen diferencias entre ellos:

- Ya que UDP a diferencia de TCP no implementa ningún algoritmo de control de congestión se podría obtener un rendimiento ligeramente superior con este protocolo.
- Con UDP se puede obtener una estadística de la perdida de datagramas durante la prueba.

- Con UDP es el servidor el que totaliza los resultados, ya que el cliente no tiene manera de saber que datagramas se han recibido.

Para obtener una buena medición del rendimiento del enlace se deben ajustar los parámetros de la prueba cuidadosamente. Es posible ajustar la cantidad de datos que se van a transferir durante la prueba, elegir una cantidad muy pequeña podría resultar en que no sea suficiente para saturar el enlace, mientras tanto elegir una cantidad demasiado grande podría resultar en una prueba innecesariamente larga, en ambos casos el valor del rendimiento obtenido no reflejará la realidad. También hay que tomar en cuenta que es difícil obtener una lectura exacta del rendimiento del enlace ya que podrían existir otros flujos en la red que afecten el resultado de la prueba.

2.4 Visualización de datos

La visualización de datos se refiere al aprovechamiento de elementos gráficos para representar información cuantitativa; cualquier conjunto de datos no tiene significado sin alguna manera de organizar y presentar los descubrimientos relevantes que se encuentran potencialmente ocultos dentro de estos.

Los humanos podemos comprender los datos de mejor manera cuando son presentados a través de imágenes y elementos gráficos que leyendo números en tablas y listas[20]. Una visualización apropiada de los datos permite de forma efectiva preguntar y responder las preguntas relevantes a una organización, por ejemplo, en el marco de un sistema de monitoreo de redes preguntas como "¿Dónde están apareciendo los cuellos de botella?" "¿Qué factores afectan el rendimiento de un enlace?" o "¿Cuales son los patrones de uso de los usuarios de la red?".

Hay una variedad de métodos apropiados para visualizar distintos conjuntos de datos, por ejemplo, los datos discretos se pueden observar a través de gráficas de barras, los gráficos de redes pueden comunicar la relación entre distintos entes, los mapas son efectivos para desplegar información geográfica, los mapas de calor permiten comparar el rendimiento de una variable a través del tiempo, etc. Cada una de estas visualizaciones puede ser enriquecida a través del uso creativo de colores y formas para agregar nuevas dimensiones a los datos representados. Es posible combinar

distintos conceptos para lograr visualizaciones aún más poderosas como mapas de calor superpuestos en mapas que pueden expresar datos de la densidad de una variable al mismo tiempo que se da una idea de su posición geográfica.

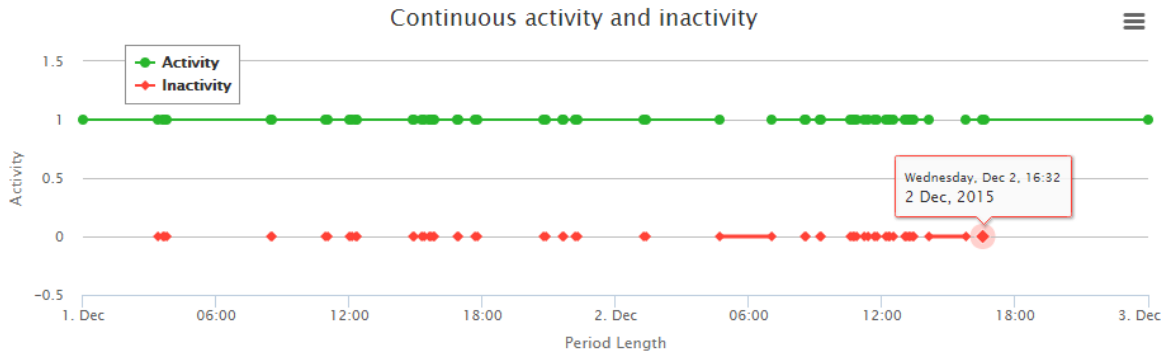

Figura 2.2: Visualización de periodos de actividad e inactividad continua

En la figura 2.2 se observa una visualización de los periodos de actividad continua como segmentos de linea verde y los periodos de inactividad como segmentos de linea roja, es posible hacer *zoom* para observar los periodos a mayor detalle y revisar el tiempo exacto del inicio del período pasando el puntero sobre un punto, la concentración de puntos de distintos colores da una idea rápida de la estabilidad del enlace.

2.5 Computación en la nube

La computación en la nube se refiere tanto a las aplicaciones desplegadas como servicios en Internet y el *hardware* y los sistemas computaciones en los centros de datos que proveen dichos servicios [21], a los servicios en si mismos se les ha dado el nombre de *Software* como Servicio (SaaS) y al *hardware* y *software* en los centros de datos es a lo que llamamos una nube. Cuando una nube se hace disponible para el público en general a través de alguna forma de pago, se le llama una nube pública y el servicio que se esta vendiendo es Computación como Utilidad (*Utility Computing*), se le llama nube privada a los centros de datos internos de negocios u otras organizaciones pero que no pueden ser usados por el publico general, por lo tanto llamamos computación en la nube a la suma de SaaS y Computación como Utilidad sin incluir nubes privadas.

La computación en la nube se caracteriza por ofrecer métodos de pago flexibles que permiten pagar solo por los recursos que se están utilizando (*pay-as-you-go*) y con una fina granularidad de modo que es lo mismo pagar mil procesadores una hora que un procesador por mil horas, esto permite a aplicaciones manejar cargas y escalas fluctuantes o patrones de uso específicos sin necesidad de tener *hardware* que esté ocioso durante largos periodos de tiempo. La nube ofrece a desarrolladores la ilusión de recursos computaciones ilimitados y disponibles a petición, eliminando la necesidad de planificar y aprovisionar equipos de hardware, y minimizando los riesgos relacionados con subestimar una aplicación que explota en popularidad o sobrestimar una aplicación que no llena las expectativas. En otras palabras no es necesario tener un gran capital de inversión inicial independientemente de la escala que resulte necesario manejar a corto o mediano plazo.

2.5.1 Almacenamiento como Servicio (STaaS)

Almacenamiento como servicio permite tanto a aplicaciones como a usuarios almacenar sus datos en discos remotos y acceder a ellos en cualquier momento y lugar[22]. El Almacenamiento como servicio permite a aplicaciones escalar mas allá de las limitaciones de su infraestructura, y abarata los costos de desarrollo y mantenimiento de la infraestructura necesaria.

Los sistemas de almacenamiento de datos en la nube deben cumplir ciertos requisitos de disponibilidad, rendimiento, seguridad, replicación, y consistencia de los datos que hacen de estos sistemas una opción atractiva el mantenimiento de los datos de una aplicación a largo plazo.

2.5.2 Plataforma como Servicio (PaaS)

Plataforma como servicio ofrece como servicio una plataforma para el desarrollo, ejecución y manejo de aplicaciones web, los recursos computacionales demandados por la aplicación son manejados automáticamente, de modo que la complejidad de manejar la infraestructura subyacente queda eliminada, esto permite reducir enormemente la complejidad necesaria para desplegar aplicaciones, que pueden pasar de la etapa de

desarrollo y pruebas rápidamente a un entorno de producción con un esfuerzo mínimo.

2.6 Herramientas usadas para el desarrollo del sistema

A continuación se describen las herramientas usadas para el desarrollo del sistema de monitoreo de redes orientado a la recolección masiva de datos.

2.6.1 Modelo Vista Controlador (MVC)

El Modelo Vista Controlador es un patrón de diseño que divide la lógica de los datos y la presentación de forma claramente identificable y bien definida [23]. Este patrón de diseño es muy popular en el marco de aplicaciones web ya que su abstracción permite escribir software altamente desacoplado y fácil de mantener y escalar.

2.6.1.1 Modelo

Según Bascón[23]: *"El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar, así por ejemplo un sistema de administración de datos climatologías tendrá un modelo que representará la temperatura, humedad ambiental, estado del tiempo esperado, etc..."*

Según la implementación de MVC el modelo puede dividirse en el modelo del dominio que es el modelo propiamente dicho, es decir, una colección de clases que modelan la realidad relevante a la aplicación, y opcionalmente el modelo de la aplicación; este modelo tiene conocimiento de las vistas y es capaz de enviar notificaciones cuando ocurren cambios en el modelo. El modelo de la aplicación también es llamado coordinador de la aplicación [23] .

2.6.1.2 Vista

La vista es la encargada de determinar que información contenida en el modelo mostrar al usuario y la presentación, por ejemplo si se está modelando una red es posible tener

una vista que dibuje gráficamente el nivel de congestión de cada enlace y otra vista que sencillamente muestre estas propiedades en una tabla.

La vista pudiera cambiar cuando se actualiza el modelo del dominio a partir de notificaciones emitidas por el modelo de la aplicación. Siguiendo con el ejemplo anterior de esta forma sería posible monitorear en tiempo real el estado de la red a partir de agentes que detecten las condiciones de cada enlace y mantengan actualizado el modelo.

2.6.1.3 Controlador

El controlador es el encargado de dirigir el flujo de control de la aplicación a partir de mensajes externos, como datos introducidos por el usuario en el caso de una aplicación de escritorio o peticiones HTTP en el caso de aplicaciones web. A partir de estos estímulos, el controlador se encargará de invocar las vistas apropiadas, actualizar el modelo y hacer todas las acciones necesarias.

Distintas implementaciones del patrón MVC se toman la libertad de establecer la linea que separa el controlador y la vista de forma distinta, en algunas implementaciones, la vista es la encargada tanto de seleccionar los datos que van a mostrarse así como de desplegar la presentación, esta ultima es una variación de MVC a veces llamado MTV (Modelo-Plantilla-Vista).

2.6.2 Django

Django es un framework de desarrollo de aplicaciones web construido en Python, en este trabajo usamos Django como el fundamento para Octopus Head. Django permite construir aplicaciones web rápidamente gracias a su filosofía de "baterias incluidas", es decir, que incluye una inmensa gama de características comunes a la mayoría de las aplicaciones web como validaciones de formularios, autenticación de usuarios, manejo de sesiones entre muchos otros [24]. Así el desarrollador puede concentrarse en escribir la lógica que es especifica a su aplicación y dejar que Django maneje los aspectos repetitivos y muchas veces tediosos de la pila de desarrollo web.

Django esta diseñado con una arquitectura MVC es decir que separa claramente la lógica, de los datos y la forma en que dichos datos son presentados al usuario, en el

caso de Django la "vista" describe que datos son presentados al usuario y el "template" representa la forma en que dichos datos son presentados.

Cuando Django recibe una petición ésta pasa por un despachador de URLs (*URL Dispatcher*), cuya tarea es emparejar el URL con una vista y delegar a la vista el manejo de la petición. La vista contiene la lógica necesaria para atender la petición entrante. Generalmente esto consiste en retirar o actualizar algunos datos del modelo, que a su vez se comunica con el manejador de base de datos, finalmente la vista combina los datos retirados de la base de datos, la petición y la sesión activa con una plantilla (*template*) para generar la respuesta que será devuelta.

Las respuestas generadas por Django pueden ser tanto paginas HTML con CSS y JavaScript pensadas para la interacción con el usuario así como respuestas en formato JSON o XML para la construcciones de APIs que pueden ser usados para la comunicación maquina-maquina, esto es especialmente útil cuando se desea desarrollar aplicaciones en otras plataformas como Android[2] o iOS[3] que compartan el mismo *backend*.

Django ha demostrado ser escalable y flexible, se sabe de instancias de Django atendiendo ráfagas de cincuenta mil peticiones por segundo, además es de código abierto, gratuito y cuenta con una extensa comunidad de colaboradores y amplia documentación.

2.6.3 Celery

Celery es un sistema de procesamiento de tareas asíncrono, que permite tanto el encolamiento de tareas en tiempo real así como la planificación de tareas para ser ejecutadas mas tarde. Celery en realidad no implementa la mayoría de sus componentes, sino que define un protocolo de comunicación entre una serie de componentes (también llamados micro-servicios)[25]:

- Bróker de mensajería.
- Planificador.

[2]Android: https://www.android.com/
[3]iOS: http://www.apple.com/es/ios/

- *Workers* (Trabajadores).
- *Backend* de Resultados.

Celery es usado comúnmente junto a Django y permite a una aplicación web escalar a bajo costo ya que solo hace falta aumentar el número de trabajadores disponibles que se pueden distribuir en tantas máquinas como sea necesario.

2.6.4 Redis

Redis es un almacén de estructuras de datos en memoria que soporta una amplia gama de tipos como cadenas de caracteres, listas, conjuntos, conjuntos ordenados, mapas de bits e indices geo-espaciales [26].

Redis es comúnmente usado como base de datos, caché o bróker de mensajería; y se caracteriza por su velocidad de respuesta ya que todos sus datos se mantienen en memoria principal y solo invierte recursos en asegurar ciertos niveles de persistencia, sin embargo, por omisión, los datos almacenados se pierden en caso de que ocurra cualquier falla inesperada.

2.6.5 Bases de Datos

Uno de los pilares fundamentales de casi toda aplicación moderna es tener un modo de almacenar datos de forma persistente en el tiempo así como consultarlos y actualizarlos de forma rápida, segura y resistente a fallas.

Según [27] una base de datos es una colección de datos estructurados. Puede ser cualquier cosa desde una simple lista de compras, una galería de fotos o las bastas cantidades de información en una red corporativa. Para agregar, acceder y procesar la data almacenada en una base de datos, se necesita un sistema manejador de base de datos. Ya que los computadores hacen un muy buen trabajo manejando grandes cantidades de datos, los sistemas manejadores de bases de datos juegan un papel central en la computación como utilidades independientes o partes de otras aplicaciones.

SQL por sus siglas en ingles *Structured Query Language* (lenguaje de consulta estructurado) es el lenguaje estandarizado mas común para acceder a bases de datos,

SQL está definido por el Estándar ANSI/ISO SQL y ha ido evolucionando desde 1986 para convertirse en un estándar de facto en el mundo de la computación.

2.6.5.1 Mysql

Mysql es un sistema manejador de bases de datos relacionales (RDBMS por sus siglas en ingles) de código abierto bajo la licencia GPL (GNU General Public License). Mysql se caracteriza por ser rápido, confiable, escalable y fácil de usar, es posible instalar Mysql tanto en una maquina junto a otras aplicaciones como servidores web o también instarlo en maquinas dedicadas para que use todo el poder de cómputo disponible. Mysql posee características para ejecutarse en *clusters* de maquinas junto con un motor de replicación para obtener una alta escalabilidad [27].

2.6.5.2 Mapeo Objeto-Relacional

El Mapeo Objeto-Relacional (ORM por sus siglas en ingles) es un método para interactuar con bases de datos relaciones desde el paradigma de la programación orientada a objetos, de esta manera es posible aprovechar conceptos como herencia y polimorfismo.

Según Ambler[28], la mayoría de las aplicaciones modernas usan lenguajes orientados a objetos como Java o C# para construir aplicaciones y bases de datos estructuradas para almacenar datos, por lo tanto, es útil tener una interfaz que transforme los datos entre estos tipos incompatibles.

El uso de un ORM simplifica enormemente el manejo de la estructura de datos subyacente ya que permite al programador manejar los datos a un mayor nivel de abstracción como si fueran objetos, sin necesidad de generar manualmente las consultas SQL, además ésta capa de abstracción permite desacoplar el código de la aplicación de los detalles específicos de cada RDBMS.

2.6.6 JSON

JSON (*JavaScript Object Notation*) es un formato textual de intercambio y almacenamiento de datos no estructurados, JSON posee un formato que es fácil de

leer para humanos y fácil de interpretar para maquinas y mas ligero que XML por lo que se ha popularizado para el desarrollo de APIs.

JSON soporta dos tipos de estructuras de datos fundamentales:

1. Colecciones de pares <nombre,valor> comparable a un diccionario o tablas *hash*.
2. Listas ordenadas de valores, similar a las listas o vectores que existen, virtualmente en cualquier lenguaje de programación

Un archivo en formato json puede estar formado de cualquiera de las estructuras de datos antes descritas o cualquier permutación de dichas estructuras anidadas.

2.6.7 Dropbox

Dropbox es una plataforma de almacenamiento de datos en la nube de tipo STaaS que permite compartir y sincronizar archivos entre un número arbitrario de clientes.

Dropbox es usado por aproximadamente 400 millones de personas y 100.000 organizaciones[29] y posee aplicaciones en Windows, Linux, Mac OS X, iOS, Android, Blackberry y web.

Como todo servicio en la nube es atractivo para desarrolladores por ser robusto y confiable y es gratis hasta alcanzar una cierta cantidad de espacio de almacenamiento usado, a partir de ese punto incluye planes de pago que dependen de la cantidad de espacio usado.

2.6.7.1 API de Dropbox

Un API (*Aplication Programming Interface*) o interfaz de programación de aplicaciones, es una serie de métodos o funciones orientados a la comunicación maquina-maquina, en el caso de la computación en la nube un API conforma un servicio que permite desarrollar aplicaciones sobre una plataforma (en este caso Dropbox).

El API de Dropbox permite realizar peticiones (como subir o descargar archivos, listar directorios o crear carpetas) sobre el espacio de almacenamiento de un usuario, el usuario debe previamente dar permiso a la aplicación para que esta pueda hacer cambios a su nombre, el usuario puede elegir denegar el acceso a la aplicación en todo

momento y existen distintos tipos de esquemas de acceso donde una aplicación solo tiene acceso a un conjunto limitado de directorios dentro del espacio de almacenamiento del usuario.

El API de Dropbox impone ciertos límites de peticiones por usuario para impedir que una aplicación realice una cantidad excesiva de peticiones en un período corto de tiempo, sin embargo el límite se considera lo suficientemente alto como para no entorpecer la inmensa mayoría de los casos de uso.

2.6.8 Diseño web adaptable

Debido a la inmensa diversidad de dispositivos desplegados en el mercado y sus distintos tamaños y formas es imposible realizar manualmente diseños que puedan ajustarse a cada uno de ellos, anteriormente una solución popular a este problema era tener varias versiones con distintas resoluciones y elegir que versión mostrar a cada cliente, sin embargo esto ya no es necesario gracias a las nuevas herramientas disponibles en HTML5 CSS y JavaScript que están ampliamente implementadas en navegadores modernos.

El diseño web adaptable es la tendencia en el diseño de paginas web que se ajusten elásticamente a cualquier resolución, adaptando la forma en que se presentan sus elementos de forma "inteligente".

Las técnicas mas comunes para lograr esto es tener elementos que ocupen el mayor espacio horizontal posible en pantallas grandes y mientras el tamaño horizontal se reduce, estos pasan a ocupar el espacio verticalmente. Siempre se ocupa el máximo del ancho disponible, evitando crear barras de desplazamiento horizontal que desorientan e incomodan a los usuarios.

Otra heurística en la creación de sitios web adaptables es escalar o esconder elementos gráficos decorativos o menús de navegación laterales, reducir el tamaño de margenes o incluso cambiar tipos de letras para que sean mas legibles en dispositivos móviles, mientras el tamaño del dispositivo es menor, cada pixel se vuelve mas precioso.

El diseño web adaptable no solo facilita el desarrollo de sitios web ahorrando a los desarrolladores y diseñadores el costo de construir múltiples versiones de un mismo sitio web sino que además se ofrece una experiencia similar independientemente del

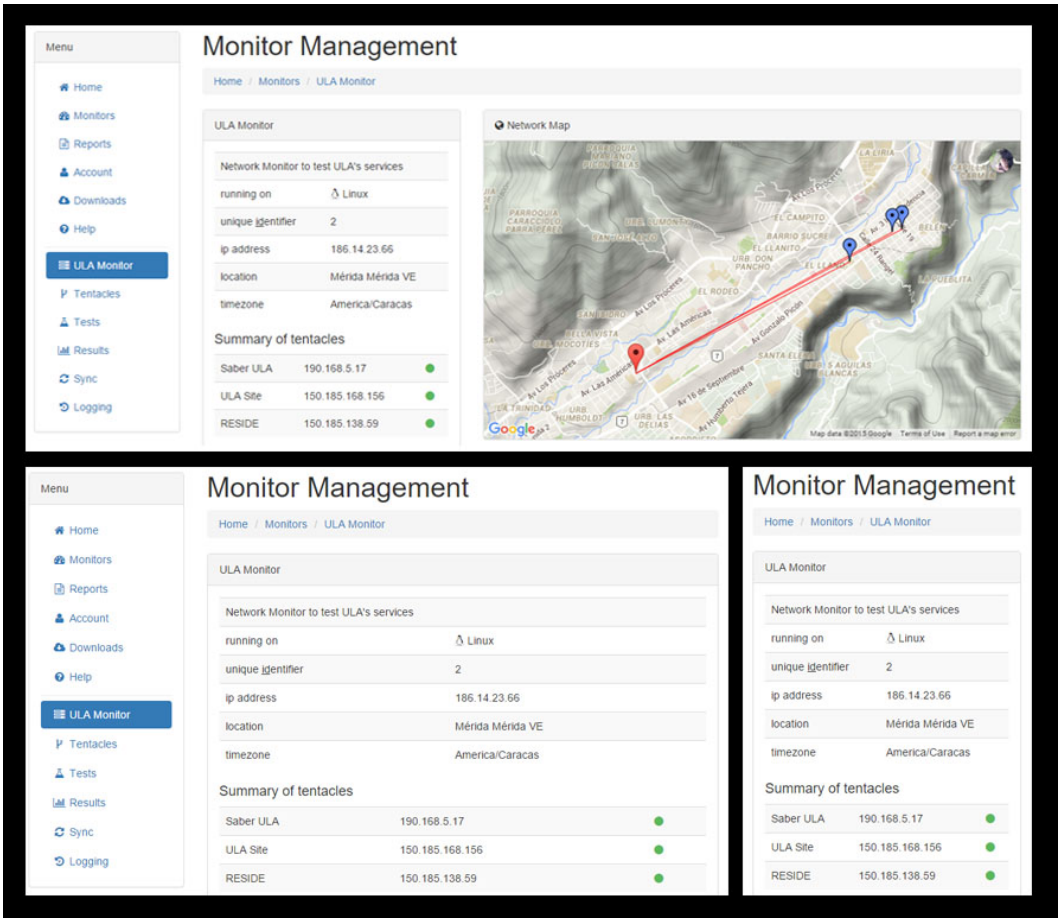

Figura 2.3: Ejemplo de diseño adaptable, de (1) a (3) se observa la adaptación vertical de la composición de la página.

dispositivo con el que se esté visitando.

2.6.9 Highcharts

Highcharts es una biblioteca JavaScript para dibujar gráficas en entornos HTML es gratis para proyectos no comerciales y de código abierto[30]. Las gráficas generadas por Highcharts aprovechan las características de HTML5 por lo que pueden soportar enormes conjuntos de datos sin afectar el rendimiento incluso en dispositivos móviles, son interactivas de manera que el usuario puede inspeccionar detalladamente el conjunto de datos y son dinámicas permitiendo actualizar la gráfica en tiempo real cuando se reciben nuevos datos.

Highcharts incluye una amplia gama de gráficas predefinidas que la hace ideal para todo tipo de visualización de datos como mapas de calor, *splines*, gráficas de área, barras, pie, mapas, entre muchos otros. Cada una de ellas ofrece un gran control sobre la forma en que son presentadas de modo que es posible lograr casi cualquier resultado deseado.

Desde el punto de vista del programador es muy fácil de usar ya que solo requiere especificar un "objeto de configuración" y uno o mas arreglos conteniendo los datos a desplegar, es posible obtener distintas representaciones de los mismos datos solo cambiando el objeto de configuración.

2.6.10 Google Maps

Google Maps es un servicio de mapas web el cual ofrece distintos tipos de mapas como mapas viales, mapas de relieve, imágenes satelitales e incluso imágenes de calles en tercera dimensión (llamado Google Street View)

Los mapas de Google Maps son dinámicos, permitiendo al usuario desplazarse, hacer *zoom* y cambiar el tipo de mapa a voluntad, Google Maps funciona dividendo el espacio mostrado en sectores que son descargados individualmente, de manera que cuando el usuario desplaza el mapa solo es necesario descargar los nuevos sectores desde los servidores de Google Maps.

Google Maps es una de las herramientas mas populares para dibujar mapas web ya que ofrece un API gratuito y fácil de usar que permite enmarcar mapas en cualquier sitio web con solo algunas líneas, además los mapas de Google son de alta calidad y se mantienen constantemente actualizados.

2.6.11 Geo-localización IP

La geo-localización IP consiste en asignar a un IP la localización geográfica de la máquina anfitriona correspondiente[31].

Existen dos paradigmas principales para aproximar la localización geográfica de una dirección IP: activo y pasivo; las técnicas activas de localización se basan en mediciones de retraso y en muchos casos proveen resultados precisos. El paradigma pasivo consiste

del uso de bases de datos que contienen rangos de direcciones IP a los que se les llaman bloques o prefijos relacionados a una localización geográfica especifica, sin embargo su precisión puede estar sujeta a errores substanciales. En ambos casos es imposible conocer la localización exacta asociada a una dirección IP sin la colaboración activa de los anfitriones finales. Sin embargo, es posible hacer buenas aproximaciones en algunos a nivel de ciudades o países.

Ya que conocer la localización de sus clientes a partir de su dirección IP es útil para muchos servicios (por ejemplo, para conducir anuncios localizados) existe una gran variedad de soluciones de geo-localización tanto gratuitas como de pago. En este trabajo aprovechamos servicios gratuitos para dibujar mapas que muestran visualmente la ruta de un paquete a través de un enlace.

2.6.12 AJAX

AJAX (*Asynchronous JavaScript And XML*) es una técnica para construir sitios web interactivos a través de peticiones asíncronas con Javascript que mantienen comunicación con el servidor web para mantener el estado del cliente actualizado sin necesidad de que el usuario tenga que refrescar la pagina o realizar consultas adicionales, de esta manera el usuario tiene una experiencia similar a la que tendría con aplicaciones de escritorio.

A pesar de que el nombre AJAX sugiera el uso XML como lenguaje para la transferencia de datos entre el cliente y el servidor, se puede usar cualquier formato como texto plano, HTML y JSON. En este trabajo, AJAX es usado para permitir la carga de gráficas con distintos datos del lado del cliente, lo cual provee una experiencia dinámica y rápida para la observación de resultados del monitoreo.

2.6.13 APScheduler

APScheduler (*Advanced Python Scheduler*) es una biblioteca Python para retrasar la ejecución de rutinas a un instante dado en el futuro ya sea como eventos de una sola vez o de forma recurrente[32]. Esta biblioteca provee las herramientas para construir cualquier esquema de planificación que se desee implementar.

Su arquitectura consta de los siguientes componentes:

- Gatillos para determinar el momento de ejecución de las tareas (por fecha, por intervalos de tiempo, o tipo *crontab*).
- Almacenes de tareas para alojar los datos de las tareas.
- Ejecutores para controlar la ejecución de las tareas.
- Planificadores que unen los componentes antes descritos y se encargan de ejecutar la lógica e iniciar las tareas en el momento planificado.

2.7 Estado del Arte

A pesar de que el monitoreo de redes no es un nuevo campo de estudio, existe una variedad de soluciones innovadoras y aplicaciones que proveen distintos enfoques para el monitoreo y evaluación de redes.

La mayoría de los sistemas disponibles a través de aplicaciones web o para dispositivos móviles tales como DSL Reports[14], ThinkBroadband[15], y SpeedTest[18], permiten a usuarios determinar el RTT y rendimiento de su servicio de Internet al hacer pruebas contra uno o mas servidores de medición. Estos servicios ofrecen la ventaja de ser gratuitos, sencillos de usar y no requieren la instalación de *software* o *hardware* adicional. Sin embargo, solo dan al usuario una idea limitada del estado de su red, pues los resultados obtenidos dependerán sustancialmente del servidor elegido para realizar la prueba.

Otras aplicaciones similares además de hacer mediciones de latencia y rendimiento, ofrecen algún valor agregado al usuario, por ejemplo, NetRadar[7], está orientado al análisis de calidad de servicio de proveedores de servicio de telefonía e Internet inalámbrico y Netalyzer[35] se especializa en la identificación de problemas en la conexión a Internet como presencia de *proxies*, protocolos bloqueados, errores en los DNS, etc.

El enfoque de los sistemas de *benchmarking* nombrados hasta ahora ofrece información útil y es suficiente para la mayoría de los usuarios, sin embargo, se necesita

tener una infraestructura costosa, con la capacidad suficiente para atender las pruebas realizadas por todos los usuarios, sin que la carga del sistema afecte los resultados de las pruebas negativamente. Los costos de mantener esta infraestructura se justifican por el aprovechamiento de los datos recolectados, ya sea haciéndolos públicos, por motivos de investigación o para venderlos a terceros.

Uno de los proyectos mas ambiciosos en el área del monitoreo de redes es Ripe Atlas[9], este intenta obtener un entendimiento profundo del estado de Internet en tiempo real. Para esto, Ripe Atlas hace uso de dispositivos medidores distribuidos a voluntarios, que conforman una red de mediciones global; estos dispositivos realizan mediciones y pruebas regulares entre sí y hacia servidores DNS. Este proyecto depende de la colaboración de los usuarios para alojar y mantener los dispositivos medidores y poner a la orden parte de su ancho de banda. A cambio, los usuarios obtienen créditos que pueden usar para realizar sus propias pruebas aprovechando toda la red de medición.

Project Bismark[8] es un proyecto orientado a la investigación de la calidad de servicio de Internet en hogares. Los usuarios que deseen participar en el proyecto reciben un enrutador con el *software* de Bismark que analiza el tráfico de la red local y realiza una batería de pruebas de forma regular. En este caso, el usuario se beneficia obteniendo un mayor entendimiento de los patrones de uso de su red y evaluando el servicio prestado por su proveedor de servicio de Internet. Sin embargo, el usuario no tiene control sobre las pruebas realizadas y los recursos utilizados. Por su parte, Project Bismark usa tanto los datos obtenidos por las pruebas regulares como el *hardware* distribuido para hacer investigaciones.

Por otra parte servicios con modelos de negocios mixtos o de pago como Pingdom[10] y UptimeRobot[11] ofrecen un servicio de monitoreo automatizado de la disponibilidad y tiempo de respuesta de servicios web. Estos sistemas funcionan realizando mediciones regulares para determinar el estado del servicio y determinan tiempo de respuesta y eventos relevantes tales como interrupciones de servicio o mensajes de error en las respuestas obtenidas. Estos sistemas son sencillos de usar ya que solo es necesario especificar el URL o dirección IP del servicio a monitorear, pero, las características mas atractivas como alertas automáticas o el ajuste de la frecuencia de las mediciones

no son gratuitas, la batería de pruebas está limitada al protocolo HTTP y solo es posible monitear un número limitado de *hosts*.

En este trabajo, hacemos énfasis en el control sobre el uso de recursos del monitoreo. El usuario puede configurar la frecuencia y tipos de pruebas que se estarán ejecutando. Además, como se trata de una solución auto-gerenciada, el usuario tiene la libertad de desplegar agentes de monitoreo fácilmente gracias a que usamos una solución de *software* que puede ser instalada incluso en una amplia gama de dispositivos.

2.7.1 Caracterización del Estado del Arte

En esta sección resumimos las características de estos sistemas a través de los siguientes parámetros:

1. **Servicio ofrecido:** Define el servicio proveído por el sistema. Por ejemplo: monitoreo de actividad de servicios web, evaluación de calidad de servicio de Internet inalámbrico, etc.

2. **Dispositivos de monitoreo:** Define el tipo de dispositivo que ejecuta las pruebas o actúa como el punto "activo" del monitoreo. Por ejemplo: dispositivos móviles, *hardware* especializado, enrutadores, etc.

3. **Localización de los dispositivos de monitoreo:** Define la localización donde los dispositivos de monitoreo son desplegados.

4. **Referencia de monitoreo:** Define el dispositivo que sirve como la referencia de monitoreo, en otras palabras, un dispositivo que espera por los mensajes del dispositivo de monitoreo y responde apropiadamente; estos no siempre son necesarios ya que muchos de los elementos presentes en la red implementan protocolos que pueden ser usados para propósitos de monitoreo. Ejemplos de estos dispositivos son: nodos en la red como enrutadores o servidores dedicados.

5. **Localización de las referencias de monitoreo:** Define la localización donde las referencias de monitoreo son desplegadas. Esto es importante ya que la distancia entre los distintos dispositivos impacta los resultados de de las pruebas.

6. **Tipo de red:** Se consideraron dos configuraciones de monitoreo de redes: **cliente-servidor** donde un conjunto de clientes ejecutan pruebas en contra de uno o mas servidores dedicados y redes ***peer-to-peer***, donde los entes de monitoreo puedes realizar pruebas entre sí (es decir, actuar como dispositivos monitores y referencias de monitoreo).

7. **Pruebas:** Detalles sobre las pruebas ejecutadas por el sistema. Por ejemplo: pruebas de rendimiento con UDP, trazado de rutas, ICMP ping, resolución de DNS, etc.

8. **Visualizaciones disponibles:** detalles sobre los métodos de visualización de datos disponibles como mapas, gráficas de barras, tablas, etc.

9. **Número estimado de usuarios:** Número estimado de usuarios en el orden de potencias de diez.

10. **Referencias:** Determina si el sistema tiene referencias en *papers* u otras publicaciones.

	Servicio	Dispositivos Monitores	Localización	Referencias de monitoreo	Localización	Tipo de red
Atlas Ripe [9]	Monitoreo de alcanzabilidad, latencia y DNS	*Probes* y *anchors* (hardware especializado)	Distribuidos a nivel mundial	Servidores DNS y *anchors*	Distribuidos a nivel mundial	Peer-to-peer y cliente-servidor
Netradar [7]	Monitoreo de calidad de WISPs	Dispositivos móbiles	Cualquier Lugar	Servidores dedicados	Localizaciones en Europa, Asia y EEUU	cliente-servidor
SpeedTest [18]	Evaluación de latencia y rendimiento	Dispositivos web	Cualquier Lugar	Servidores dedicados	Distribuidos a nivel mundial	cliente-servidor
Guifinet [33]	Evaluación de rendimiento y monitoreo de uso	Nodos centrales de la red guifi.net	Desplegados en la red guifi.net en Barcelona, España	Nodos centrales de la red guifi.net	Desplegados en la red guifi.net en Barcelona, España	Peer-to-peer
Broadband DSL Report [14]	Evaluación de latencia y rendimiento	Dispositivos web	Cualquier Lugar	Servidores dedicados	Localizaciones en EEUU	cliente-servidor
Project BISmark [8]	Monitoreo de calidad de banda ancha en hogares	Enrutadores OpenWRT, Raspberry Pi y Android	Distribuidos a nivel mundial	Servidores dedicados	Distribuidos a nivel mundial	cliente-servidor
Pingdom [10]	Monitoreo de servidores web	Servidores dedicados	Localizaciones en Europa y EEUU	Servidores web	Cualquier Lugar	cliente-servidor
Monitis Visual Traceroute Tool [34]	Evaluación de alcanzabilidad	Servidores dedicados	Localizaciones en Europa, Asia y EEUU	Dispositivos web	Cualquier Lugar	cliente-servidor
Uptime Robot [11]	Monitoreo de servidores web	Servidores dedicados	Localizaciones en Europa, EEUU y Japón	Servidores Web	Cualquier Lugar	cliente-servidor
ICSI Netalyzr [35]	Evaluación de acceso a Internet	Dispositivos web y Android	Cualquier Lugar	Servidores dedicados, DNS y *proxies*	Desconocido	cliente-servidor

Tabla 2.1: Tabla del Estado del Arte parte I

	Pruebas	Visualizaciones	Usuarios	Referencias
Atlas Ripe	Ping, Traceroute y DNS.	Mapas de Internet con Resultados en los *anchors* y *probes*	10.000	si
Netradar	*Throughput* TCP (mono-flujo), medición de RTT pasivo con TCP, GPS geo-localización.	Mapas de calor geográfico con detalles de calidad por área y proveedor de servicio	1.000.000	si
SpeedTest	Throughput TCP (multi-flujos) sobre HTTP	Gráfica de velocidad de subida/bajada vs. tiempo.	100.000.000	si
Guifinet	Desconocido	Mapa de la red con detalles sobre el uso y rendimiento por enlace.	100.000	si
Broadband DSL Report	Throughput TCP (multi-flujos)	Gráfica de velocidad de subida/bajada vs tiempo, gráfica de telaraña de.	1.000.000	no encontradas
Project BISmark	Throughput TCP (multi-flujos) y Ping	Latencia y throughput vs tiempo.	100.000	si
Pingdom	Ping sobre HTTP	Gráfica de latencia vs. tiempo, lista de períodos de actividad e inactividad.	1.000.000	si
Monitis Visual Traceroute Tool	Traceroute	Mapa de la posición geográfica de los saltos	Desconocido	no encontradas
Uptime Robot	Ping sobre HTTP	Gráfica de latencia vs tiempo, lista de periodos de actividad e inactividad.	Desconocido	si
ICSI Netalyzr	Ping, resolución de DNS, *throughput*, detección de NATs, chequeo de correctitud de *proxys*, etc.	Lista de los eventos relevantes durante el monitoreo y problemas encontrados marcados en rojo	10.000	si

Tabla 2.2: Tabla del Estado del Arte parte II

Capítulo 3

Aplicación Web "Optopus Head"

El sistema de monitoreo de redes "Octopus Monitor" consiste de cuatro elementos principales: (1) La aplicación web "Octopus Head" que funciona como coordinador del monitoreo, (2) un conjunto de agentes monitores de red llamados "Tentacle Probe Source" (TPS) que realizan acciones de monitoreo a partir de las instrucciones de Octopus Head, (3) Nodos a monitorear llamados "Tentacle Probe Destination" (TPD), y (4) El servicio de almacenamiento en la nube, que ofrece funcionalidades de paso de mensajes y alojamiento de datos compartido entre los distintos entes.

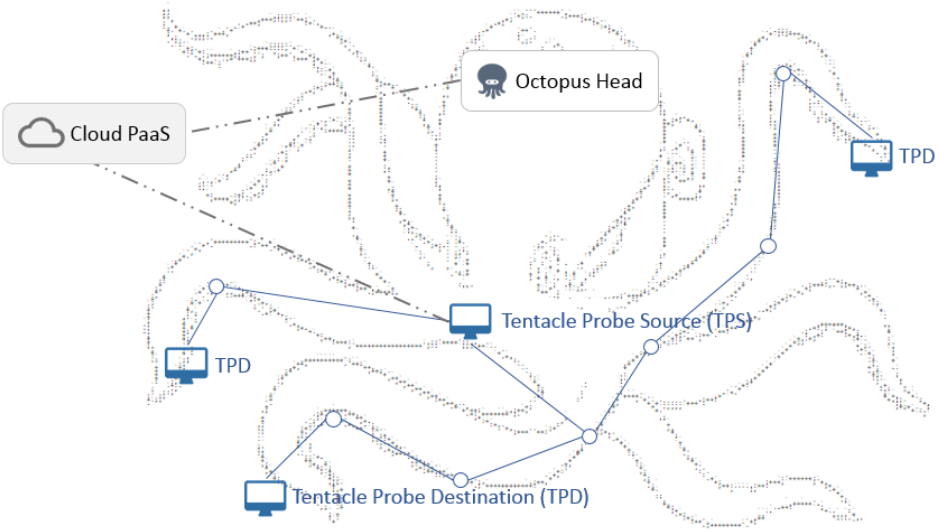

Figura 3.1: Sistema de Monitoreo "Octopus Monitor"

En la Figura 3.1 se puede ver como hemos usado una analogía sencilla para identificar todos los entes que participan en el sistema, la cabeza del pulpo representa a la aplicación web, los tentáculos están conformados por monitores de red Tentacle Probe Sorce, nodos Tentacle Probe Destination, y una cantidad variable de nodos intermedios entre ellos, en otras palabras, los tentáculos conforman los enlaces de interés a a monitorear. Se desea que usuarios no-técnicos puedan entender fácilmente el esquema del sistema y así puedan monitorear sus redes y servicios.

Los monitores de red remotos son los que ejecutan el plan de monitoreo, es decir, el conjunto de pruebas planificadas para ser ejecutadas por un Tentacle Probe Sorce con el fin de recoger datos relevantes de la red. Es en la aplicación web que los usuarios pueden definir (y re-definir) las políticas de monitoreo y sincronización de los datos; toda la comunicación entre la aplicación web y los monitores de red ocurre a través de la nube, la aplicación web publica mensajes destinados a un monitor específico y los monitores a su vez son notificados por el servicio de notificaciones la nube cuando hay nuevos mensajes para ellos, el monitor de red se mantiene al día sobre los cambios realizados al plan de monitoreo y ejecuta fielmente el plan de monitoreo.

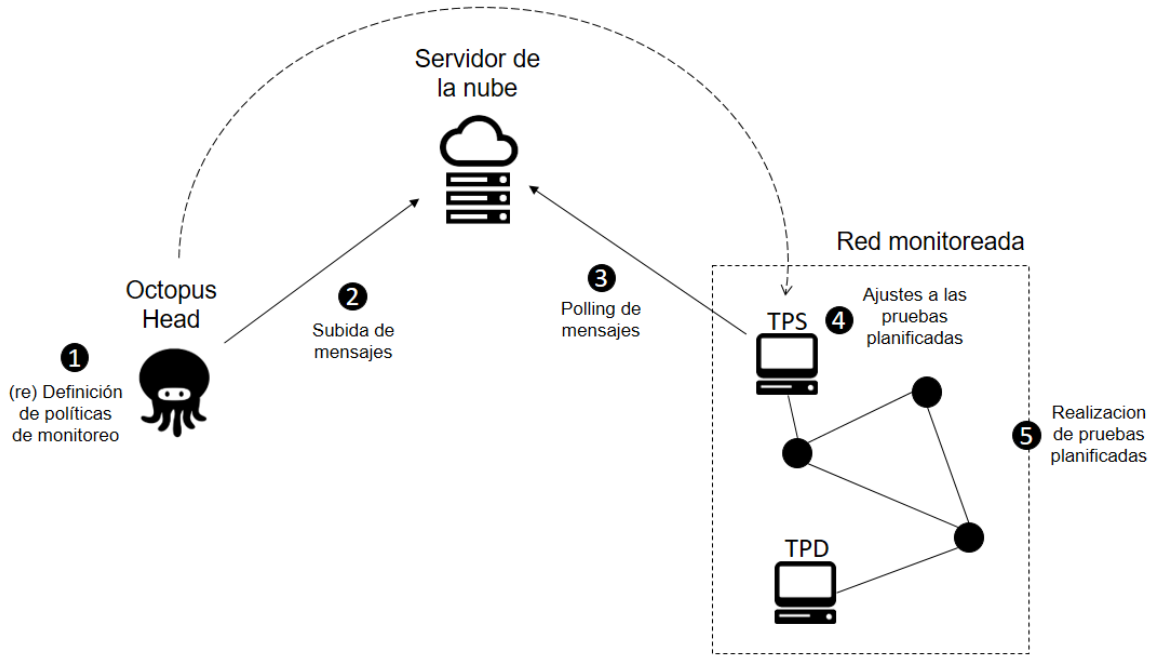

Figura 3.2: Definición de políticas de monitoreo

Este esquema asegura que la aplicación web sirva como interfaz entre el usuario y los monitores, y simplifica el manejo de monitores de red remotos que podrían potencialmente ser abandonados en sitios de difícil acceso. La nube no solo sirve para almacenar datos a bajo costo y a largo plazo, sino que podemos aprovechar sus funcionalidades para mantener cualquier número de monitores actualizados de forma tolerante a fallas sin necesidad de mantener un servicio de notificaciones propio.

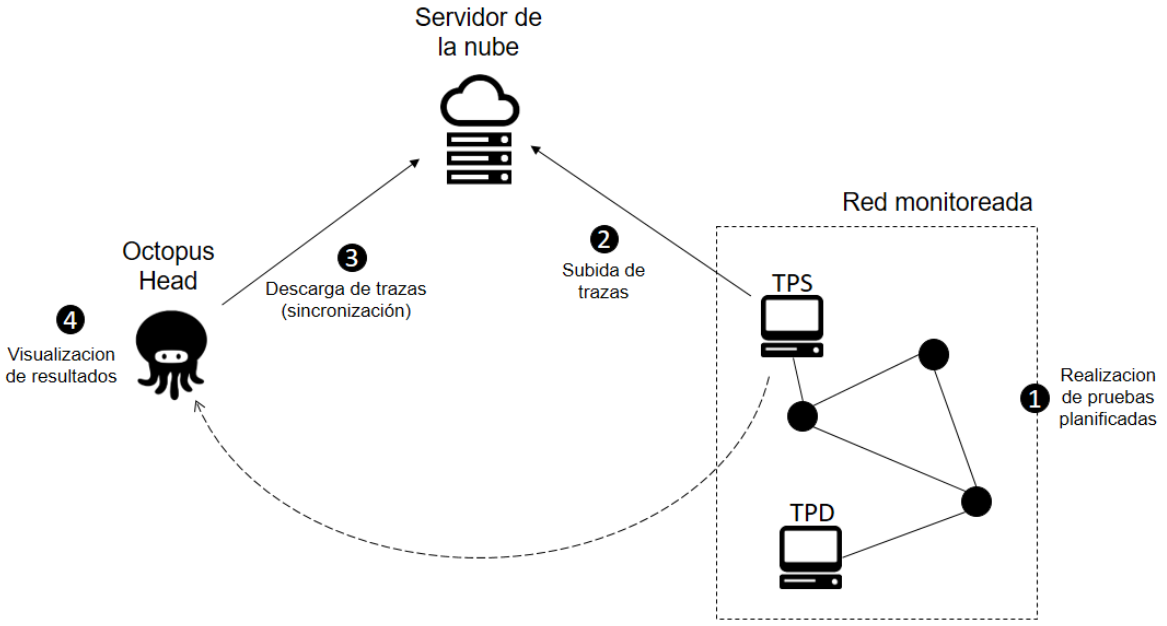

Figura 3.3: Sincronización y visualización de datos

La aplicación web es también la responsable de desplegar visualizaciones a partir de los datos obtenidos del monitoreo, para esto, los monitores constantemente suben los datos obtenidos del monitoreo y la aplicación web a su vez recolecta dichos datos y los procesa en un formato que facilite el cálculo de las visualizaciones, a este proceso lo llamamos **sincronización** y es otra de las tareas esenciales de la aplicación web. El proceso de sincronización y visualización de datos puede ser visto en la Figura 3.3.

A partir de este diseño, es claro que la aplicación web debe poseer una arquitectura que pueda manejar los recursos disponibles de forma eficiente y así poder ofrecer tanto un servicio web rápido como planificar y ejecutar las tareas de sincronización y cómputo de visualizaciones.

3.1 Diseño de la aplicación web

Para el desarrollo de esta aplicación se usó el *framework* de desarrollo web Django que implementa un patrón MVC, ya que nuestra aplicación web se encargará de manejar tareas computacionalmente intensas, o de largo tiempo de ejecución, Django se integró con el sistema de procesamiento de tareas distribuido Celery. Esto no solo con la finalidad de que el servidor web pueda delegar estas tareas y responder rápidamente al usuario, sino también para permitir una mejor escalabilidad del sistema, que entonces podrá responder a un mayor número de peticiones por unidad de tiempo.

3.1.1 Arquitectura

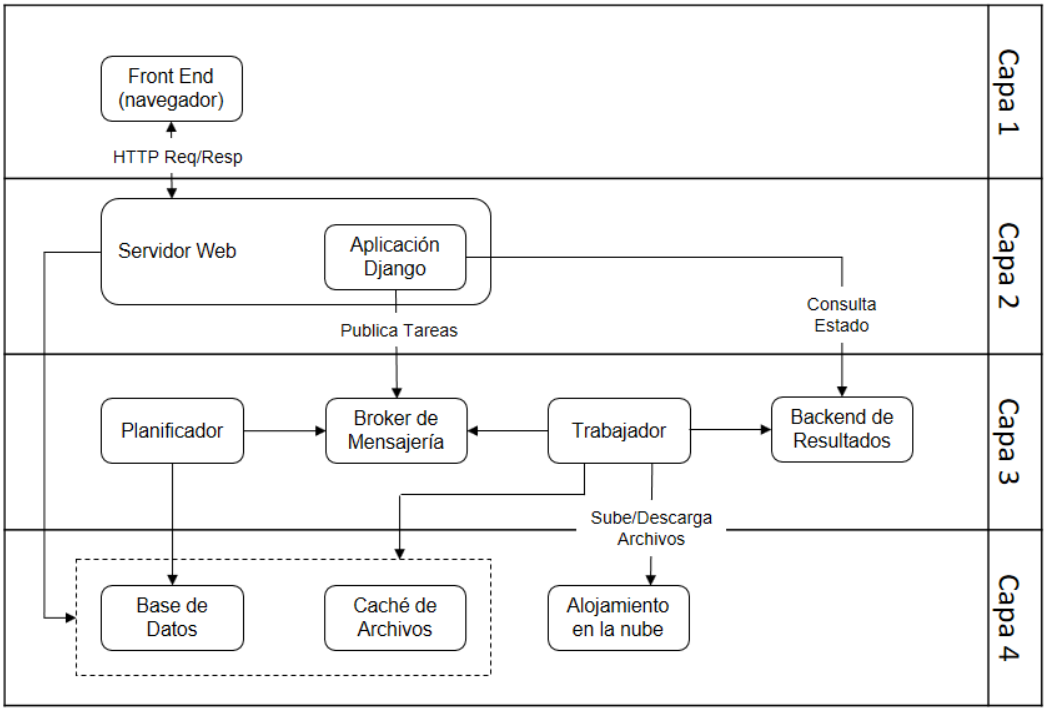

Figura 3.4: Arquitectura de la aplicación web.

La arquitectura de Octopus Head consiste en cuatro capas, la capa superior o capa de presentación se ejecuta en el navegador del cliente monitor, para ver los resultados obtenidos a partir del monitoreo, esta capa se comunica con la capa dos o capa de servicio que es ejecutada por el servidor web, y es responsable de responder a las

peticiones de los usuarios del sistema, la capa tres es la encargada de ejecutar tareas largas o computacionalmente intensas así como de planificar e iniciar tareas periódicas, la capa cuatro o capa de datos aloja los datos de la aplicación como usuarios, monitores, planes de monitoreo, historiales, datos recolectados de las redes, reportes, caché y datos en la nube.

3.1.2 Capa 1: Presentación

La capa de presentación es la interfaz gráfica que permite la interacción del usuario con el sistema de monitoreo. Esta capa consiste en los despliegues en navegadores en cualquier dispositivo conectado a Internet y su propósito es ofrecer al usuario una interfaz gráfica para ingresar al sistema, configurar y manejar monitores TPS, planificar pruebas, observar los resultados obtenidos, entre otros.

La capa de presentación puede ser extendida para permitir la implementación de distintos *front-ends* como aplicaciones nativas para sistemas operativos móviles o de escritorio a través de APIs programáticas.

3.1.3 Capa 2: Servidor Web

El servidor web es el punto de entrada a la aplicación, éste responde a las peticiones realizadas desde el *front-end* a través del protocolo HTTP, el enfoque del servidor web es manejar tareas ligeras de la forma mas rápida posible y retornar respuestas para ofrecer al usuario baja latencia en su interacción con el sistema.

La ejecución de tareas largas y pesadas es delegada a la capa de procesamiento asíncrono de tareas (3.1.4), de esta manera un solo usuario no ocupa los recursos del servidor web durante mucho tiempo independientemente de la petición que realice. Esto hace posible manejar un mayor numero de usuarios que visiten el sitio de forma concurrente, igualmente, el servidor web no se sobrecarga aunque exista una alta carga de trabajos de fondo como computo de visualizaciones y sincronizaciones.

El servidor web debe manejar dos tipos de peticiones: aquellas que requieren respuestas dinámicas ajustadas a los datos específicos que pertenecen a cada usuario particular y peticiones de archivos estáticos como CSS, JavaScript, imágenes o archivos

Ruta	Acción
/static/	servicio de archivos estático
/media/	servicio de archivos estático
/	re-dirección a la aplicación web.

Tabla 3.1: Tabla de enrutamiento del servidor web

HTML pre-definidos que cambien poco o rara vez. A pesar de que la aplicación web es capaz de manejar ambos tipos de peticiones, cada una de ellas tendrá que pasar por todo el *pipeline* de Django (*middlewares*, resolución de URLs, procesamiento de la vista, etc), esta complejidad es innecesaria cuando se trata de archivos estáticos y va a sobrecargar el servidor web en entornos de producción.

Para manejar eficientemente todo tipo de peticiones, debemos implementar un mecanismo que sea capaz de servir los archivos estáticos rápidamente y que sirva como *proxy* entre el *front-end* y la aplicación web, hay una gran variedad de servidores web ligeros como Ngnix[1] o Lighttpd [2] diseñados especialmente con este propósito, estos tendrán una sencilla tabla de enrutamiento que decidirá como manejar las peticiones. La aplicación web podrá correr entonces en un proceso separado y solo atenderá peticiones dinámicas, así se asegurará la eficiencia en el uso de los recursos computaciones disponibles.

3.1.4 Capa 3: Procesamiento asíncrono de tareas

Mantener el servidor web ocupado con tareas largas puede degradar su calidad de servicio y ocupar rápidamente sus recursos disponibles, mas aún muchas veces no es posible dejar al usuario esperando indefinidamente mientras se procesa su petición. Por este motivo es menester tener un sistema de procesamiento asíncrono de tareas, de manera que las tareas se puedan ejecutar en segundo plano y ofrecer una respuesta inmediata al usuario.

Ya que es imposible conocer de antemano el número de tareas que se van a estar ejecutando concurrentemente necesitamos una manera de encolar las tareas para que

[1]Ngnix http://nginx.org/
[2]Lighttpd https://www.lighttpd.net/

puedan ser ejecutadas cuando alguno de los trabajadores se desocupe, de igual forma, para la implementación de sincronizaciones periódicas de los monitores, necesitamos tener un planificador que inicie ciertas tareas según el horario definido por los usuarios.

La inmensa mayoría de las peticiones realizadas por los usuarios serán manejadas directamente por el servidor web, las tareas delegadas a este capa son las siguientes:

- **Conexiones con el servicio de almacenamiento en la nube:** A pesar de que la mayoría de las tareas que incluyen conexiones con el API de la nube son bastante sencillas, estas tareas pueden considerarse de largo tiempo de ejecución ya que es necesario obtener recursos presentes en Internet.

- **Computo de visualizaciones:** El tiempo y poder de cómputo necesario para generar visualizaciones depende altamente del tipo de visualización y la cantidad de datos involucrados, algunas de ellas como mapas de calor, periodos de actividad continua o días y horas activos pueden tardar en el orden de las decenas de segundos en calcularse y requerir una cantidad extraordinaria de computo especialmente cuando se están visualizando periodos de tiempo de meses o años.

- **Sincronizaciones:** Las sincronizaciones combinan varias sub-tareas altamente costosas: La primera de ellas es descargar los archivos de la nube, esto depende de la cantidad de archivos por descargar y del tamaño de los archivos. La segunda es procesar los archivos para convertirlos a un formato adecuado para pasarlos a la base de datos. La tercera y última es la inserción a la base de datos, la cual depende de los indices de la tabla y la cantidad de datos en la base de datos.

El sistema de procesamiento asíncrono de tareas está construido usando Celery (2.6.3), Celery no es solo cada uno de los micro-servicios necesarios para el procesamiento distribuido de las tareas, sino también el protocolo de comunicación entre ellos. Hay que tener en mente que el esquema de Celery **no incluye ninguna autoridad central** que lleve la cuenta de todos los entes participantes. A continuación se explica cada uno de los micro-servicios y su implementación en el marco de Octopus Head.

3.1.4.1 Planificador

El planificador despacha la ejecución de tareas según un horario dado, Celery incluye un planificador llamado Beat que se puede configurar fácilmente para usar la base de datos de la aplicación web a modo de almacén para sus tareas a ejecutar. Beat consulta constantemente una tabla en la base de datos buscando cambios en las tareas planificadas, la aplicación web solo modifica esa tabla y deja que Beat se encargue de iniciar las tareas en el momento adecuado.

Beat no manda a ejecutar las tareas directamente por un trabajador sino que las inserta a la cola de tareas, es por esto que no es posible asegurar que las tareas se ejecuten justo en el momento planificado, sino que puede existir un retraso variable en la ejecución basado en el tamaño de las colas y la carga de trabajo de los trabajadores.

3.1.4.2 Bróker de Mensajería

El bróker de mensajería es un micro-servicio responsable de pasar mensajes entre los entes que encolan tareas (aplicación web y planificador) y los trabajadores que las ejecutan. Es posible enrutar tareas a trabajadores específicos a través del uso de múltiples colas, por ejemplo, sería posible tener una cola para las tareas de sincronización, y otra cola para el computo de visualizaciones y subscribir un distinto número de trabajadores a cada una de ellas[2]. Este sistema asegura una gran granularidad para ejercer control en el orden de ejecución de las tareas, ya que además es posible establecer prioridades, mandando tareas urgentes rápidamente al tope la cola.

Establecer un esquema apropiado de prioridades y encolamiento es esencial para asegurar un tiempo de espera óptimo, las tareas de cómputo de visualizaciones deben tener una muy baja latencia ya que generalmente el usuario está esperando por una respuesta lo más rápida posible, la subida de mensajes al buzón del monitor puede tener una latencia ligeramente mayor, sin embargo no mayor a unos 10 segundos, por otra parte, las sincronizaciones planificadas pueden tener un mayor tiempo de espera en cola donde 10-60 segundos es aceptable[36].

Existen distintos sistemas que pueden hacer el papel de bróker de mensajería tales como Redis (ver Sección 2.6.4), o algún RDBMS. La elección debe depender de volumen

de tareas que se va a manejar, el uso de base de datos puede ser aceptable cuando se maneja una cantidad reducida de tareas, pero Redis es recomendado en entornos de producción.

3.1.4.3 Trabajadores

Los trabajadores son los encargados de ejecutar las tareas encoladas, estos se subscriben a una o mas colas y toman las tareas en el tope, determinar la cantidad necesaria de trabajadores para cada cola es esencial para evitar que las colas crezcan indefinidamente (y por lo tanto también, el tiempo de espera de las tareas).

Tener múltiples trabajadores permite tener un gran control sobre los recursos computacionales usados por el sistema, es posible instanciar nuevos trabajadores ejecutándose en distintas maquinas físicas durante los momentos de alta carga del sistema, y liberar estos recursos cuando no se estén usando, esto hace posible manejar una escala variable y minimiza los costos de operación.

Ya que los trabajadores se pueden estar ejecutando en múltiples máquinas físicas es necesario tener alguna manera de controlarlos remotamente, para esto, los trabajadores se subscriben a una cola de *broadcasting* de alta prioridad en la que se pueden dirigir comandos a todos o a un grupo específico de trabajadores. Existe un conjunto de comandos predefinidos para realizar acciones básicas como revocar tareas, apagar trabajadores o revisar su estado. Es posible implementar nuevos comandos para los trabajadores, por ejemplo, sería posible implementar un comando para que los trabajadores puedan actualizar su propio código y reiniciar, de este modo podrían aceptar nuevas tareas implementadas[2].

3.1.4.4 Backend de Resultados

Puesto que Celery es un sistema distribuido sin una autoridad central que conozca el estado de todo el sistema, es necesario tener algún micro-servicio donde los distintos entes puedan consultar el estado de una tarea. El *backend* de resultados puede ser visto como un espacio de memoria compartido donde se guarda el estado y valor de retorno de las tareas que están siendo o fueron ejecutadas por el sistema. El valor de retorno de las tareas se guarda durante un período de tiempo y luego se desecha para

evitar que el uso de memoria crezca indefinidamente. En este trabajo hemos elegido Redis (2.6.4) como *backend* de resultados.

El *backend* de resultados no solo permite obtener el valor de retorno de una tarea, también hace posible hacer seguimiento del estado de las tareas e implementar mecanismos que indiquen el progreso de alguna tarea particularmente costosa. Los estados predefinidos se pueden ver en la Tabla 3.1.4.4, sin embargo, es posible definir cualquier estado que sea necesario.

Estado	Descripcion	Metadata
PENDING	La tarea está esperando ser ejecutada	-
STARTED	La tarea ha sido iniciada	*pid* y *hostname* del trabajador
SUCESS	La tarea se ejecuto con exito	valor de retorno de la tarea
FAILURE	La tarea no se ejecuto con exito	traza de la exepcion
RETRY	Se está reintando ejecutar la tarea	traza de la última excepcion
REVOKED	La tarea se ha cancelado	-

Tabla 3.2: Estados de las tareas

3.1.5 Capa 4: Datos

La capa de datos incluye todos los sistemas encargados de almacenar o alojar datos de forma persistente ya sea a través de bases de datos, en caché o la nube.

3.1.5.1 Base de Datos

La base de datos es el almacén principal de los datos estructurados de la aplicación web, almacena todo tipo de información como preferencias del usuario, datos de los monitores, horario de sincronizaciones, parámetros de conexión a Dropbox, y detalles de las pruebas implementadas en el *Framework* de Integración de Pruebas del que se hablará en el Capitulo 5.

El diseño de la base de datos es uno de los pilares fundamentales del diseño del sistema, el correcto diseño e indización de las tablas de la base de datos tiene repercusiones importantes en la escalabilidad y tiempo de respuesta del sistema, el diseño de la base de datos se explica a fondo en la Sección 3.1.6.

3.1.5.2 Caché de archivos

El caché de archivos tiene como objetivo almacenar visualizaciones pre-computadas y así evitar la repetición de procesamiento ya realizado y reducir los tiempos de espera del usuario.

Los archivos se guardan en distintos directorios dependiendo del tipo de visualización, cada visualización tiene un código *hash* asociado que se genera a partir de sus parámetros, por lo tanto el acceso a los archivos en el caché es directo. La velocidad de búsqueda en el caché depende de la implementación del sistema de archivos (por ejemplo ext3, ext4, XFS), los sistemas de archivos modernos pueden manejar cientos de miles de archivos en el mismo directorio sin ningún problema de rendimiento. Usar un caché de archivos ciertamente es mas lento que usar un caché en memoria, pero el caché de archivos es capaz de almacenar una cantidad mucho mayor de datos a un menor costo.

3.1.5.3 Almacenamiento en la nube

El almacenamiento en la nube sirve como intermediario entre la aplicación web OH y los monitores remotos TPS, la nube ofrece muchas ventajas con respecto a la escalabilidad y facilidad de implementación del sistema. Usamos el almacenamiento en la nube para guardar los archivos de trazas generados por los monitores en las redes remotas y también a modo de "buzón" para enviar mensajes a los monitores remotos.

3.1.6 Diseño de la base de datos

La base de datos de Octopus Head modela un conjunto de monitores junto con sus enlaces monitoreados (tentáculos), así como los resultados obtenidos para cada enlace (o monitor) de cada una de sus pruebas. De igual manera modela las pruebas y sus parámetros y el historial de cambios en el plan de monitoreo. Puesto que Django usa un ORM (2.6.5.2) para manejar la base de datos, desde el punto de vista de la aplicación web, cada tabla de la base de datos es una clase por lo que es posible aprovechar todas las características de la programación orientada a objetos: herencia de clases, clases abstractas, implementación de métodos específicos a un modelo, y sobrecarga de

métodos de los padres.

A continuación se describen los modelos que forman parte de la base de datos:

Cada usuario registrado está representado por el modelo *User*, este guarda la información necesaria para autenticar al usuario y sus permisos. Cada usuario tiene un *UserProfile* o perfil de usuario con una relación uno-a-uno con el modelo *User*. El perfil de usuario aloja los detalles adicionales del usuario tales como credenciales de Dropbox y códigos de confirmación.

El modelo *Monitor* representa un Tentacle Probe Source. Cada usuario puede tener uno o mas TPS, y a su vez, cada TPS tiene un número de enlaces monitoreados, un plan de monitoreo así como un horario de sincronización y sus detalles específicos como zona horaria, posición geográfica y dirección IP.

Los enlaces monitoreados son un concepto fundamental en el enfoque de monitoreo de este trabajo, cada uno de ellos consiste de un Tentacle Probe Source (modelo *Monitor*) y un Tentacle Probe Destination (un nodo monitoreado) y cualquier número de nodos intermedios entre ellos, para representar esto usamos el modelo *Link* que guarda el IP del Tentacle Probe Destination y otros detalles como posición geográfica, nombre y descripción.

Las pruebas son procedimientos que el usuario puede planificar para que sus TPS ejecuten según una planificación dada, es a esto a lo que llamamos el 'plan de monitoreo'. Una prueba consiste en cualquier código Python ejecutable, generalmente con el objetivo de obtener algún dato de la red monitoreada. Desde el punto de vista de la aplicación web una prueba se modela como una agregación de parámetros configurables.

Los resultados de las pruebas (también llamados trazas), se guardan según las características especificas de cada prueba y no existe ningún limitante a la hora de diseñar modelos para los resultados de las pruebas. Sin embargo, las pruebas implementadas hasta ahora tienen en común una marca de tiempo y el enlace relacionado a la prueba. Adicionalmente, sería posible guardar trazas que no estén relacionadas a ningún enlace (por ejemplo, resultados del sondeo de el número de errores en una interfaz del TPS).

El modelo *MonitorTest* relaciona los TPS con las pruebas a ejecutar. Este modo

contiene una referencia al TPS que ejecutará, el tipo de prueba, la lista de parámetros, el tiempo inicial y final, el intervalo entre pruebas (si lo hay) y el estado (activa o inactiva). En otras palabras, el conjunto de *MonitorTest* para un TPS conforman su plan de monitoreo.

Ya que el usuario tiene la libertad de modificar el plan de ejecución en cualquier momento, se guarda un historial de los cambios hechos a las instancias del modelo *MonitorTest*, para esto usamos el modelo *MonitorTestHistory*. De esta manera, es posible reconstruir con que parámetros estaba siendo ejecutada una prueba en cualquier momento especifico, esto no solo con fines informativos para el usuario, sino también puede ser útil para algoritmos de análisis de los datos que necesiten conocer los parámetros de ejecución.

El objetivo de recoger datos a través de pruebas es poder hacer análisis sobre esos datos para obtener información relevante y sacar conclusiones. El modelo *Visualization* representa cualquier tipo de visualización de resultados, tales como mapas de calor, gráficas de barras, mapas geográficos de la red, etc.

Las sincronizaciones consisten en recoger los datos dejados por los monitores remotos en la nube e insertarlos a la base de datos, generalmente haciendo algún pre-procesamiento o *parsing*, se guarda un historial de sincronizaciones como forma de informar al usuario de la cantidad de archivos recolectados y si ocurrieron errores. Adicionalmente, el mecanismo de sincronización depende de un cursor que indica al sistema cuales son los archivos nuevos o modificados que deben ser insertados, el mecanismo de sincronización se explicará a fondo en la Sección 3.2.3.

El planificador depende de la base de datos para determinar el horario de sincronizaciones, por lo que es necesario un modelo para guardar el horario de sincronizaciones de cada TPS, ya sea este por intervalos con el modelo *IntervaSchedule* o a una hora especifica del día con el modelo *CustomSchedule*.

Los reportes son una forma de mostrar y compartir conjuntos de visualizaciones en una sola vista, de esta manera es posible hacer comparaciones de gráficas de distintos enlaces, o incluso de distintos monitores y además compartirlas con usuarios no autenticados haciendo los resultados públicos.

3.1.6.1 Diagrama de entidad-relación

La figura 3.5 muestra las entidades relacionadas a los usuarios, monitores, enlaces y resultados de las pruebas implementadas.

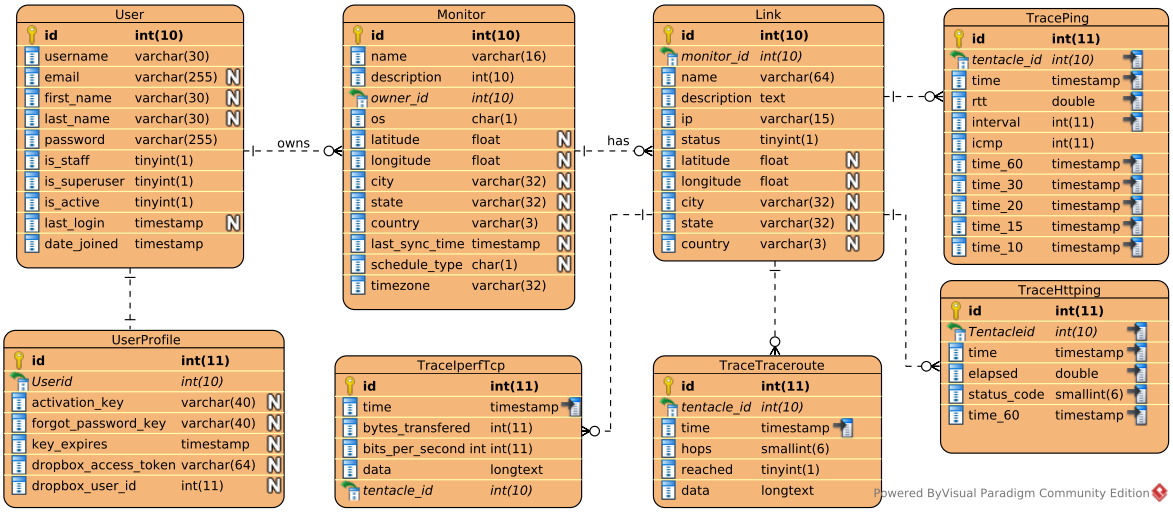

Figura 3.5: Entidades principales de la base de datos

La figura 3.6 muestra las entidades relacionadas al plan de monitoreo, así como su relación con la entidad monitor.

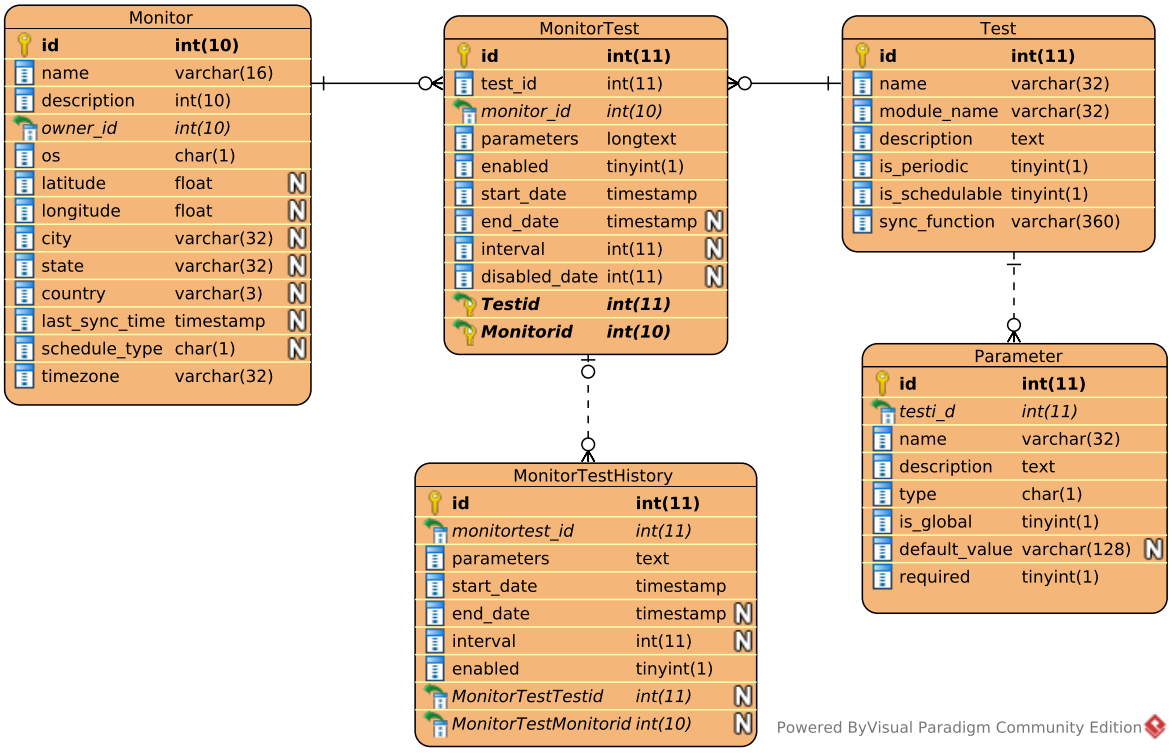

Figura 3.6: Entidades del plan de monitoreo

La figura 3.7 muestra las entidades relacionadas a mecanismo de sincronizacion y horario de sincronizaciones del monitor.

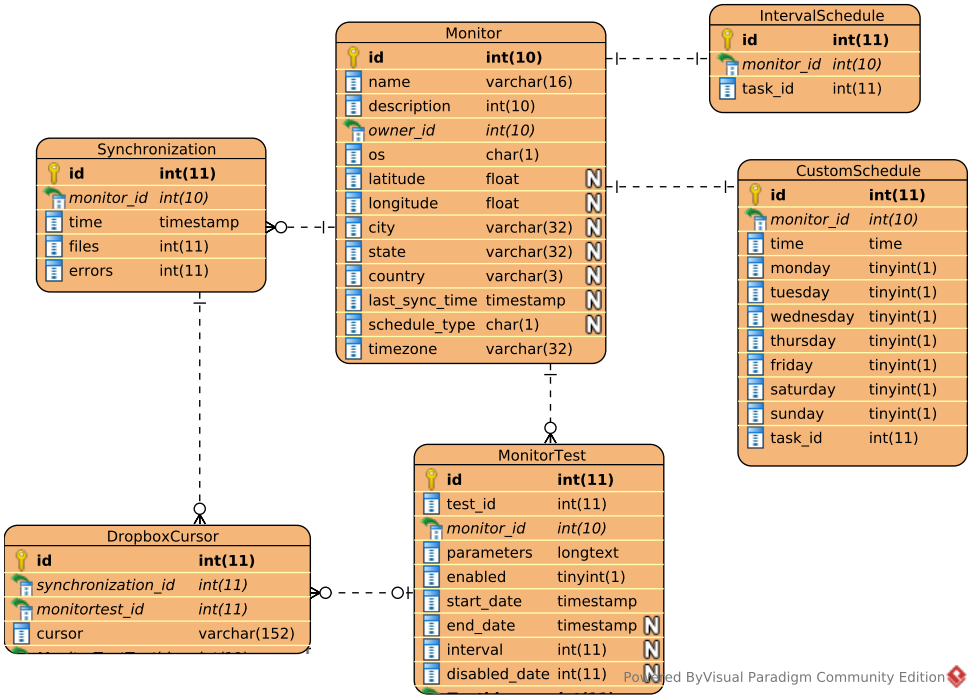

Figura 3.7: Entidades de sincronizaciones

La figura 3.8 muestra las entidades relacionadas a las vistas de resultados y reportes.

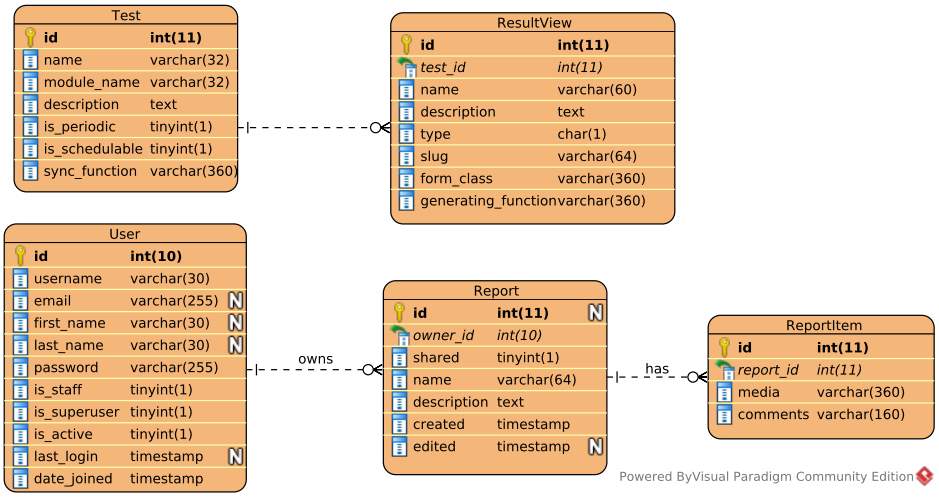

Figura 3.8: Entidades de visualizaciones y reportes

3.1.6.2 Entidades

Las siguientes tablas resumen las entidades y la función de las entidades que forman parte de la aplicación web.

Tabla 3.3: Entidades relacionadas al manejo de usuarios

Nombre de la entidad	**Función**
User	Almacena credenciales e información básica del usuario.
UserProfile	Almacena credenciales de dropbox y códigos de confirmación del usuario .

Tabla 3.4: Entidades relacionadas a los monitores

Nombre de la entidad	**Función**
Monitor	Almacena detalles del Tentacle Probe Source como dirección ip, posición geográfica y zona horaria.
Link	Almacena detalles del Tentacle Probe Destination como dirección IP, posición geográfica y si está activo para el monitoreo.
Test	Almacena el nombre, descripción y tipo de una prueba.
Parameter	Almacena el nombre, descripción tipo de dato y otros detalles sobre un parámetro relacionado a una prueba.
MonitorTest	Tabla intermedia entre monitor y test que almacena una prueba planificada para un monitor, así como su tiempo inicial, final, su estado y parámetros.
MonitorTestHistory	Almacena los cambios realizados a una prueba planificada así como sus parámetros y el período en que estos fueron efectivos.

Tabla 3.5: Entidades relacionadas a los resultados

Nombre de la entidad	**Función**
TracePing	Almacena el resultado de una prueba de ping como marca de tiempo, RTT y TPD relacionado.
TraceHttping	Almacena el resultado de una prueba de httping como marca de tiempo, tiempo de respuesta, código de estado y TPD relacionado.
TraceTraceroute	Almacena el resultado de una prueba de traceroute como marca de tiempo, número de saltos, alcanzabilidad y TPD relacionado.
TraceIperf	Almacena el resultado de una prueba de iperf como marca de tiempo, dirección del flujo, rendimiento obtenido en bytes por segundo y TPD relacionado.

Tabla 3.6: Entidades relacionadas al planificador

Nombre de la entidad	**Función**
Syncronization	Almacena el resultado de una planificación y algunos detalles como número de archivos recolectados y número de errores.
DropboxCursor	Almacena un cursor(**??**) obtenido de la última sincronizacion de un monitor.
IntervalSchedule	Almacena el intervalo entre sincronizaciones de un monitor.
CustomSchedule	Almacena la hora de y los días de la semana en que debe sincronizarse un monitor.

Tabla 3.7: Entidades relacionadas a las visualizaciones y reportes

Nombre de la entidad	**Función**
ResultView	Almacena el nombre y descripción de una visualización así como su tipo y slug.
Report	Almacena el nombre y descripción de un reporte, su dueño y tipo (publico o privado).
ResultItem	Almacena el url relacionado a una visualización que es parte de un reporte y comentarios.

3.1.6.3 Relaciones

- **users - userprofiles (1-1)**: Un usuario solo puede tener un perfil de usuario.
- **users - monitors (n-1):** Un usuario puede ser dueño de uno o mas monitores, pero un monitor solo puede tener un dueño.
- **monitors - links (n-1):** Un monitor puede tener uno o mas tentacles, pero un tentacle solo puede estar en un monitor.
- **links - traces (n-1):** Un enlace puede tener cero o mas trazas relacionadas, pero una traza solo puede estar relacionada a un enlace.
- **test - parameter (n-1):** Una prueba puede tener uno o mas parámetros, pero un parámetro solo puede ser parte de una prueba.
- **test - monitor (n-n):** Una prueba puede ser planificada una o mas veces en uno o mas monitores.
- **monitortest - monitortesthistory (n-1):** Una instancia de monitortest puede tener una o mas entradas en el historial, pero una entrada en el historial solo se refiere a un monitortest.
- **monitor - synchronization(n-1):** Un monitor puede ser sincronizado cero o mas veces pero una sincronizacion solo puede estar relacionada a un monitor.

- **synchronization - dropboxcursor (n-1):** Una sincronizacion puede tener uno o mas cursores, pero un cursor solo pertenece a una sincronizacion.
- **monitor - intervalschedule (1-1):** Un monitor solo puede tener un horario de sincronizacion en intervalo y este solo puede pertenecer a un monitor.
- **monitor - customschedule (1-1):** Un monitor solo puede tener un horario de sincronizacion personalizado y este solo puede pertenecer a un monitor.
- **test - resultview (n-1):** Una prueba puede tener una o mas vistas de resultados, pero una vista de resultados solo puede pertenecer a una prueba.
- **user - report (n-1):** Un usuario puede ser dueño de cero o mas reportes y un reporte solo puede pertenecer a un usuario.
- **report - reportitem (1-1):** Un reporte puede tener uno o mas items y un item solo puede pertenecer a un reporte.

3.1.7 Diseño de Pantallas

Todas las pantallas del sistema fueron creadas bajo el paradigma del diseño web adaptable, es decir que pueden ajustarse a cualquier resolución de pantalla sin reducir la usabilidad o sacrificar la experiencia del usuario, el servidor no tiene que decidir entre un conjunto de plantillas para distintas resoluciones sino que envía al cliente un solo documento HTML y este combina las reglas de presentación de archivos CSS y la lógica en archivos JavaScript para desplegar una pagina web adaptada al dispositivo del usuario.

Ya que una aplicación web está compuesta por decenas de vistas que comparten componentes como barras de navegación, cabeceras, y pies de pagina, Django incluye un micro-lenguaje de plantillas con funcionalidades de herencia e inclusión de plantillas y estructuras de repetición y decisión, esto hace que sea posible tener una taxonomía de plantillas, de manera que solo es necesario escribir los elementos comunes a un conjunto de plantillas una vez y crear nuevas pantallas en el tope de otras. Se le llama *rendering* al proceso de combinar los datos de la base de datos y la petición del usuario, junto con una plantilla predefinida en la aplicación para generar páginas web personalizadas.

Todas las plantillas heredan de una plantilla base, esta incluye la declaración del archivo HTML, la cabecera y define bloques que pueden ser sobrescritos por las plantillas "hijas". Los bloques que define la plantilla base son: **(1) bloque de título** permite definir el titulo de la pagina, **(2) bloque de estilo** permite reemplazar o agregar archivos de estilo/JavaScript, **(3) bloque de modales** permite introducir código antes del cuerpo del documento, **(4) bloque de contenido** bloque principal para el contenido de la pagina, **(5) bloque de JavaScript** bloque para incluir archivos y métodos JavaScript. La plantilla base es a su vez heredada por la plantilla "dashboard" que incluye el menú de navegación lateral y a su vez esta es heredada por la plantilla "monitor" que agrega el sub-menú del monitor.

A continuación se muestra el diseño de las principales pantallas de la interfaz de usuario del servicio web.

- **Pantalla Inicial del sistema,** esta es la pantalla principal de la aplicación web y sirve como punto de inicio donde los usuarios no-autenticados pueden registrarse, iniciar sesión y obtener información básica sobre el sistema.

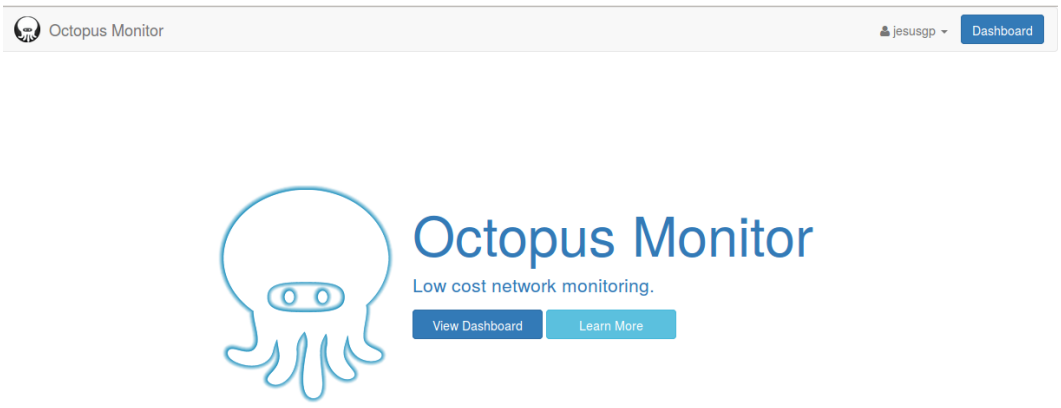

Figura 3.9: Pantalla Inicial del sistema

- **Pantalla de registro de usuario,** en esta pantalla un usuario que desee utilizar el sistema de monitoreo introduce su nombre de usuario, contraseña y correo electrónico para registrarse.

- **Pantalla de inicio de sesión,** en esta pantalla un usuario autenticado puede introducir su nombre de usuario y contraseña para iniciar sesión, también se incluyen enlaces para recuperar contraseña o crear una nueva cuenta en caso de no tener una.

- **Pantalla "Home",** esta es la pantalla de bienvenida al sistema para un usuario autenticado, esta muestra algunas estadísticas rápidas del uso del sistema (como número de monitores, tentáculos, pruebas, sincronizaciones ejecutada, etc), así como enlaces a las principales subsecciones de cada monitor. Todas las pantallas a partir de ahora comparten un menú de navegación superior con enlaces al home, y funcionalidades para ver el perfil de usuario y hacer *logout* y un menú de navegación lateral con enlaces a las principales secciones del sistema.

- **Pantalla de selección de monitores,** esta pantalla muestra los monitores del usuario y le permite seleccionar alguno de ellos para dirigirlo a la pantalla del monitor, además incluye un botón para crear nuevos monitores.

- **Pantalla de creación de monitores,** en esta pantalla se introducen del detalles del monitor a crear como nombre, descripción, dirección IP, sistema operativo del dispositivo *host* y zona horaria del monitor.

- **Pantalla del monitor,** esta es la pantalla principal del monitor donde se muestran los detalles del monitor, un mapa de la red mostrando la localización de los distintos dispositivos de red (TPS y TPD) generado a partir de la información suministrada por el usuario y una lista de verificación para ayudar al usuario a seguir los pasos para monitorear una red. Todas las vistas relacionadas a un monitor específico despliegan un sub-menú de navegación para visitar las distintas subsecciones de manejo del monitor como tentáculos, resultados y sincronizaciones.

- **Pantalla de tentáculos,** esta pantalla muestra los detalles de los distintos tentáculos de un monitor, y incluye enlaces para registrar nuevos tentáculos y editar los detalles de los existentes.

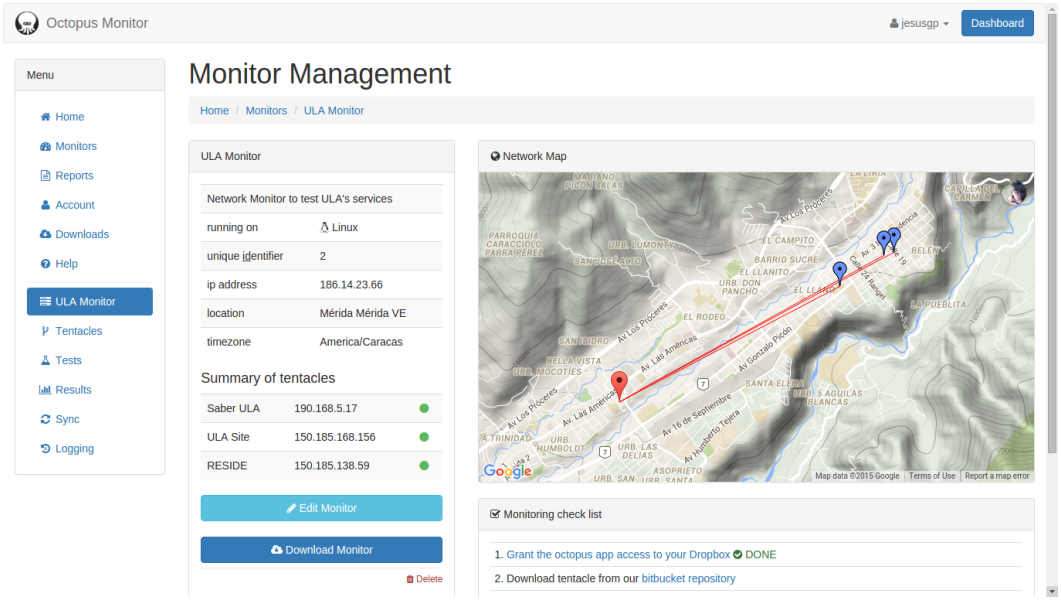

Figura 3.10: Pantalla del monitor

- **Pantalla de Plan de monitoreo,** esta pantalla permite ver el plan de monitoreo por tipo de prueba, incluye enlaces para editar pruebas planificadas y para planificar nuevas pruebas ya sean periódicas o de una sola vez. En caso de que hayan muchas pruebas planificadas separa el contenido en páginas numeradas.

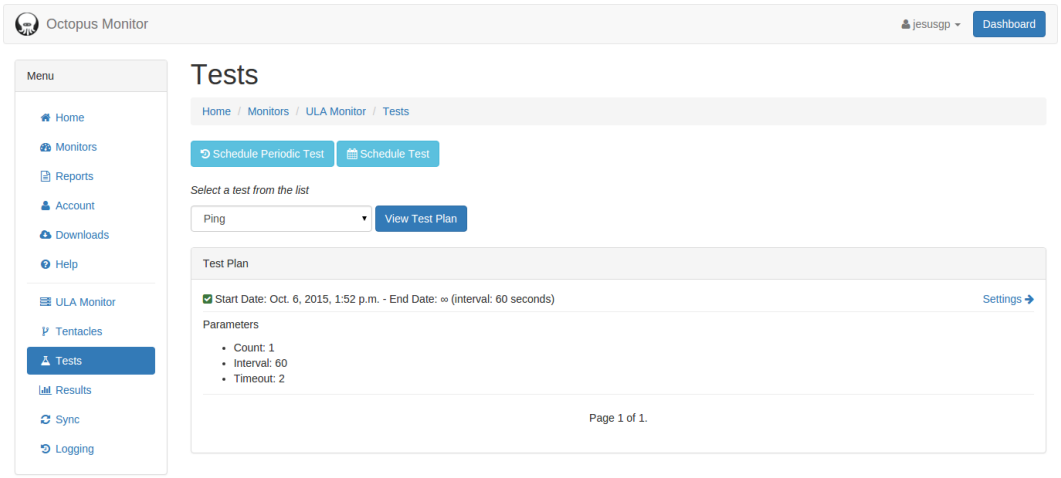

Figura 3.11: Pantalla de Plan de monitoreo

- **Pantalla de planificación de pruebas,** esta pantalla permite agregar pruebas al plan de monitoreo, si se trata de una prueba de una sola vez permite elegir la hora y fecha de ocurrencia, en el caso de pruebas periódicas permite elegir fecha inicial, fecha final e intervalo entre pruebas, en todos los casos se incluye el formulario para fijar los parámetros de ejecución de la prueba.

- **Pantalla de detalles de la prueba,** esta pantalla muestra los parámetros activos de una prueba planificada en un formulario que se puede editar, así como opciones para habilitar o deshabilitar. En el panel lateral se muestra el historial de la prueba, mostrando los períodos en que estuvo inactiva y sus distintos parámetros en períodos de actividad, así como enlaces para volver a activar parámetros anteriores en cualquier punto de la historia.

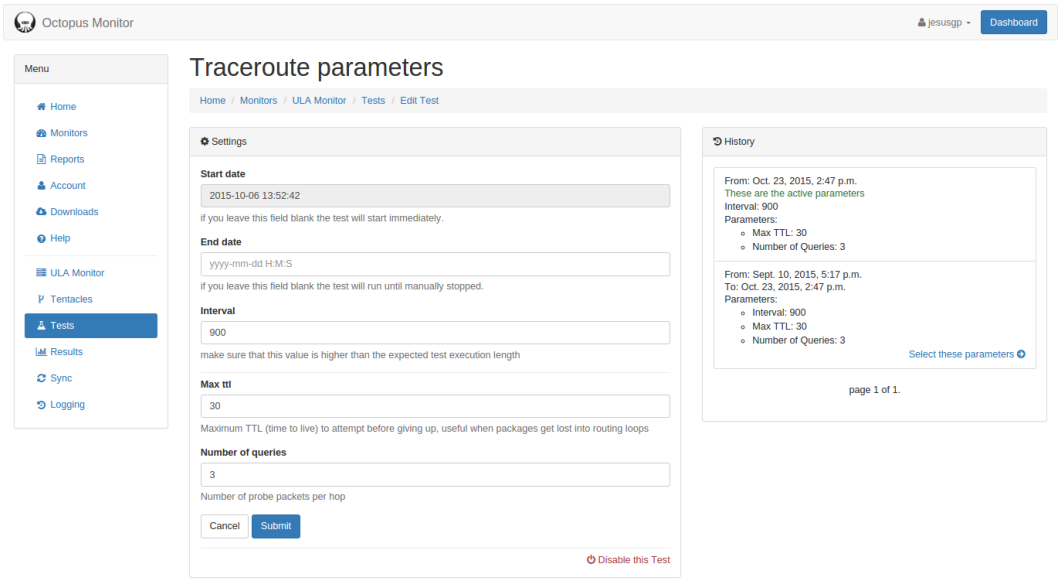

Figura 3.12: Pantalla de detalles de la prueba

- **Pantalla de resultados,** esta pantalla muestra todas las visualizaciones disponibles para un monitor (a partir de las pruebas activas o planificadas en algún momento) y una barra de búsqueda para filtrar los resultados.

- **Pantalla de visualización de resultados,** esta pantalla muestra un formulario para introducir los datos a visualizar y una barra de herramientas, con opciones

para re-computar la visualizaciones, obtener enlaces directos para compartir o agregar la visualización a un reporte dado. Esta pantalla muestra un animación de carga para indicar que una gráfica se está computando o retirando del servidor y mensajes de error en caso de que el servicio no esté disponible o un problema haya aparecido.

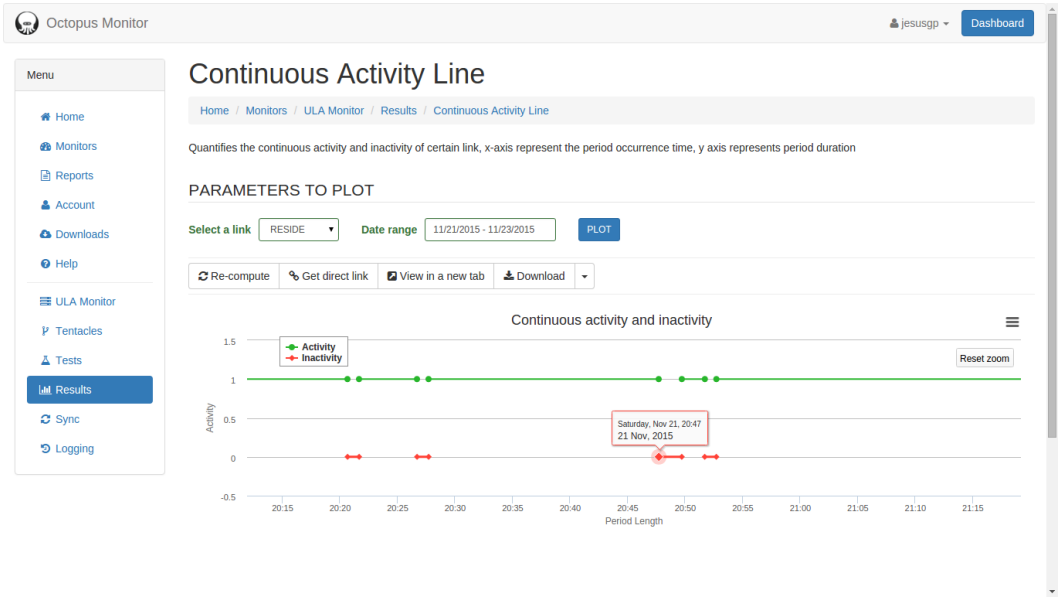

Figura 3.13: Pantalla de visualización de resultados

- **Pantalla de sincronizaciones,** esta pantalla muestra el tiempo desde la última sincronización y el tiempo para la próxima así como un botón para forzar una sincronización inmediata, también muestra un formulario para definir el horario de sincronización y en un panel lateral muestra el historial de sincronizaciones, este ofrece información sobre la cantidad de archivos recolectados y el numero de errores encontrados.

- **Pantalla de reportes,** esta pantalla muestra una tabla de los reportes del usuario con enlaces para ver cada uno de ellos y su última fecha de modificación así como un botón para ir al formulario de creación de reportes

- **Pantalla de visualización y edición de reporte,** esta pantalla puede desplegarse de distintas maneras según el usuario que la observe, si se trata del

usuario al que pertenece el reporte entonces este verá una barra de herramientas para agregar visualizaciones al reporte, cambiar la privacidad del reporte (publico o privado) o editar y borrar, del mismo modo podrá editar los comentarios de cada una de las visualizaciones o quitarlas del reporte. Un usuario que no sea dueño del reporte solo podrá leer su contenido y observar las visualizaciones.

- **Pantalla de cuenta del usuario,** esta pantalla muestra los detalles de la cuenta del usuario, así como enlaces para cambiar la contraseña y renovar el *token* de acceso a Dropbox en caso de que en algún momento este haya sido revocado.

3.1.8 Diseño de URLs

Ya que la aplicación web consiste de un largo conjunto de vistas, debemos diseñar URLs que apunten a cada una de ellas de forma ordenada y lógica. Para esto, se ha usado una convención en la que la unidad principal de operación es una colección de objetos, como monitores, o reportes, a continuación se muestran ejemplos de las reglas usadas para diseñar los URLs:

- Operaciones sobre colecciones:
  - **GET /monitors** retorna la lista de monitores.
  - **GET /monitors/new** retorna un formulario para crear un nuevo monitor.
  - **POST /monitors/new** envía los campos para crear un nuevo monitor.
- Operaciones sobre un registro:
  - **GET /monitor/1** retorna el primer monitor.
  - **GET /monitor/1/edit** retorna un formulario para editar el primer monitor.
  - **POST /monitor/1/edit** envía los campos para editar el monitor 1.
  - **GET /monitor/1/delete** destruye el primer monitor y todos sus registros relacionados.

- Operaciones sobre relaciones de un registro:
  - **GET /monitor/1/links** retorna la lista de enlaces del monitor.
  - **GET /monitor/1/link/7/edit** retorna un formulario para editar el enlace con id #7 del primer monitor.
  - **POST /monitor/1/link/7/edit** somete los campos para editar el enlace con id #7 del primer monitor.
- Invocar operaciones especiales sobre un registro:
  - **GET /monitor/1/sync/now** manda a sincronizar el primer monitor inmediatamente.
  - **GET /monitor/1/test/12/toggle** activa o desactiva la prueba con id #12 del primer monitor.
  - **POST /monitor/1/form/average-rtt-heatmap/** envía los campos del formulario del mapa de calor de rtt para ser validados por el servidor.

3.2 Componentes

En esta sección se explican los distintos componentes de software que son parte de la aplicación web y su funcionalidad.

3.2.1 Enlazador de cuentas de Dropbox

Para realizar todas las tareas relacionadas con el almacenamiento remoto en la nube, la aplicación web hace peticiones a nombre de un usuario de Dropbox, sin embargo primero es necesario realizar un proceso de autenticación que dará permisos a la aplicación web de realizar estas peticiones.

El proceso de autenticación es un algoritmo a dos pasos que se puede observar en el Algoritmo 1:

1. Si la petición es de tipo GET, entonces la aplicación web inicia el proceso de enlazado con Dropbox:

Algoritmo 1 Enlazamiento con Dropbox

```
1: procedimiento LINK_DROPBOX_ACCOUNT(request)
2:     si request.method = 'POST' entonces
3:         form ← DropboxCodeForm
4:         si form es valido entonces
5:             flow ← DropboxOAuth2FlowNoRedirect
6:             access_token , user_id ←flow.finish(form.code)
7:             profile ← get(UserProfile, id = request.user.id)        ▷ Retira un objeto
   UserProfile por su id de la base de datos
8:             profile.dropbox_access_token ← access_token
9:             profile.dropbox_user_id ← user_id
10:            profile.save()                    ▷ Guarda los cambios en la base de datos
       devolver HttpResponseRedirect("/dropbox/link-account/done")
11:    si no
12:        flow ← DropboxOAuth2FlowNoRedirect
13:        url ← flow.start
14:        form ← DropboxCodeForm
       devolver render(request,"link_dropbox.html",form)
```

(a) La aplicación web solicita a Dropbox un URL que se usará para que el usuario se autentique con Dropbox y de permisos a Octopus Monitor para hacer cambios en su carpeta.

(b) La aplicación web devuelve un formulario con una caja de texto para introducir un código y un enlace con el URL generado en el paso previo.

(c) El usuario hace click en el enlace el cual lo lleva al sitio de Dropbox, Dropbox pregunta al usuario si desea dar acceso a Octopus Monitor a su carpeta de Dropbox:

i. Si el usuario acepta se genera un *token* único.

ii. Si el usuario no acepta se termina el proceso.

(d) El usuario introduce el *token* y envía el formulario.

2. Si la petición es de tipo POST, entonces el servidor valida el formulario a partir de los datos enviados:

(a) Si el formulario es válido entonces:

i. El servidor llama al método *finish*, este se comunica con Dropbox, valida el *token* y retorna el código de acceso del usuario para nuestra aplicación.

ii. Se guarda en la base de datos el código de acceso del usuario.

iii. El servidor re-direcciona al usuario a una vista indicando que la operación fue exitosa.

(b) Si el formulario no es valido, el servidor devuelve el formulario mostrando los mensajes de error correspondientes; los datos introducidos previamente por el usuario persisten, facilitando la corrección del formulario.

A partir de este momento tenemos los dos elementos necesarios para hacer peticiones al API de Dropbox: los permisos sobre la carpeta en la nube del usuario y el token de acceso a su Dropbox.

3.2.2 Subidor de mensajes

Una de las tareas fundamentales de la aplicación web es mantener a los monitores remotos informados de los cambios hechos al plan de monitoreo por parte de los

usuarios, para esto usamos un concepto similar al de un buzón de correo electrónico, la aplicación web sube archivos a modo de mensajes a una carpeta en la nube que hace las veces de buzón, hay dos tipos mensajes: aquellos relacionados a los enlaces monitoreados (nuevo enlace registrado o cambios a un enlace existente) y los relacionados a las pruebas (nueva prueba planificada, cambios a los parámetros, cambios al intervalo entre pruebas, prueba deshabilitada).

Los mensajes que se suben a la nube se escriben en formato JSON, este formato ofrece grandes ventajas para el desarrollo de aplicaciones distribuidas ya que es fácil de codificar del lado del servidor y fácil de decodificar del lado del monitor es mucho mas ligero que otros formatos como XML y al mismo tiempo es legible para seres humanos, todos las funciones que suben archivos a la nube funcionan de forma simular:

1. Se retira de la base de datos el *token* de acceso a Dropbox del usuario.
2. Se determina el nombre del archivo a subir, concatenando el id del monitor, el nombre de la carpeta buzón y el tipo de archivo e identificador de la prueba o enlace, por ejemplo:
   - **/monitor1/mailbox/test_12:** especifica los detalles de la prueba id #12 del monitor id #1.
   - **/monitor1/mailbox/link_6:** especifica los detalles del enlace id #6 del monitor id #1.
3. Se crea un objetivo tipo archivo o *buffer* en memoria.
4. Se codifica el contenido del archivo en formato JSON, y se inserta en el *buffer*; el contenido será un diccionario con los detalles relevantes a la prueba o enlace.
5. Se sube el archivo a la nube a través del API de Dropbox.

Los mensajes deben contener la información necesaria para que el monitor pueda reconstruir el plan de monitoreo a partir de ellos, la estructura y posibles valores de los mensajes pueden ser vistos en la Tabla 3.8 para los mensajes relacionados a las pruebas y en la Tabla 3.9 para los mensajes relacionados a los tentáculos.

Algoritmo 2 Subir prueba a Dropbox

procedimiento UPLOAD_TEST_FILE(monitor_test,data)
    $user \leftarrow monitor\_test.monitor.owner$ ▷ Retira el usuario relacionado
    $profile \leftarrow$ GET($UserProfile, id = user.id$) ▷ Retira el perfil de usuario
    $client \leftarrow$ DROPBOXCLIENT($profile.dropbox\_access\_token$)
    $directory \leftarrow MONITOR\_FOLDER\_NAME + monitor\_test.monitor.id +$ 'mailbox'
    $filename \leftarrow directory +$ 'test_' $+ monitor\_test.id +$ '.json'
    $buffer \leftarrow$ STRINGIO ▷ Crea un objeto tipo archivo en memoria
    $json \leftarrow$ DUMPS($data$) ▷ Codifica una cadena de caracteres json a partir de un objeto python
    $buffer$.WRITE($json$)
    $client$.PUT_FILE($filename, buffer, overwrite = True$)
    devolver

Clave	Valor	Descripción
disable	true|false	Si es *false* indica que la prueba debe ser desactivada
name	cadena de caracteres	nombre de módulo de la prueba
end_date	fecha|null	fecha final de la prueba, si es *null* la prueba se ejecuta indefinidamente
start_date	fecha	fecha inicial de la prueba
interval	entero positivo|null	Indica el intervalo entre pruebas, si es nulo, la prueba se ejecuta una sola vez
kwargs	diccionario	Conjunto de pares <nombre,valor> que representa los parámetros de la prueba

Tabla 3.8: Contenido del mensaje de prueba planificada

Clave	Valor	Descripción
status	true|false	Si es *false* indica que el enlace debe ser desactivado
ip	cadena de caracteres	dirección IP del TPD del enlace.
id	entero positivo	identificador único del enlace
host	cadena de caracteres	nombre de dominio del TPD del enlace.

Tabla 3.9: Contenido del mensaje de enlace monitoreado

3.2.3 Sincronizador de datos

El objetivo de la realización de pruebas en las redes a monitorear es recolectar una cantidad de datos suficientes para obtener información del estado de la red, si el monitor tiene un plan de monitoreo definido y está realizando pruebas, este dejará en la nube archivos con resultados de forma constante; es trabajo de la aplicación web a su vez recolectar estos archivos e introducirlos a la base de datos en un formato tal que permita generar visualizaciones de la forma mas conveniente posible para el usuario, en el marco de este sistema, a este proceso se le da el nombre de sincronización.

El algoritmo general de sincronización (Algoritmo 3), retira de la base de datos el monitor a sincronizar, el *token* de acceso a Dropbox del usuario y las pruebas activas para el monitor, luego para cada prueba llama a una subrutina que descarga los archivos, los procesa y introduce los resultados obtenidos en la base de datos. Finalmente totaliza el número de archivos recolectados, el número de errores (ya sea errores de *parsing* de los archivos o problemas de conexión a Dropbox), guarda en la base de datos el registro de la sincronización, los cursores y actualiza el tiempo de ultima sincronización del monitor.

Debemos recordar que los resultados de las pruebas se guardan en una carpeta por prueba, donde constantemente se están guardando nuevos archivos, por lo tanto, la rutina de sincronización debe determinar cuales son los archivos nuevos y modificados desde la última sincronización. Para facilitar este proceso, el API de Dropbox provee el método *delta*, que determina los cambios realizados a una carpeta a partir de un momento específico, a continuación puede verse un ejemplo de la respuesta en formato JSON obtenida tras llamar el método delta para una carpeta:

Algoritmo 3 Sincronizar monitor

```
1:  procedimiento SYNCRONIZE(monitor_id)
2:      monitor ← GET(Monitor, id = monitor_id)        ▷ Retira el monitor por su id
3:      profile ← GET(UserProfile, id = monitor.owner.id) ▷ Retira el perfil de usuario
4:      client ← DROPBOXCLIENT(profile.dropbox_access_token)
5:      tests ← Get all active tests for this monitor
6:      cursors ← empty dictionary
7:      files ← 0
8:      errors ← 0
9:      para test en tests hacer
10:         cursor,_files,_errors ← _SYNCRONIZE(test, monitor, client)
11:         Insert cursor on cursors using test as key
12:         files ← files + _files
13:         errors ← errors + _errors
14:     sync ← SYNCRONIZATION(monitor, files, errores)
15:     sync.SAVE
16:     para key,value en cursors hacer
17:         cursor ← DROPBOXCURSOR(value, sync, monitor_test)
18:         cursor.SAVE
19:     monitor.last_sync_time ← get current time
20:     monitor.SAVE
        devolver
```

```
{
    "reset":false,
    "cursor":"AAHskXVmJSRVG_bgh4Oq2VPQqG79nWM26Tl8jSmR...",
    "has_more":false,
    "entries":[
        [
            "/monitor1/ping/link_2_2015_11_02_15.txt",
            {
                "revision":3,
                "rev":"30ec9923d",
                "thumb_exists":false,
                "bytes":0,"modified":
                "Wed, 20 Mar 2013 05:58:43 +0000",
                "path":"/proj1",
                "is_dir":true,
                "icon":"folder_app",
                "root":"app_folder",
                "size":"0 bytes"
            }
        ],
    ]
}
```


Como se puede observar, se trata de un diccionario con cuatro entradas:

- **reset**: determina si se debe "limpiar" el estado local antes de procesar las entradas delta, es verdadero solo durante la primera llamada a delta o en raras ocasiones.

- **cursor**: el cursor es una cadena de caracteres de longitud 64, y representa el estado de la carpeta en un momento dado, se puede usar en llamadas sucesivas de "delta" para obtener los cambios a partir de ese momento específico.

- **has_more**: determina si hay mas entradas delta en la carpeta, en tal caso es

necesario volver a llamar el método delta inmediatamente de nuevo, esto solo es necesario cuando hay grandes cantidades archivos modificados.

- **entries**: es una lista de "entradas delta" cada entrada representa un archivo o carpeta que cambió desde la última llamada a delta, cada entrada es un par <path,metadata> donde *path* es el nombre del archivo y *metadata* es un diccionario conteniendo algunos datos relevantes al archivo, si *metadata* es nulo indica que el archivo fue eliminado.

El algoritmo de sincronización depende del cursor para determinar los cambios desde la última sincronización, este llama al método delta y obtiene la lista de entradas delta luego recorre aquellas entradas delta cuya *metadata* no es nula y le pasa el contenido del archivo a un algoritmo de *parsing*. El *parser* lee el archivo, lo procesa e inserta su contenido en el formato apropiado en la base de datos. Debemos recordar que cada archivo de traza puede tener un formato distinto dependiendo del tipo de prueba al que pertenezca, por lo que también es trabajo del algoritmo de sincronización elegir el *parser* apropiado al cual pasará el archivo.

3.3 Computo de visualizaciones

El computo de las visualizaciones consiste de tres fases comunes a todas las visualizaciones: primero, los datos necesarios son retirados de la base de datos, después estos datos son transformados o pre-procesados de alguna manera, por ejemplo, las marcas de tiempo se localizan según la zona horaria del monitor, o se eliminan valores atípicos de la muestra, y finalmente estos datos se combinan con una plantilla para generar una página HTML.

Muchas de las visualizaciones obtenidas en el sistema se pueden generar trivialmente pasando a la biblioteca de *rendering* de gráficas un arreglo de pares <x,y> por ejemplo RTT vs. tiempo. Sin embargo otras visualizaciones requieren de algoritmos de análisis que hagan algún pre-procesamiento de los datos y saquen algún tipo de conclusión para mostrar al usuario. A continuación se explica el calculo de las visualizaciones no-triviales del sistema.

3.3.0.1 Cálculo de Mapas de Calor

En este trabajo usamos mapas de calor para representar el valor del tiempo de respuesta promedio en relación a la fecha y la hora del día, los valores "altos" están representados por colores cálidos y los valores "bajos" están representados por colores fríos. Cabe destacar que un valor "alto" para un red de fibra óptica puede ser "bajo" para una red satelital, de modo que el cálculo estos valores dependen unicamente de los datos contenidos en la muestra.

Los mapas de calor permiten evaluar rápidamente la fluctuación del RTT durante un día especifico o notar patrones a largo plazo, otra ventaja de los mapas de calor es que es posible observar periodos de inactividad de la red como "espacios en blanco" en la gráfica ya que para estos periodos no hay datos.

Los parámetros de los mapas de calor son los siguientes:

- **Enlace:** el enlace seleccionado para observar
- **Intervalo de agrupación:** la "granularidad" del mapa de calor, a mayor intervalo de agrupación mas muestras se van a promediar por intervalo. El valor máximo es 60 minutos, lo cual resultará en un mapa de calor con 24 puntos por día, el menor es 10 minutos, lo cual resultará en un mapa de calor con 144 puntos por día.
- **Rango de tiempo:** ajusta el período de tiempo seleccionado para mostrar, se puede seleccionar por meses, años o todo el tiempo.

Para calcular el mapa de calor debemos primero filtrar las muestras por enlace y rango de tiempo seleccionado, luego, estas se deben agrupar según el intervalo de agrupación y sacar el promedio de cada grupo, hay muchas formas de realizar esta operación, sin embargo, cuando se trata de grandes conjuntos de datos esta puede ser una operación costosa por lo que preferimos dejar todos los filtros, ordenamientos, agrupaciones, cálculo de promedios, y sumas a la base de datos. Si la tabla está apropiadamente indexada el RDBMS puede realizar dichos cálculos ordenes de magnitud mas rápido que retirando todos los datos y haciéndolos directamente en la aplicación.

timezone	time	time_60	time_10
-4:30	2015-11-17 22:29:42	2015-11-17 17:00:00	2015-11-17 17:50:00
UTC	2015-11-17 22:29:42	2015-11-17 22:00:00	2015-11-17 22:20:00

Tabla 3.10: Ejemplo de los valores de *time_60* y *time_10* para la misma traza en UTC(+0:00) y Caracas (-4:30)

La tabla *TracePing* tiene los campos *time_60*, *time_30*, *time_15*, *time_20* y *time_10*, los cuales tienen un valor pre-calculado de tiempo que aloja el período de agrupación al que pertenece la muestra. Estas columnas facilitan al RDBMS agrupar las muestras en periodos de 60, 30, 20, 15 o 10 minutos respectivamente, mas aún, son guardados en la base de datos tomando en cuenta la zona horaria del monitor, de modo que no es necesario volver a localizar las fechas cuando se está generando el mapa de calor. Un ejemplo de esto puede ser visto en la tabla 3.10.

Para acelerar aún mas el computo, evitamos usar el ORM para hacer consultas a la base de datos, esto se debe a que, a pesar de que el ORM haga las mismas consultas, este desperdicia tiempo y recursos en convertir estos datos crudos en los objetos correspondientes según el modelo de la base de datos, por lo tanto, para este caso es preferible construir nuestras propias consultas SQL.

La consulta realizada para retirar los valores de la base se puede ver a continuación:

```
1 SELECT time_n,AVG(rtt)
2 FROM traceping
3 WHERE link_id=x AND time >= s AND time <= f AND rtt>0
4 GROUP BY time_n
```

Donde:

- **n**: es el período de agrupación
- **x**: es el id del enlace
- **s**: es la fecha inicial
- **f**: es la fecha final

Esta consulta retorna entonces el promedio del valor del RTT de las trazas con RTT positivo agrupadas según su intervalo de agrupación, un RTT negativo indica que la prueba se realizó pero no obtuvo respuesta del *host* o el paquete se perdió, por lo tanto

no se consideran para el cálculo de esta gráfica.

Ya teniendo estos datos es trivial desplegar el mapa de calor usando Highcharts, sencillamente se prepara un arreglo de tuplas de tipo <fecha, hora, RTT promedio> donde fecha es el día mes y año del grupo y hora es un punto flotante entre 0 y 24 que resulta de sumar la hora del grupo (0-23) al minuto (10-60) dividido entre 60 ($hora = hora\_grupo + minuto\_grupo/60$).

Figura 3.14: Mapa de Calor de RTT con granularidad de 10 minutos para el mes de noviembre 2015 (REDISE)

3.3.0.2 Cálculo de horas activas

En el marco de este trabajo llamaremos a una hora "activa" cuando se pudieron recoger al menos el 50% de las muestras esperadas durante dicha hora, análogamente una hora "inactiva" es aquella donde no se pudieron recoger el 50% de las muestras. Para determinar si una hora es activa se usa la siguiente fórmula:

$$T_{min} = 3600/interval * 0.5 \tag{3.1}$$

Donde *interval* es el tiempo entre pruebas en segundos y T_{min} es la cantidad mínima de muestras necesarias para considerar una hora activa.

$$activo = \begin{cases} Si, & T_{min} \leq T_{observadas} \\ No, & T_{min} > T_{observadas} \end{cases} \tag{3.2}$$

Y $T_{observadas}$ es la cantidad de trazas capturadas exitosamente durante una hora.

De forma similar al algoritmo de cálculo de mapas de calor, debemos agrupar las trazas por hora y contar el número de trazas por grupo. Afortunadamente podemos aprovechar la columna *time_60* para hacer este agrupamiento, luego debemos comparar con el umbral mínimo T_{min} para esa hora. Para determinar el intervalo entre pruebas que estuvo activo durante la prueba, podemos buscar los parámetros activos en cualquier momento usando el historial del plan de monitoreo, sin embargo, cada traza se guarda junto con el intervalo entre pruebas, de manera que podemos hacer una consulta mucho mas sencilla a la base de datos.

Sabiendo esto, la consulta a la base de datos queda así:

```
1 SELECT time_60, COUNT(*), AVG('interval')
2 FROM traceping
3 WHERE link_id=x AND time_60 >= s AND time_60 < f AND rtt>0
4 GROUP BY time_60
```

Donde:

- **x**: es el id del enlace
- **s**: es la fecha inicial
- **f**: es la fecha final

La consulta anterior retorna una marca de tiempo (*timestamp*) que representa la hora de agrupación, el número de muestras observadas ($T_{observadas}$), y el intervalo entre pruebas promedio. Calculamos el promedio ya que el usuario puede cambiar el intervalo entre pruebas en cualquier momento, de modo que una misma hora puede tener muestras tomadas con intervalos distintos. Calculando el promedio, obtendremos el mismo valor de tiempo entre pruebas para una hora en que el usuario no hizo cambios y un valor aproximado para horas en que hubo cambios.

El Algoritmo 4 totaliza el número de horas activas por hora del día, es decir que si se elige un período de n días el algoritmo retorna un arreglo de tamaño 24, donde cada posición del arreglo tendrá un entero entre 0 y n. El algoritmo recorre el arreglo retornado por la consulta SQL verificando la condición para que la hora sea activa, si es activa, suma uno al total para esa hora del día, luego es sencillo calcular el porcentaje de horas activas por hora del día dividiendo entre el número total de días.

Algoritmo 4 Calculo de horas activas

```
1:  procedimiento ACTIVE_HOURS(parameters)
2:      link ← GET(Link, id = parameters.link_id)        ▷ Retira el enlace por su id
3:      start_date ← parameters.start_date
4:      end_date ← parameters.end_date
5:      array ← get data from sql query        ▷ Ejecuta la consulta a la base de datos
6:      days ← (start_date − end_date).DAYS                    ▷ Número de dias total
7:      total_hours ← days ∗ 24
8:      hours ← array[24]                    ▷ Arreglo tamaño 24 inicializado en 0
9:      si array lenght > 0 entonces
10:         j ← 0
11:         para i en RANGE(0,total_hours) hacer
12:             current_hour ← start_date + i              ▷ Suma a la hora inicial i horas
13:             si array[j][0] = current_hour entonces
14:                 si j= length of  array entonces BREAK
15:                 si array[j][1] ⩾ 1800/array[j][2] entonces  ▷ Condicion para que la
    hora sea activa
16:                     hours[i%24] ← hours[i%24] + 1      ▷ le suma a las horas activas
    para esa hora
17:             j ← j + 1
        devolver hours
```

3.3.0.3 Cálculo de periodos de actividad continúa

Como se dijo en la Sección 2.3.6 llamaremos "actividad continua" a un evento durante el cual un servicio está continuamente disponible para un cliente, en este trabajo un

servicio está disponible en un instante de tiempo cuando se ha capturado exitosamente una muestra durante ese instante, en caso contrario se guarda un valor de RTT de "-1" indicando que el servicio no estaba disponible, las pruebas que usamos para determinar periodos de actividad continua son las de ping y httping (ver Sección 4.2.4) ya que ambas realizan un trabajo similar, pero usando protocolos distintos.

El algoritmo de cálculo de periodos de actividad continúa recorre las trazas para el intervalo de tiempo seleccionado extendiendo periodos de actividad cuando encuentra trazas **consecutivas** para las cuales el valor de RTT es positivo (hubo respuesta del TPD y por lo tanto el servicio está disponible) y de forma análoga extiende periodos de inactividad continua cuando encuentra trazas **consecutivas** para las cuales el valor de RTT es -1 (no hubo respuesta). La consulta a la base de datos es muy sencilla ya que solo hace falta consultar el RTT y marca de tiempo de las trazas ordenadas por tiempo:

```
1 SELECT time,rtt
2 FROM traceping
3 WHERE link_id=x AND time >= s AND time <= f
4 ORDER BY time
```

Donde:

- **x**: es el id del enlace
- **s**: es la fecha inicial
- **f**: es la fecha final

Tras ejecutar la consulta a la base de datos, se verifica que se tenga al menos una muestra, en caso contrario, a falta de información el período se considera totalmente inactivo. En caso contrario, se revisa la primera traza, si es positiva entonces se crea un período activo y se marca el inicio del período como el tiempo inicial, luego se pasa a revisar la siguiente traza hasta que se encuentre una traza no positiva y se marca como el tiempo final del período activo al tiempo de la traza negativa. Este proceso se repite análogamente para el período inactivo hasta llegar a la última traza. Cada vez que se determina el instante final de un período este se añade a la lista de periodos activos o inactivos según corresponda.

Todo el proceso se puede observar a detalle en el Algoritmo 5.

3.3.0.4 Cálculo de días activos

A diferencia de las horas activas, definimos a un día como "activo" cuando podemos observar un período de actividad continua de al menos una hora durante ese día. El algoritmo para determinar días activos es similar al de cálculo de periodos de actividad continua (Algoritmo 5), la diferencia radica en que primero se agrupan las trazas por día y luego se aplica el algoritmo de actividad continua, si se observa un período continuo de al menos una hora, el día se considera activo y se pasa al siguiente día.

Ya que el algoritmo de días activos es usado generalmente para dibujar un gráfico de barras indicando el porcentaje de días activos, también es necesario determinar la cantidad de días totales en el período seleccionado por el usuario. Luego cuando se tenga el total de días activos por día de la semana se puede dividir entre los días totales, un ejemplo del resultado deseado como salida del algoritmo de días activos puede ser visto en la Tabla 3.11

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
Activos	4	5	3	4	4	3	4
Porcentaje	100	100	60	100	100	75	100
Totales	4	5	5	4	4	4	4

Tabla 3.11: Tabla de días activos para el mes de septiembre del servidor de RESIDE

Para facilitar la comprensión del algoritmo este se va a separar en tres subrutinas, el Algoritmo 6 corresponde al cálculo de días totales en el período, el Algoritmo 7 corresponde a la separación de trazas por día y el Algoritmo 8 corresponde al computo de días activos.

Algoritmo 5 Calculo de periodos de actividad continua

```
1: procedimiento CONTINUOUS_ACTIVITY(parameters)
2:     link ← GET(Link, id = parameters.link_id)        ▷ Retira el enlace por su id
3:     start_date ← parameters.start_date
4:     end_date ← parameters.end_date
5:     traces ← get data from sql query        ▷ Ejecuta la consulta a la base de datos
6:     num_traces ← length of  traces
7:     active_periods ← new array
8:     inactive_periods ← new array
9:     si num_traces = 0 entonces
10:        append to inactive_periodos (start_date, end_date)
        devolver active_periods, inactive_periods
11:    period_start ← start_date
12:    si traces[0][1] > 0 entonces
13:        period_active ← True
14:    si no
15:        period_active ← False
16:    para i en RANGE(1,num_traces-1) hacer
17:        si period_active and traces[i][1] <0 entonces
18:            append to active_periods (duration, period_start, traces[i][0])
19:            period_active ← False
20:            period_start ← traces[i][0]
21:        si no if !period_active and traces[i][1] >0 then
22:            append to inactive_periods (duration, period_start, traces[i][0])
23:            period_active ← True
24:            period_start ← traces[i][0]
25:    si period_active entonces
26:        append to active_periods (duration, period_start, end_time)
27:    si no
28:        append to inactive_periods (duration, period_start, end_time)
        devolver active_periods, inactive_periods
```

Algoritmo 6 Calculo de días totales

```
1: procedimiento TOTAL_DAYS(start_date,end_date)
2:     number_of_days ← (end_date − start_date).DAYS
3:     total_days ← array[7]                    ▷ Arreglo de tamaño 7 inicializado en 0
4:     para i en RANGE(0,number_of_days) hacer
5:         current_day ← start_date + 1         ▷ Suma a la fecha inicial un día
6:         weekday ← current_day.WEEKDAY        ▷ Toma el día de la semana de la fecha
7:         total_days[weekday] ← total_days[weekday] + 1
       devolver total_days
```

Algoritmo 7 Separación de trazas por día

```
1: procedimiento SPLIT_TRACES(traces)
2:     days ← empty dictionary
3:     para trace en traces hacer
4:         ordinal ← trace[0].TOORDINAL    ▷ Obtiene el ordinal proléptico gregoriano
   de la fecha
5:         si ordinal en days entonces
6:             append trace todays[ordinal]
7:         si no
8:             days[ordinal] ← empty list
9:             append trace todays[ordinal]
       devolver days
```

Algoritmo 8 Cálculo de días activos

```
1:  procedimiento ACTIVE_DAYS(parameters)
2:      link ← GET(Link, id = parameters.link_id)
3:      start_date ← parameters.start_date
4:      end_date ← parameters.end_date
5:      total_days ← TOTAL_DAYS(start_date, end_date)
6:      active_days ← array[7]                    ▷ Arreglo tamaño 7 inicializado en 0
7:      traces ← get data from sql query          ▷ ejecuta la consulta a la base de datos
8:      days ← SPLIT_TRACES(traces)
9:      para key en days.KEYS hacer               ▷ itera el diccionario por claves
10:         active_period ← 0
11:         traces ← days[key]
12:         length ← length of traces
13:         para i en RANGE(1,length) hacer
14:             si traces[i][1]>0 entonces
15:                 delta_seconds ← traces[i][0] − traces[i − 1][0]
16:                 active_period ← active_period + delta_seconds
17:                 si active_period > 3600 entonces          ▷ El día es activo
18:                     weekday ← traces[i][0].WEEKDAY
19:                     active_days[weekday] ← active_days[weekday] + 1
20:                     break
21:             si no
22:                 active_period ← 0
        devolver total_days,active_days
```

3.4 Pruebas de Rendimiento

Ya que la aplicación web está pensada para atender a una cantidad arbitraria de usuarios, deseamos conocer el número máximo de usuarios concurrentes a los que se les puede dar servicio con el hardware disponible. Llevaremos a cabo pruebas de carga en las cuales se simulará un ráfaga de peticiones variable para determinar la cantidad de peticiones concurrentes que el servidor puede manejar antes de que ocurra una degradación de servicio o el servidor sea incapaz de responder a mas peticiones.

Consideramos que ocurre una degradación de servicio cuando el tiempo de espera del usuario aumenta mas allá de un umbral máximo, según Nielsen [36], existen tres umbrales importantes a la hora de evaluar el tiempo de respuesta de una aplicación:

1. **0.1 segundo:** Límite en el cual un usuario siente que está manipulando los objetos desde la interfaz de usuario.
2. **1 segundo:** Límite en el cual un usuario siente que está navegando libremente.
3. **10 segundos:** Límite en el cual se pierde la atención del usuario, es buena idea proveer barras de cargas y medios para cancelar la operación cuando se trata de operaciones de larga duración.

Para las siguientes pruebas consideraremos las siguientes métricas de rendimiento de la aplicación web:

- Tiempo de respuesta promedio
- Tiempo de respuesta mediana
- Tiempo de respuesta mínima
- Tiempo de respuesta máximo
- Porcentaje de error
- Rendimiento en términos de peticiones por segundo durante la duración de la prueba

- Rendimiento en términos de bytes por segundo durante la duración de la prueba

Consideraremos una prueba como exitosa cuando el porcentaje de error sea igual a 0 y el tiempo máximo de espera de una petición no supere los 10.000ms, tomando en cuenta los umbrales definidos por [36]. Se considerará optima aquella prueba que además de cumplir con ser exitosa, tenga el mayor rendimiento en términos de peticiones por segundo (Req/s) entre todas las pruebas realizadas.

La herramienta elegida para ejecutar las pruebas fue JMeter [37], esta herramienta permite simular casos de uso específicos de una aplicación web como ráfagas de peticiones que pueden ser configuradas para seguir cualquier patrón dado. En nuestro caso, Jmeter se usará para generar una ráfaga de peticiones con un intervalo constante entre cada petición durante un tiempo de un minuto. Jmeter provee un componente llamado *Summary Report* que totaliza y promedia los resultados obtenidos de todas las peticiones enviadas del que podemos extraer todas las métricas de interés.

3.4.1 Metodología de las pruebas

Para determinar la carga máxima que la aplicación web es capaz de soportar en términos de peticiones por minuto, simularemos una ráfaga de peticiones en un intervalo de tiempo de un minuto, variando el número de peticiones hasta obtener un valor óptimo. La metodología a seguir es la siguiente:

1. Se comenzará ejecutando una primera prueba con un número prudencialmente bajo de peticiones durante 60 segundos, este número debe preferiblemente ser múltiplo de dos.

2. Se varía el número de peticiones enviadas en la ráfaga multiplicando por dos cada vez que una prueba es exitosa, así si la primera prueba se hace con 16 peticiones, la siguiente tendrá 32, 64, 128 y así sucesivamente.

3. Tras la primera prueba fallida, se calculará el número de peticiones de la siguiente prueba (P_{sig}) como el punto medio entre la cantidad de peticiones de la prueba previa (P_{prev}) y la prueba actual (P_{act}).

$$P_{sig} = \begin{cases} P_{prev} + |P_{act} - P_{prev}|/2 & \text{Si la prueba fue exitosa} \\ P_{prev} - |P_{act} - P_{prev}|/2 & \text{Si la prueba fue fallida} \end{cases}$$

4. El paso 3. se repite hasta que no se obtenga mayor precisión con pruebas sucesivas

3.4.2 Condiciones de las pruebas

Todas las pruebas se ejecutaron sobre una red local entre dos máquinas conectadas a un mismo enrutador: una de ellas ejecuta el servidor web, y la otra ejecuta las pruebas simulando las peticiones y totalizando los resultados observados; la latencia de la red se considera menor a 5ms.

Las características del servidor web se pueden observar en la Tabla 3.12

Procesador	Memoria	Sistema Operativo
4x Intel i5-2500 CPU @3.30GHz	8 GB RAM	Ubuntu 15.04

Tabla 3.12: Especificaciones del servidor web

3.4.3 Resultados obtenidos

1. **Archivos estáticos con Apache[3]:** En esta prueba se configuró un servidor Apache para servir solo páginas web y archivos estáticos, como se puede observar en la Tabla 3.13, se ha elegido el valor de 640 peticiones por minuto como valor óptimo ya que tiene el mejor rendimiento entre las pruebas exitosas.

2. **Páginas dinámicas con Django sobre Apache:** En esta prueba se configuró Apache para ejecutar **solo** la aplicación web Django a través del módulo *mod_wsgi* y no se solicitaron los archivos estáticos vinculados a las páginas visitadas. En la tabla 3.14 se observa un valor óptimo de 768 peticiones por minuto, para el cual la media y la mediana se mantienen por debajo del umbral de los 0.1 segundos.

3. **Páginas dinámicas con Django y archivos estáticos sobre el mismo servidor Apache**: en esta prueba se configuró el servidor Apache tal como

[3]**Apache: Servidor Web HTTP https://httpd.apache.org/**

# Req	Avg (ms)	Med (ms)	Min (ms)	Max (ms)	Error %	Req/s	kb/sec
128	42	37	28	135	0.0 %	2.1	24.3
256	44	38	26	273	0.0%	4.3	48.4
512	38	37	24	153	0.0%	8.5	96.5
1024	49468	57000	43	189852	30.37%	5.3	46.3
768	29929	23248	45	172688	9.24%	4.1	43.4
640	73	55	27	289	0.00%	10.6	120.6
704	8339	458	28	64813	0.00%	6.6	74.9
672	1125	217	45	18280	0.00%	9.6	108.9
656	206	120	28	4194	0.00%	10.4	117.6

Tabla 3.13: Peticiones de archivos estáticos con apache

está descrito en la Sección 3.1.3, es decir que Apache servirá directamente las peticiones de archivos estáticos y pasará a la aplicación Django las peticiones dinámicas. En la tabla 3.15 se observa un valor óptimo de 32 peticiones por minuto con una media y mediana por encima del umbral del segundo.

3.4.4 Análisis de los resultados

Como se pudo observar en las pruebas anteriores y corroborando lo dicho en la Sección 3.1.3, de tenerse los recursos suficientes, Django debería ejecutarse preferiblemente en una máquina separada a la encarga de servir archivos estáticos. Se ha observado que un servidor de las características descritas puede manejar una carga de 768 peticiones dinámicas por minuto, y solo 640 peticiones estáticas, lo cual representa una diferencia de mas del 16%.

Independiente de la cantidad de peticiones que el servidor pueda manejar en condiciones ideales, en la realidad cada visita única, es decir, cada usuario que visite por primera vez el sitio web deberá descargar todos los archivos estáticos junto con la página HTML generada dinámicamente, como se observó en la tabla 3.15, el servidor es capaz de atender unas 32 peticiones de este tipo por minuto, lo cual representa una marcada diferencia del 95% en comparación con la prueba sin archivos estáticos, por lo tanto, los archivos estáticos representan un cuello de botella importante para la

# Req	Avg (ms)	Med (ms)	Min (ms)	Max (ms)	Error %	Req/s	kb/sec
128	37	33	23	140	0.00%	2.1	9.8
256	35	33	24	237	0.00%	4.3	19.5
512	47	33	23	1038	0.00%	8.5	38.9
1024	13709	575	27	97869	3.61%	7.9	35.3
768	39	33	23	1034	0.00%	12.8	58.2
896	12483	2384	24	101648	1.56%	7.1	31.9
832	390	35	24	9630	0.00%	12.7	57.7
672	1125	217	45	18280	0.00%	9.6	108.9
656	206	120	28	4194	0.00%	10.4	117.6

Tabla 3.14: Peticiones de páginas dinámicas con Django

# Req	Avg (ms)	Med (ms)	Min (ms)	Max (ms)	Error %	Req/s	kb/sec
64	77018	74557	2558	321297	9.38%	11.5	66
32	3438	3177	2484	5432	0.00%	31.1	179.3
48	41056	40148	8178	100190	0.00%	25.2	144.9
40	12023	8176	2121	36000	0.00%	32.6	188
36	10152	8335	2091	41631	0.00%	23.1	133
34	17259	14613	4165	42601	0.00%	24.6	141.9

Tabla 3.15: Peticiones de páginas dinámicas con sus archivos estáticos.

aplicación.

A pesar de los anterior, en peticiones sucesivas el navegador web usará los archivos en el caché de modo que en la práctica sería posible atender muchas mas peticiones por minuto.

Desde el punto de vista de la velocidad de respuesta del servicio ofrecido, como se ve en la tabla 3.14 tanto la velocidad de respuesta promedio como la mediana se mantienen muy por debajo del umbral de los 0.1 segundos, e incluso el tiempo de respuesta máximo se mantiene cerca del umbral del segundo, dejando de lado la latencia de la red, esto conforma una experiencia de navegación rápida.

Capítulo 4

Tentáculos

En el marco de este trabajo hemos definido una analogía sencilla que compara un enlace en una red a un tentáculo de un pulpo, a cada extremo del tentáculo se encuentran los agentes relevantes al sistema de monitoreo: monitores de red que hemos llamado Tentacle Probe Source ya que estos generan sondas y funciones como los puntos activos del monitoreo y los Tentacle Probe Destination que son los nodos de interés a monitorear. Cada tentáculo está compuesto además por una cantidad variable de nodos intermedios como enrutadores o *proxies* que establecen la comunicación entre el TPS y cada TPD.

4.1 Diseño del Monitor

El diseño del monitor esta sujeto a los siguientes baremos:

- **Estabilidad:** ya que el monitor podría dejarse desatendido es esencial que sea estable, si el monitor no se está ejecutando el usuario no obtendrá los resultados esperados en la aplicación web.
- **Flexibilidad:** el monitor debe ser capaz de cargar nuevas pruebas en tiempo de ejecución, esto es especialmente importante ya que no desea interrumpir otras pruebas que se estén ejecutando.

- **Planificación precisa:** es importante que las pruebas se ejecuten en el momento adecuado, siguiendo de manera fiel el plan de monitoreo.
- **Ejecutable en equipos de bajo costo:** el monitor debe incluir el código mínimo para su funcionamiento, tener un uso eficiente de memoria y evitar desbordar la memoria secundaria del *host*.
- **Configuración remota:** el monitor debe implementar algún protocolo para actualizar su plan de monitoreo a partir de cambios hechos en la aplicación web.
- **Bitácora:** ya que el monitor se ejecuta como un proceso daemon, se desea tener una bitácora donde se puedan leer mensajes sobre los eventos relevantes durante la ejecución.

4.2 Tentacle Probe Source

Tentacle Probe Source (TPS) es un monitor de red ligero, diseñado para ajustarse a las limitaciones de sistemas de bajo costo y con la intensión de ser desplegado en las redes que se desea monitorear y ser dejado desatendido durante periodos arbitrarios de tiempo. Su misión es ejecutar pruebas periódicas siguiendo el plan de monitoreo definido por el usuario con el objetivo de recoger datos sobre las métricas relevantes a la red a monitorear, los resultados de las pruebas son guardados temporalmente en la memoria del dispositivo y se suben a la nube regularmente. La frecuencia en que se suben los datos a la nube depende de la cantidad de memoria disponible en el dispositivo, la cantidad de datos generados por el monitoreo, y el ancho de banda disponible.

Uno de los objetivos principales del monitor es ejecutar las pruebas en el momento preciso dado por el plan de monitoreo, ya que el posterior análisis de los datos por parte de Octopus Head exige que las muestras se tomen con un patrón regular y conocido, para esto, un planificador que asegura la ejecución precisa de las pruebas es central en la arquitectura del monitor. El comportamiento del monitor viene dado por un plan de monitoreo, este es definido por el usuario en Octopus Head, toda la comunicación entre los monitores Tentacle Probe Source y Octopus Head ocurre a través de la nube,

Octopus Head transfiere el plan de monitoreo a través de mensajes individuales que especifican cambios tales como re-planificación de pruebas, cambios a los parámetros de ejecución, e incluir nuevos enlaces a monitorear.

4.2.1 Arquitectura

Como se puede observar en la figura 4.1, Tentacle Probe Source tiene una arquitectura basada en la biblioteca APScheduler(ver 2.6.13), esta posee componentes altamente desacoplados con responsabilidades bien delimitadas para facilitar el desarrollo de aplicaciones que requieran controlar finamente el momento de ejecución de ciertas rutinas. Cada uno de los componentes de APScheduler pueden ser reemplazados fácilmente ya que comparten una interfaz común.

4.2.2 Diagrama de actividades

La ejecución del monitor comienza en el cliente, el cliente es el encargado de instanciar el proceso daemon, a este proceso lo llamaremos daemonización y se explicará en la Seccion 4.2.3.1; para asegurarse de no ejecutar múltiples instancias del monitor TPS se revisa un archivo `.pid`, si el archivo existiera, significa que el monitor ya se está ejecutando y el cliente informaría del error al usuario.

Después de que el proceso está daemonizado, comienza la fase de inicialización. Para esto se crea el planificador y se cargan las tareas del almacén de tareas, importando el código de las pruebas a ejecutar. A este punto las pruebas están planificadas tentativamente y cargadas en memoria pero solo serán ejecutadas después de que se inicie el planificador.

Al iniciar el planificador, éste pasa a ejecutarse como un subproceso y se encarga de iniciar las tareas en el momento preciso. El planificador no ejecuta las tareas sino que se las envía al ejecutor. Cuando una tarea se ha completado, el ejecutor envía un mensaje al planificador, de esta manera el planificador puede conocer el estado de las tareas y evitar que una misma tarea ocupe mas de un hilo de ejecución en caso de que esté tardando mas de lo normal o esté bloqueada.

Mientras tanto, el hilo principal se conecta a la nube usando el API de Dropbox que

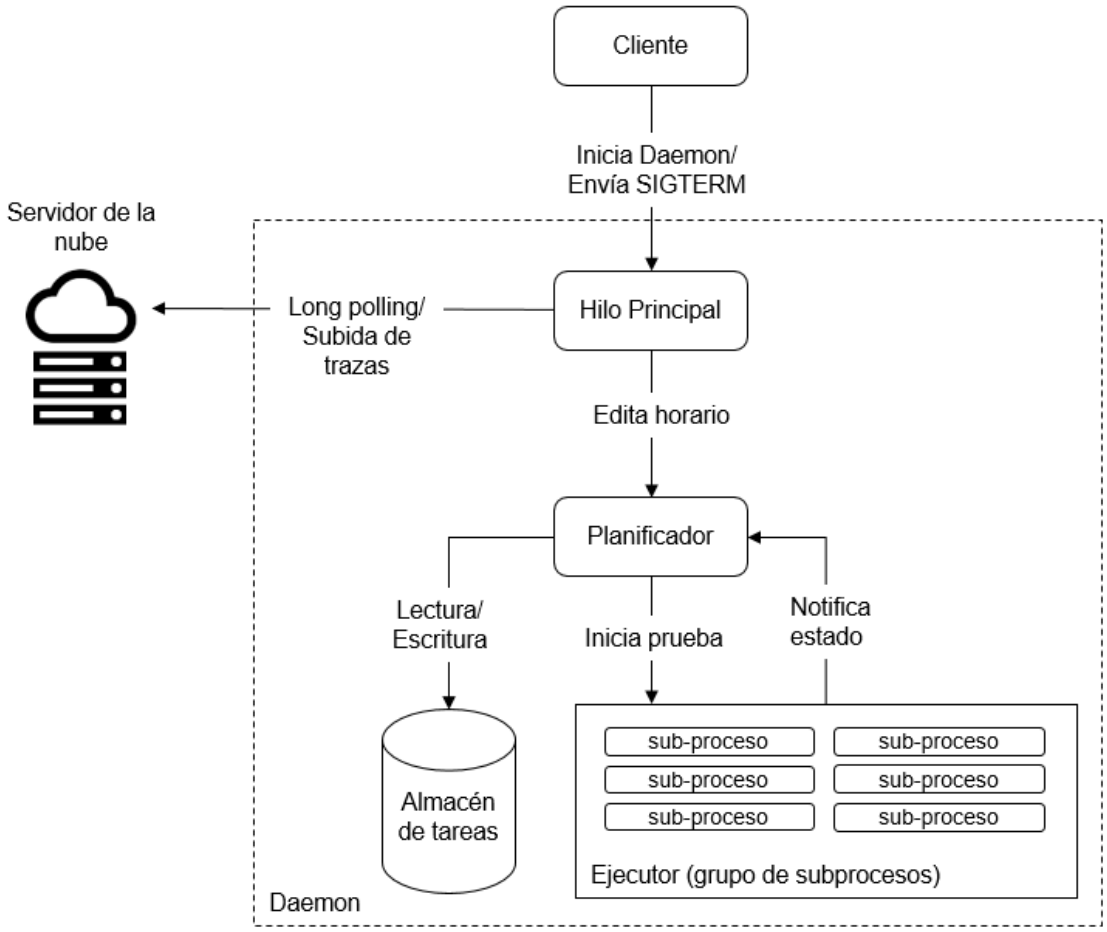

Figura 4.1: Diagrama de Componentes del TPS

posee un método para notificar a los clientes sobre cambios a una carpeta en tiempo real. El método llamado *Longpoll Delta* consiste en abrir una conexión HTTP con un alto valor de *timeout* (entre 30 y 120 segundos), si ocurre un cambio en la carpeta, el servidor de Dropbox responde de inmediato indicando que ocurrieron cambios y el monitor procederá a manejar este evento (descargando los archivos nuevos y aplicando los cambios al plan de monitoreo). En caso contrario, una vez expirado el temporizador, Dropbox responde indicando que no ocurrieron cambios y el monitor repite el proceso.

Los eventos relevantes al monitor son los siguientes: (1) una prueba se ha agregado al plan de monitoreo y debe cargarse a memoria y planificarse, (2) una prueba se ha eliminado y debe ser eliminada del plan de monitoreo, (3) el intervalo entre pruebas ha

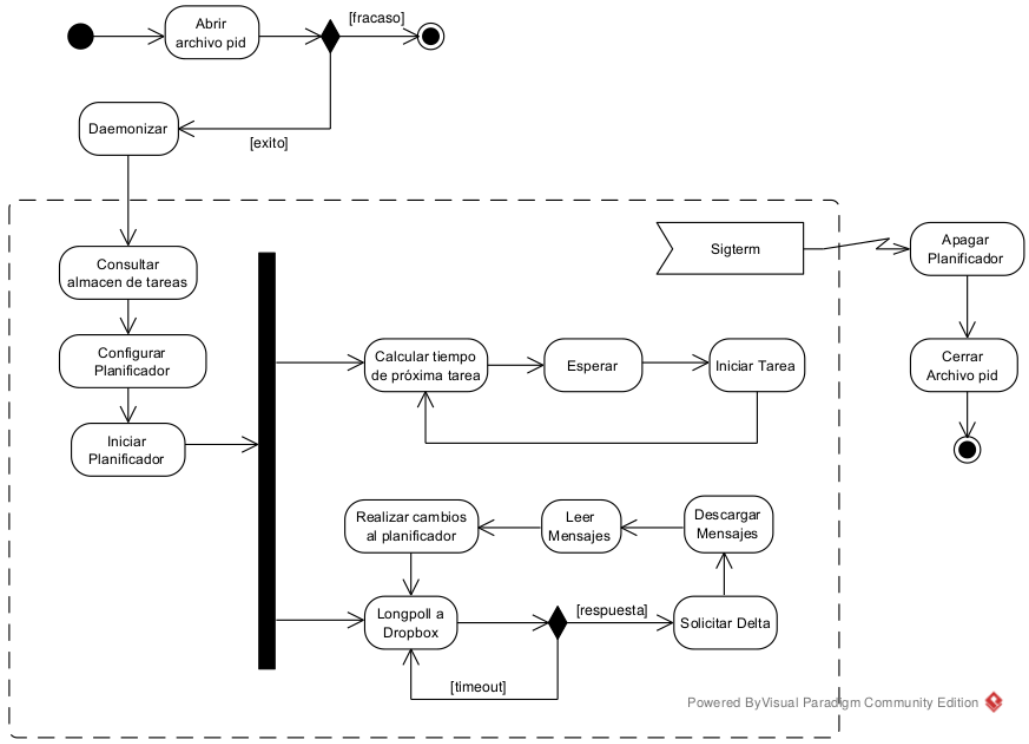

Figura 4.2: Diagrama de actividades de Tentacle

cambiado y debe replanificarse la próxima ejecución, (4) los parámetros de una prueba han cambiado, (5) se ha agregado un nuevo enlace a monitorear, (6) se ha eliminado o desactivado un enlace monitoreado.

El monitor solo puede detenerse a través de una señal del sistema operativo, el cliente utiliza el archivo `pid` para determinar el identificador del proceso y enviar la señal. El manejador de excepciones del monitor entonces inicia una secuencia de apagado, deteniendo las pruebas, apagando el planificador y desbloqueado el archivo `.pid`.

Esta secuencia de actividades puede verse en la figura 4.2.

4.2.3 Componentes

En esta sección se explican a detalle los componentes que forman parte de la arquitectura del monitor.

4.2.3.1 Cliente

El cliente de TPS es el programa encargado de iniciar o detener la ejecución del monitor en modo daemon y funge como interfaz entre el usuario y monitor.

La tarea principal del cliente consiste en daemonizar el monitor, Según Finney [38], el proceso para formar un daemon con comportamiento correcto son los siguientes :

1. Disociar el programa del terminal
2. Convertirse en un líder de sesión
3. Convertirse en un líder de grupo de procesos
4. Ejecutarse como una tarea de fondo haciendo *fork* y luego dejando que el programa "padre" finalice inmediatamente, de este modo el proceso "hijo" queda huérfano, este proceso se repite una vez mas para convertir el programa en un líder de sesión.
5. Establecer el directorio raíz (/) como el directorio de trabajo (*working directory*) del programa
6. Cerrar todos los descriptores de archivos
7. Redirigir los flujos estándar (stdout, stderr, stdin) a un archivo de bitácora, o a /dev/null

La Tabla 4.1 muestra los comandos que pueden ser usados al invocar el cliente; los comandos start y restart se pueden invocar con la opción –no-daemon para iniciar el monitor en modo consola (sin daemonizar), de esta manera se puede ver la salida de la bitácora del monitor por consola, esto puede ayudar al desarrollador o al usuario a encontrar y solucionar problemas.

Ejemplo:

```
$ python tentacle.py start --no-daemon
```

Comando	Descripción
start	Inicia el monitor como un proceso daemon, falla si el archivo `.pid` ya existe (el monitor se esta ejecutando)
stop	Detiene el monitor enviando la señal SIGTERM al proceso daemon, falla si el archivo `.pid` no existe (el monitor no se esta ejecutando)
restart	Detiene e inicia el monitor
jobs	Muestra las tareas pendientes en el planificador (plan de monitoreo)

Tabla 4.1: Comandos del cliente TPS

4.2.3.2 Hilo Principal

El hilo principal de ejecución es el punto de partida desde el momento en que el proceso ya se ha convertido en un daemon, se encarga de inicializar el planificador el cual a su vez pasa a ejecutarse en segundo plano, luego, su tarea consiste en mantener el plan de monitoreo al día a partir de los cambios que se hagan en Octopus Head.

El hilo principal consiste en un bucle de ejecución que ejecuta el siguiente algoritmo:

1. Llama al método *Delta* de Dropbox sobre la carpeta que hace las veces de buzón de mensajes del monitor sin usar un cursor (ver Sección 3.2.3), este llamado retorna como lista de entradas delta todos los archivos que hay en la carpeta, de esta manera, el monitor puede leer los mensajes que hayan sido dejados en la carpeta mientras no se estaba ejecutando. Luego guarda el cursor retornado por el método delta para ser usado en llamadas sucesivas.

2. Toma la lista de entradas delta obtenida de cada llamada a *Delta* y descarga archivo por archivo, leyendo su contenido, a partir del nombre y contenido del mensaje pueden ocurrir los siguientes escenarios:

   - Se ignora y elimina el mensaje en los siguientes casos:
     - Si el nombre del archivo no está en el formato esperado.
     - Si el contenido del archivo no puede ser interpretado.
     - Si el contenido del archivo no contiene la información esperada; el contenido esperado de los mensajes se puede ver en las tablas 3.8 y

3.9.

- Si se trata de un mensaje cuyo nombre comience con *link* se lee el identificador del enlace y se agrega, activa o desactiva el enlace según el valor de *status* del mensaje.
- Si se trata de un mensaje cuyo nombre comience con *test* se lee el identificador de la prueba. Si una prueba con ese identificador ya existe entonces se re-planifica o se actualizan sus parámetros. En caso de que no exista la prueba se inserta al planificador. En caso de que el valor de *status* sea falso, entonces la prueba debe ser eliminada de la planificación.

3. Realiza peticiones al API de notificaciones de Dropbox usando el método *Longpoll Delta*. Si la conexión con el servidor de Dropbox es estable, este método emula una comunicación directa con el servidor de Dropbox y permite que el monitor detecte los cambios en el buzón a baja latencia..

4. Si no ocurrieron cambios en la carpeta de Dropbox antes del *timeout* se repite el paso 3, en caso contrario se pasa a repetir el paso 1 para así determinar cuales fueron los cambios al buzón y leer los potenciales mensajes.

El diagrama de secuencia del hilo principal del monitor puede ser observado en la Figura 4.3.

4.2.3.3 Planificador

Para la implementación del planificador se hizo uso de APScheduler, una biblioteca que permite retrasar la ejecución de código python a instantes específicos en el futuro; el planificador se puede ejecutar como un subproceso de modo que otro hilo puede encargarse de mantener el estado de planificador actualizado, indicando cuando se deban agregar, re-planificar o eliminar pruebas.

El planificador está compuesto de gatillos para determinar el tiempo de ejecución de las pruebas, ejecutores, y almacenes de tareas que guardan el estado del planificador, todos estos componentes son configurables y se pueden extender o reusar para obtener cualquier comportamiento deseado. A continuación se explica su funcionamiento y como se usaron en el marco de este trabajo.

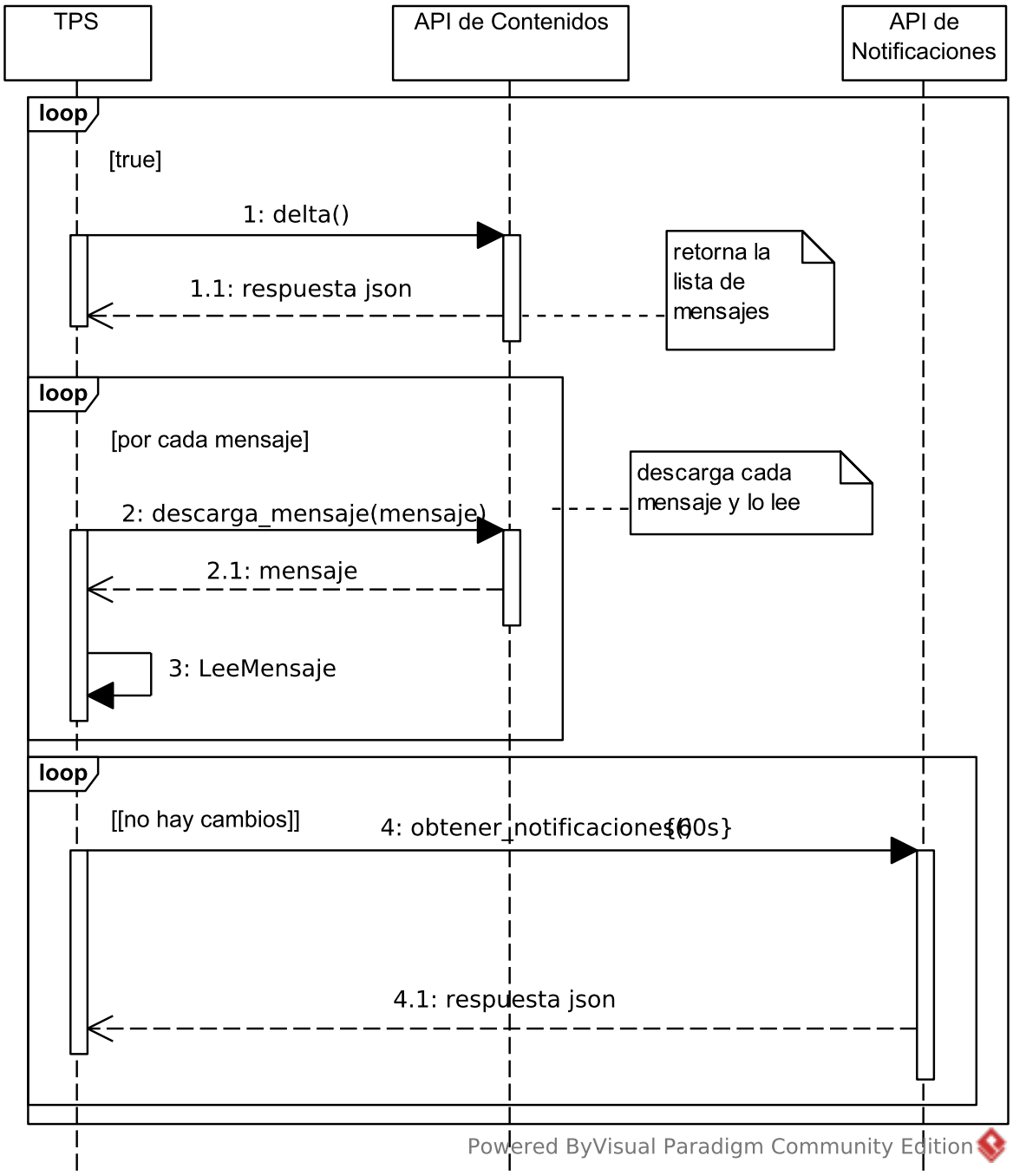

Figura 4.3: Diagrama de secuencia del hilo principal del Monitor

- **Gatillos**

  Los gatillos contienen la lógica para determinar en que momento se debe ejecutar una tarea, la biblioteca APScheduler incluye varios gatillos predefinidos de los cuales dos se han usado para el desarrollo de esta aplicación:

  - **Gatillo por intervalos:** ejecuta pruebas en intervalos regulares, opcionalmente se pueden suministrar fechas finales e iniciales de modo que las pruebas solo se ejecuten durante un período especifico, después de la fecha final el planificador elimina la prueba automáticamente.
  - **Gatillo por fecha:** ejecuta la prueba en una fecha especifica dada, una sola vez, útil para ejecutar pruebas que usen muchos recursos de red como *benchmarks* o capturas de tráfico de la red.

- **Ejecutores**

  El ejecutor es el ente encargado de llevar a cabo la ejecución de las tareas, en nuestro caso se ha hecho uso de un grupo de subprocesos (*thread pool*), la cantidad de hilos en el grupo esta dado por el número de pruebas que estaremos ejecutando de modo que siempre se tenga al menos un hilo disponible cuando se inicia una prueba.

  El ejecutor comunica el estado de ejecución de las tareas al planificador, de modo que se pueden implementar limites de concurrencia de forma tal que una tarea solo pueda ser ejecutada simultáneamente un cierto número de veces. En este trabajo solo puede ser ejecutada una prueba del mismo tipo a la vez, en otras palabras la concurrencia máxima es uno.

- **Almacén de tareas**

  El almacén de tareas guarda las tareas planificadas, el comportamiento por omisión es guardar las tareas en memoria, pero existen distintos tipos de almacenes como Redis [26] y bases de datos. Hemos usado la clase SQLAlchemyJobStore que permite guardar tareas en una base de datos ligera

como Sqlite[1] y así asegurar la persistencia de los datos de la aplicación en caso de interrupciones o reinicios.

El almacén de tareas debe guardar toda la información necesaria para recrear las tareas, en otras palabras debe guardar el gatillo, la ruta del ejecutable de la prueba, el arreglo de parámetros y el tiempo de la última ejecución. La estructura de la base de datos es muy sencilla: se guarda el identificador de la tarea como clave, el tiempo de la última ejecución (*timestamp*) y los datos de la ruta del ejecutable y los parámetros se serializan y se guardan en un campo BLOB.

4.2.3.4 Almacenamiento Compartido

El almacenamiento compartido se encarga de alojar los resultados de las pruebas en archivos de trazas, por cada prueba que se esté ejecutando se crea una carpeta donde se alojan sus respectivos archivos.

El costo de alojar archivos en la nube no depende del número de archivos sino del espacio total de disco en uso, sin embargo, Dropbox impone un limite de peticiones diarias por usuario, por lo que debemos intentar minimizar la cantidad de peticiones que realizamos. Para reducir el número de archivos en el almacenamiento compartido se crea para cada enlace un archivo por hora y todas las trazas que se generen en ese período se anexan al archivo correspondiente.

4.2.4 Pruebas Implementadas

Gracias a la arquitectura del TPS, implementar una prueba es tan sencillo como crear un paquete e implementar la función `run` en el archivo `init.py`, todas las pruebas implementadas hasta ahora comparten una flujo de ejecución similar:

1. Se leen los argumentos de la función, en caso de que el argumento de la función sea un diccionario se valida que tenga las claves y valores esperados.

2. Se obtienen los enlaces a monitorear y los parámetros globales leyendo el archivo de configuración del monitor

[1]Sqlite: https://www.sqlite.org/

3. Se ejecuta un comando externo como ping o traceroute o se usa código Python para evaluar alguna métrica de cada enlace monitoreado. Esto puede hacerse en paralelo en caso de que el tiempo sea crítico o en secuencia en caso de tratarse de una operación costosa en términos de recursos de la red o procesamiento.

4. (Opcional) si se ejecuta un comando externo se hace un *parsing* para extraer los resultados relevantes de la salida del programa y convertirlos a tipos de datos Python.

5. Se guardan los resultados en un archivo de trazas; generalmente el resultado de una prueba está representado por una linea en el archivo de trazas, sin embargo, el desarrollador tiene libertad de elegir el formato que desee para generar archivos de trazas.

Durante el desarrollo de este trabajo se han implementaron *wrappers* para los programas ping, traceroute e iperf y se desarrolló una prueba para determinar el estado de servidores a través del protocolo HTTP.

4.2.4.1 Ping

Esta prueba hace uso del comando ping para obtener datos de la latencia en un enlace, como se menciono en la sección 2.3.7, ping viene incluido en todas las distribuciones de linux por lo que no es necesario instalar ninguna dependencia o programa externo. La prueba consiste en instanciar un hilo por cada enlace monitoreado y ejecutar el comando ping. La prueba se realiza en paralelo para asegurar que la diferencia entre el tiempo de inicio esperado de la prueba y su tiempo ejecución efectivo sea mínimo.

El comando se ejecuta con la opción -D para que ping imprima cada resultado de latencia con una marca de tiempo entre corchetes. La cantidad de sondas ICMP, el tiempo entre sondas y el *timeout* se especifican con las opciones -c, -i y -W respectivamente. A continuación se puede ver un ejemplo de una llamada al comando ping:

```
$ping 150.185.138.59 -D -c 10 -i 1 -W 2
```

Finalmente, se toma la salida del comando ping y se extraen los datos que se guardarán en el archivo de trazas, como se puede ver en la figura 4.4, cada linea que comienza con un *timestamp* entre corchetes representa el resultado obtenido para una sonda, luego es sencillo obtener el valor del RTT realizando operaciones sobre cadenas de caracteres, esto puede verse en el siguiente segmento de código:

```
1 rtt = []
2 for line in lines:
3         if line.startswith('['):
4                 tokens = line.split()
5                 if "time=" in token:
6                         rtt.append(token.replace("time=",""))
7 rtt = median(rtt)
```

Independientemente del número de sondas que se envíen elegiremos solo la mediana como valor de la latencia para esa prueba. Si para una prueba no se obtiene ninguna respuesta entonces guardamos una traza con RTT=-1 indicando que el enlace está inactivo o el nodo está rechazando las sondas.

```
jesus@jesus-pc:~$ ping 150.185.138.59 -c 10 -i 1 -D
PING 150.185.138.59 (150.185.138.59) 56(84) bytes of data.
[1443758089.876135] 64 bytes from 150.185.138.59: icmp_seq=1 ttl=47 time=8485 ms
[1443758090.547867] 64 bytes from 150.185.138.59: icmp_seq=2 ttl=47 time=8148 ms
[1443758091.448875] 64 bytes from 150.185.138.59: icmp_seq=3 ttl=47 time=8041 ms
[1443758092.293731] 64 bytes from 150.185.138.59: icmp_seq=4 ttl=47 time=7878 ms
[1443758093.116296] 64 bytes from 150.185.138.59: icmp_seq=5 ttl=47 time=7693 ms
[1443758093.928141] 64 bytes from 150.185.138.59: icmp_seq=6 ttl=47 time=7497 ms
[1443758094.578914] 64 bytes from 150.185.138.59: icmp_seq=7 ttl=47 time=7140 ms
[1443758095.072853] 64 bytes from 150.185.138.59: icmp_seq=8 ttl=47 time=6625 ms
[1443758095.625384] 64 bytes from 150.185.138.59: icmp_seq=9 ttl=47 time=6170 ms
[1443758095.701993] 64 bytes from 150.185.138.59: icmp_seq=10 ttl=47 time=5246 m
s

--- 150.185.138.59 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9065ms
rtt min/avg/max/mdev = 5246.154/7292.862/8485.394/958.056 ms, pipe 9
```

Figura 4.4: Salida del comando Ping

4.2.4.2 Httping

Esta prueba usa el protocolo HTTP para hacer un *HEAD Request* y obtener el tiempo de respuesta y código de estatus HTTP. Esta prueba no es un *wrapper* de un comando

Parámetro	Tipo	Descripción
Numero de sondas	Entero	Número de sondas ICMP a enviar.
Timeout	Float	Tiempo en segundos a esperar por una respuesta antes de asumir una sonda como perdida.
Intervalo entre sondas	Float	Tiempo en segundos entre el envío de cada sonda individual.

Tabla 4.2: Parámetros de la prueba ping

externo, sino que llama directamente a una función para hacer la petición HTTP. Para facilitar el manejo de peticiones HTTP se usó la biblioteca Requests[2].

Esta prueba recibe como entrada un diccionario de parámetros, por lo tanto, debe validar que el diccionario tenga los valores esperados, existen dos tipos de parámetros: globales, que se aplican a todos los enlaces y los que son específicos a un enlace. Los parámetros específicos se identifican como el nombre base del parámetro seguido del identificador separado por el carácter "_". Por ejemplo, el puerto para el enlace 1 se inserta en el diccionario de parámetros con la clave "port_1". A continuación se puede observar la construcción de una petición para un enlace:

```
1  url = "http://" + link["ip"]
2  port_name = "port_%s" % link["id"]
3  path_name = "path_%s" % link["id"]
4
5  if port_name in kwargs:
6          url += ":" + str(kwargs[port_name])
7  if path_name in kwargs:
8          if kwargs[path_name]: #valida que la variable no esté vacia.
9                  url += kwargs[path_name]
10 response = requests.head(url,timeout=timeout)
```

La tabla 4.2.4.2 muestra los parámetros de la prueba httping.

Esta prueba es útil para monitorear todo tipo de servidores web y posee la ventaja sobre ping de utilizar un protocolo de capa aplicación por lo que da una idea mas

[2]Requests: HTTP para Humanos http://docs.python-requests.org/es/latest/

Parámetro	Tipo	Descripción
Timeout	Float (Global)	Tiempo de espera máximo por una respuesta
Path	String (Específico)	Cadena de caracteres que se concatena al IP o nombre de dominio del enlace, especialmente útil para probar servicios específicos de una aplicación o servicio web.
Port (Específico)	Entero	Espeficia el puerto al que se envía la peticion HTTP.

Tabla 4.3: Parametros de la prueba Httping

precisa de la experiencia del usuario al visitar dicho servicio; a diferencia de ping, el resultado obtenido no solo es la latencia de la red, sino la suma de la latencia de la red y el tiempo de respuesta del servidor, que puede estar sujeto, por ejemplo, al nivel de carga que esté manejando dicho servicio, o a la petición especifica que se esté realizando.

4.2.4.3 Traceroute

Esta prueba ejecuta el comando traceroute para obtener la ruta entre un par de nodos a través de una red IP, llamamos ruta a una secuencia de saltos (*hops*) que hace un paquete al atravesar un enrutador.

El comando traceroute imprime la ruta como una lista ordenada donde cada linea representa un salto, con su dirección IP, nombre de dominio y latencia, dependiendo del número de sondas que se estén enviando por salto pueden existir casos en que se obtenga respuesta de mas de una dirección IP. Este comportamiento se puede ver en la figura 4.5, en el salto 9 se observa que obtenemos una respuesta de la dirección IP 154.54.31.230 y dos respuestas de la 154.54.47.154.

A partir de la salida de traceroute se debe obtener una estructura de datos que facilite el análisis de la ruta, para esto se usó el modulo traceroute parser.py[3]. Este

[3]traceroute parser.py esta disponible a través del proyecto Netalyzr: http://netalyzr.icsi.berkeley.edu

```
jesus@jesus-pc:~$ traceroute 150.185.138.59
traceroute to 150.185.138.59 (150.185.138.59), 30 hops max, 60 byte packets
 1  192.168.1.1 (192.168.1.1)  0.296 ms  0.399 ms  0.504 ms
 2  186.14.23.1 (186.14.23.1)  463.401 ms  463.450 ms  464.369 ms
 3  200.82.134.67 (200.82.134.67)  460.462 ms  461.378 ms  462.339 ms
 4  10.1.232.145 (10.1.232.145)  492.168 ms  493.138 ms  495.226 ms
 5  10.1.230.98 (10.1.230.98)  582.430 ms *  583.399 ms
 6  ro-ccs-bdr-02-TenGig4-1-0.ln.inter.com.ve (10.1.230.62)  496.203 ms  458.475 ms  503.555 ms
 7  tengigabitethernet4-1.asr1.ccs1.gblx.net (64.215.248.93)  505.796 ms  506.888 ms  507.880 ms
 8  te0-0-0-34.ccr21.mia03.atlas.cogentco.com (154.54.12.69)  593.042 ms  591.014 ms  592.035 ms
 9  te4-1.mag01.mia03.atlas.cogentco.com (154.54.31.230)  590.031 ms te7-1.mag01.mia03.atlas.cogentco.com (154.54.47.154)  596.323 ms  593.998 ms
10  * 38.104.95.186 (38.104.95.186)  594.960 ms  570.787 ms
11  pa-us.redclara.net (200.0.204.6)  634.901 ms  637.250 ms *
12  reacciun-pa.redclara.net (200.0.204.150)  761.604 ms  758.751 ms  756.794 ms
13  150.185.255.86 (150.185.255.86)  680.235 ms  677.786 ms  680.704 ms
14  190.168.0.5 (190.168.0.5)  709.765 ms  709.753 ms  709.954 ms
15  150.185.163.248 (150.185.163.248)  710.052 ms  687.120 ms *
16  * 150.185.138.59 (150.185.138.59)  654.827 ms  654.779 ms
```

Figura 4.5: Salida del comando traceroute

módulo crea un archivo JSON extrayendo la información de la cabecera (IP destino y nombre de dominio), y crea una una lista de los saltos. Cada salto es a su vez una lista de sondas, cada sonda tiene dirección IP, nombre de dominio, RTT y anotaciones. Este archivo JSON luego puede ser leído por Octopus Head para iterar fácilmente sobre los saltos y leer la información de cada sonda específica.

El formato provisto por Netalyzer está compuesto de la siguiente manera:

```
[
        [
                {
                        "anno": anno_1,
                        "rtt": rtt_1,
                        "ipaddr": ipaddr_1,
                        "name": name_1
                },
                {
                        "anno":  anno_2,
                        "rtt": rtt_2,
                        "ipaddr": ipaddr_2,
                        "name": name_2
                },
                ...
        ],
        ...
]
```

Cada vez que se realiza una prueba con traceroute se anexa al archivo de trazas correspondiente una entrada con la marca de tiempo de inicio de la prueba, un carácter de separación y luego una cadena en el formato JSON antes mostrado.

Las parámetros de esta prueba son los siguientes:

Parámetro	Tipo	Descripción
Numero de sondas	Entero	Número de sondas a enviar por cada valor de TTL.
TTL Máximo	Entero	Numero máximo de saltos antes de asumir que el nodo no es alcanzable.

Tabla 4.4: Parámetros de la prueba con traceroute

4.2.4.4 Throughput con Iperf

La prueba de *throughput* con Iperf hace mediciones de ancho de banda entre un par de nodos usando el programa Iperf. Ya que Iperf es un programa de tipo cliente-servidor, el usuario que realiza el monitoreo debe asegurarse de mantener una instancia de Iperf en modo servidor en el TPD, en otras palabras es un requisito para esta prueba que el usuario tenga acceso o cuente con la colaboración explicita de los TPD.

El comando Iperf se puede ejecutar en modo servidor lo cual le permite conectarse a un puerto y esperar conexiones para ejecutar pruebas de *throughput*, el comando para iniciar Iperf en el TPD en modo daemon es el siguiente:

```
$iperf -s -D
```

A diferencia de las pruebas como las de ping o httping que solo envían unas pocas sondas ICMP o peticiones *HTTP HEAD*, Iperf utiliza una gran cantidad de recursos de red intentando inundar el enlace y obtener el mayor *throughput* posible, por lo tanto, el usuario debe asegurarse de no interferir constantemente el rendimiento de la red planificando demasiadas ejecuciones de esta prueba. Aunado a esto, cada ejecución del comando Iperf se realiza secuencialmente, de esta manera, se evita que los distintos enlaces monitoreados tengan que compartir los recursos de red y se generen cuellos de botella en el TPS.

Además de las consideraciones de recursos de red consumidos, cada prueba con Iperf debe calibrarse apropiadamente para obtener un valor de *throughput* óptimo, por lo tanto, la mayoría de los parámetros de esta prueba son específicos a cada enlace. A continuación se puede ver la construcción del comando Iperf para un enlace:

```
1
2  for link in links.values():
3          test_this_link = kwargs.get("test this link_%s"%link["id"],False)
4          if test_this_link:
5                  _time = kwargs.get("time to transmit_%s"%link["id"],None)
6                  _bytes = kwargs.get("bytes to send_%s"%link["id"],None)
7                  flows = kwargs.get("number of flows_%s"%link["id"],1)
8                  port = kwargs.get("port_%s"%link["id"],5001)
9
10                 args = ["iperf","-c",link["ip"],"-y","C","-p",str(port)]
11                 if _time != None:
12                         args.append("-t")
13                         args.append(str(_time))
14                 elif _bytes != None:
15                         args.append("-n")
16                         args.append(_bytes)
17                 if flows != None:
18                         args.append("-P")
19                         args.append(str(flows))
20                 out = check_output(args) #ejecuta el comando y guarda la salida en la variable out
```

4.3 Tentacle Probe Destination

Los Tentacle Probe Destination son los nodos de interés a monitorear, cualquier nodo en la red es candidato a convertirse en un TPD, sin embargo, el usuario deberá tener en cuenta las características del nodo y del enlace al elegir las pruebas que recolectaran datos del enlace. La mayoría de los nodos de una red IP son capaces de responder a mensajes del protocolo ICMP y por lo tanto será posible monitorearlos a través de

Parámetro	Tipo	Descripción
Probar este enlace	Booleano	Determina si se debe o no ejecutar la prueba para un enlace especifico.
Tiempo de transmisión	Entero	Numero de segundos para transmitir datos durante la prueba.
Bytes a enviar	String	Cantidad de bytes a enviar durante la prueba, se puede especificar si el numero es en bytes o megabytes usando las letras B o M respectivamente, este parámetro se ignora si se establece el parámetro anterior..
Numero de flujos	Entero	Numero de flujos TCP simultáneos a usar durante la prueba.
Puerto	Entero	Especifica el puerto en que el servidor está escuchando en el TPS.

Tabla 4.5: Parámetros de la prueba con iperf

las pruebas de ping o traceroute. Otros ejemplos en que aprovechamos características ya disponibles en los nodos de una red es al monitorear servidores web a través del protocolo HTTP.

Mientras es posible obtener información de los TPD sin necesidad de tener privilegios para instalar y ejecutar software en ellos, determinar ciertas métricas como el *throughput* requiere que un programa se ejecute en el TPD y esté esperando peticiones por parte del TPS. Sumado a esto, el usuario deberá asegurar que el monitoreo no entorpezca el funcionamiento del TPD, por ejemplo, un intervalo entre pruebas de un segundo para la prueba de HTTP agregaría decenas de peticiones por minuto lo cual estaría desperdiciando sus recursos internos innecesariamente.

Capítulo 5

Framework de integración de pruebas

A pesar de que Octopus Monitor incluye un conjunto de pruebas como ping, httping, traceroute y Iperf que proveen datos sobre latencia, rendimiento, disponibilidad y alcanzabilidad de los enlaces y servicios monitoreados, una de sus características mas importantes es la facilidad de implementación de nuevas pruebas.

El framework de integración de pruebas le ahorra al desarrollador la repetición de las tareas comunes a todas las pruebas como definición de URLs, formularios de parámetros, lógica de las vistas o escritura de plantillas. Gracias a esto, el desarrollador se puede concentrar en escribir solo el código especifico a cada prueba particular facilitando y acelerando la expansión de la aplicación.

Para entender los requisitos que resultarán en los distintos componentes que serán parte del framework, debemos entender el flujo de trabajo del monitoreo a través del cual se obtienen datos de las redes monitoreadas:

1. El usuario selecciona una prueba de la lista de pruebas disponibles para el monitor.

2. La aplicación web despliega un formulario donde el usuario puede planificar la ejecución de la prueba seleccionada y establecer sus parámetros.

3. La aplicación web sube un mensaje a la carpeta monitor de red con los detalles

de la prueba planificada.

4. El monitor es notificado del nuevo mensaje, lee su contenido y prepara la prueba para su ejecución

5. El monitor ejecuta la prueba y guarda los resultados en un archivo de trazas temporal

6. El monitor sube los archivos generados a la nube para su almacenamiento a largo plazo

7. La aplicación web inicia una rutina de sincronización para descargar los archivos, los procesa según el formato del archivo y los guarda en la base de datos.

8. El usuario selecciona una visualización de la lista de visualizaciones disponibles y selecciona los datos y las opciones a través de un formulario.

9. El servidor busca la visualización solicitada en el caché, si no la encuentra, llama a la función correspondiente para computarla, la introduce en el caché para futuras consultas y la muestra al usuario.

A partir de este flujo podemos inferir los siguientes requisitos:

1. Un módulo de planificación genérico de pruebas que incluya los siguientes componentes:

   - Definiciones de las pruebas disponibles
   - Una lista de parámetros para cada una de las pruebas
   - Un formulario para permitir la selección de los parámetros cada vez que el usuario planifica una prueba

2. Un módulo de sincronización genérico que pueda procesar los archivos de trazas y almacenar los datos de las pruebas en la base de datos.

3. Un módulo de análisis y visualización de los datos recolectados que incluya los siguientes componentes:

- Definiciones de las visualizaciones disponibles para los datos de cada prueba.
- Un formulario de configuración para cada visualización.
- Una función para computar cada visualización.
- Un mecanismo de *caching* genérico.
- Una plantilla para dibujar cada visualización.

5.1 Planificación genérica de pruebas

La planificación genérica de pruebas provee una interfaz única que permite a los usuarios agregar pruebas a ejecutar para sus monitores remotos. Como se explicó en la Sección 3.1.6 mantenemos en la base de datos dos tablas que describen las pruebas que son parte del sistema, la tabla *Test* que incluye los detalles básicos de cada prueba como su nombre, descripción y nombre de módulo y la tabla *Parameter* que incluye el nombre, tipo, valor por omisión, ámbito y otros detalles de los parámetros. Una prueba puede ser vista como una agregación de sus parámetros.

Las pruebas planificadas para un monitor se guardan en la tabla intermedia *MonitorTest*, además de guardar las claves del monitor y la prueba relacionada, esta tabla guarda los parámetros de ejecución en formato JSON y el momento de la planificación.

Es posible ejecutar pruebas según dos esquemas de planificación. Primero, **pruebas periódicas** que son ejecutadas en intervalos regulares, y **pruebas de una sola ejecución**. Algunas pruebas como las de ping y httping están pensadas para ser ejecutadas de forma periódica y así obtener datos de la evolución de una variable a través del tiempo (Ver Sección 4.2.4). Otras pruebas, como capturas de tráfico de la red, requieren ser ejecutadas en momentos específicos para asegurar que se obtengan los datos deseados.

Cuando el usuario selecciona una prueba, el sistema crea un formulario partir de la lista de parámetros. Los parámetros de las pruebas pueden ser "globales" de forma tal que se apliquen a todos los enlaces monitoreados o "específicos" de forma tal que se puedan establecer por enlace. Por ejemplo, la prueba de httping tiene parámetros globales como el *timeout* y parámetros específicos a los enlaces como *path* y *port*.

5.1.1 Formulario de Parámetros de la Prueba

Los formularios en Django se modelan a través de la clase *Form* que provee todas las funcionalidades necesarias para el despliegue de formularios y validación de datos introducidos por el usuario. Cuando deseamos crear un nuevo formulario debemos crear una subclase de *Form* e incluir todos los campos que serán parte del formulario y cualquier validación especifica a dichos campos.

Ya que cada prueba tiene una lista de parámetros específica que además puede depender de los enlaces de un monitor TPS, es necesario crear formularios de forma dinámica. Para esto, python ofrece características de meta-programación que permiten crear clases en tiempo de ejecución. En este caso, definimos una función capaz de generar subclases de la clase *Form* y que tengan como atributos los campos que serán parte del formulario.

El Algoritmo 9 muestra la estructura básica para agregar campos al formulario según el tipo de datos de los parámetros, sin embargo cada parámetro también incluye datos para determinar si el campo es obligatorio, incluir textos de ayuda y proveer valores por omisión.

La clase generada luego es pasada a una plantilla genérica capaz de dibujar cualquier formulario dado, cuando el usuario introduce los datos, estos son validados por la misma forma. El formulario generado incluye validaciones básicas de tipos de datos, en un trabajo futuro se podrían incluir validaciones mas complejas como intervalos máximos y mínimos para los capos numéricos o validaciones personalizables para campos de tipo *string* que puedan ser adjuntados fácilmente a los parámetros existentes.

5.1.2 Formulario de Planificación

Junto con el formulario de parámetros de la prueba, debemos incluir los campos para introducir las fechas iniciales, finales y intervalo entre pruebas en caso de ser una prueba periódica o el tiempo de ejecución en caso de ser una prueba de una sola ejecución.

La validación de este formulario es usada para asegurar las reglas de negocio del sistema:

1. La fecha inicial y la fecha final no pueden ser previas al tiempo actual.

Algoritmo 9 Generación dinámica de formularios

```
1: procedimiento MAKE_PARAMETER_FORM(test,monitor)
2:     parameters ← get all parameters for test
3:     form_fields ← empty ordered dictionary
4:     para field en parameters hacer
5:         si field is global entonces
6:             form_fields[field.name] ← GET_FIELD(field.type)
7:         si no
8:             links ← all monitor links
9:             para link en links hacer
10:                key ← concatenate field.name '_' link.id
11:                form_fields[key] ← GET_FIELD(field.type)
       devolver Una clase que herede de FORM conteniendo los form_fields
12: procedimiento GET_FIELD(type)
13:    si type is boolean entonces devolver boolean field
14:    si no si type is integer entonces devolver integer field
15:    si no si type is float entonces devolver float field
16:    si no si type is string entonces devolver char field
17:    si no
18:        arrojar excepcion "Tipo de dato no soportado"
```

2. la fecha final no puede ser previa a la fecha inicial.

3. La fecha inicial no puede ser modificada después de que la prueba ya ha comenzado.

4. El intervalo entre pruebas no puede ser menor o igual que 0.

5. Solo puede ser planificada una prueba periódica simultánea del mismo tipo por monitor.

Todas las fechas se validan usando la zona horaria del monitor, para validar la condición 5. se retiran de la base de datos todos los otros *MonitorTest* pertenecientes

a dicho monitor y cuya fecha final sea mayor al tiempo actual (es decir, aquellos que se estén ejecutando o se van a ejecutar en el futuro) y se valida que la prueba a insertar en el plan de monitoreo no ocurra de forma concurrente con ninguna otra.

5.2 Sincronización genérica de datos

El mecanismo general de recolección de datos se ha explicado detalladamente en la Sección 3.2.3, se ha dicho que por cada prueba activa para el monitor se llama a una subrutina de sincronización que descarga los archivos de trazas e introduce los resultados en la base de datos, sin embargo no se ha entrado en detalle sobre el método de procesamiento de los archivos y la forma en que los datos de cada prueba se almacenan en la base de datos. A continuación, se discuten los elementos que conforman la sincronización genérica de datos.

5.2.1 Almacén de datos

Se le da el nombre de almacén de datos al conjunto de tablas relacionadas donde se guardan los resultados de un prueba. En este trabajo la mayoría de los almacenes de datos consisten de una sola tabla donde se guarda una marca de tiempo, el enlace relacionado, y uno o mas valores conteniendo los resultados obtenidos para las métricas observadas. Ya que los resultados de cada prueba poseen una estructura particular, es necesario agregar a la estructura de la base de datos tablas ajustadas a las características particulares de cada prueba y optimizadas para facilitar y acelerar el computo de las visualizaciones relacionadas.

El desarrollador tiene la libertad de elegir cualquier estructura que vea conveniente para almacenar los datos de las pruebas. Ya que Django maneja la base de datos a través de un ORM (ver Sección 2.6.5.2), definir almacenes de datos consiste en declarar un conjunto de clases y las relaciones entre ellas. Django se encargará de determinar los cambios realizados a la estructura de la base de datos y construir las tablas necesarias o asentar cambios realizados a tablas existentes.

5.2.2 Procesamiento de archivos de trazas

Ya que los requisitos y características de cada prueba son distintas, el desarrollador de las pruebas tiene la libertad de establecer cualquier formato al guardar sus archivos de trazas, es por esto que el sincronizador no tiene una forma "genérica" de procesar dichos archivos. Depende entonces del desarrollador escribir la función o *parser* que tome como entrada un archivo de trazas en el formato establecido para el tipo de prueba y lo transforme en objetos a introducir en la base de datos.

Para elegir la función correcta para procesar un archivo de trazas específico, el mecanismo de sincronización busca en la base de datos la dirección absoluta del método y lo importa. Python incluye funcionalidades para importar módulos en tiempo de ejecución y pasar métodos a modo de variables (Línea 4, Algoritmo 10).

Algoritmo 10 Getter de la función de sincronización

```
1: procedimiento GET_PARSER(test)
2:     function_path ← test.sync_function      ▷ Se obtiene la ruta absoluta del parser
3:     module_name,function_name ← split function_path      ▷ Se separa la ruta
   separando el nombre de la funcíon del nombre del módulo
4:     module ← IMPORT(module_name)
5:     function ← GET(module, function_name)  ▷ Retorna la funcion buscándola en
   el modulo
       devolver function
```

Es responsabilidad del desarrollador tanto escribir la ruta correctamente en la base de datos como implementar el *parser* en sí, la función puede vivir en cualquier parte del código pero por convención se colocan en el módulo *parsing.py* dentro del paquete *management*.

El *parser* debe manejar archivos dañados o mal formados evitando propagar excepciones al mecanismo general de sincronización y debe imponer las reglas de negocio específicas a la prueba. Por ejemplo, no tendría sentido introducir a la base de datos valores negativos para el número de saltos para alcanzar un *host* o valores imposiblemente altos. Introducir valores espurios a la base de datos resultará en visualizaciones inesperadas o erróneas.

En un trabajo futuro se podría incluir un esquema de archivos de trazas genéricos en los cuales cada línea del archivo representa una traza y cada traza viene dada por un diccionario en formato JSON. Luego el *parser* puede leer el archivo, mapear los diccionarios a objetos, e introducirlos en la tabla correspondiente a la prueba en la base de datos. Esto se podría ajustar perfectamente a pruebas ya implementadas como las de ping, httping y Iperf (ver Sección 4.2.4), por ejemplo, para la prueba ping un objeto JSON como `{"time": 1454013808, "rtt": 154, "icmp": 4}` podría decodificarse como un diccionario que a su vez se puede pasar como argumento al constructor de la clase *TracePing*.

5.3 Visualización genérica de datos

Después de haber recolectado y almacenado los datos de las pruebas, el último paso y objetivo final del sistema de monitoreo es la visualización de los datos. Las visualizaciones son cargadas y desplegadas a través de una interfaz genérica que incluye un formulario para ajustar los parámetros de la visualización y una barra de herramientas para realizar acciones como re-computar, crear enlaces directos o ver en una nueva pestaña.

Las visualizaciones, al igual que las pruebas, se guardan en su propia tabla en la base de datos; esta tabla lleva el nombre de *ResultView* (ver Tabla 3.7), y sirve para ayudar al cargador genérico de visualizaciones tanto a computar gráficas como crear URLs legibles y desplegar el formulario apropiado cuando se genera la gráfica.

Entre los tipos de visualizaciones incluidos en el *framework* están los mapas de calor, gráficas de barras, mapas geográficos, tablas, gráficas lineales y logarítmicas, gráficos de área, entre otros.

5.3.1 Construcción de URLs genéricos

En el contexto del desarrollo de un *framework* en el tope de una aplicación web, debemos tomar en cuenta que el servidor debe tener una manera de identificar las peticiones y poder relacionarlas a una vista específica. En la mayoría de los casos sencillamente incluimos el identificador del objeto en el URL, por ejemplo, la dirección

"/monitor/2/" identifica de forma única la página del monitor con ID 2. Podríamos identificar las visualizaciones de la misma manera, sin embargo, preferimos usar cadenas de caracteres legibles por humanos que identifiquen la visualización por su nombre. A este tipo de cadenas de caracteres se les da el nombre de *slug*[1]. Cuando deseamos ver una visualización específica en vez de visitar un URL como "/monitor/2/view/1" se visita un URL legible como "/monitor/2/view/average-rtt-heatmap".

Para generar el *slug* correspondiente a la visualización, sobrescribimos el método *save* de la clase *ResultView*. Ya que solo deseamos generar el *slug* la primera vez que se guarda el registro, revisamos si el id es nulo (solo se le asigna el id a un objeto después de que es guardado por primera vez). Django incluye un método llamado *slugify* que transforma el nombre de la visualización en un *slug* válido. Finalmente debemos asegurarnos de que el nuevo *slug* sea único, para esto buscamos en la base de datos otro *ResultView* con el mismo *slug* y si se encuentra anexamos al final un guión y un número, en caso de que este *slug* también exista se repite el proceso incrementando el número hasta hallar un *slug* único.

5.3.2 Interfaz de visualización

Cuando un usuario selecciona una visualización de la lista de visualizaciones disponibles para su monitor, el servidor elije el formulario correspondiente, cada visualización tiene distintos parámetros de configuración. Por ejemplo, algunas pueden pedir al usuario que introduzca un tiempo inicial y un tiempo final para filtrar los datos, otras podrían tener opciones para seleccionar un mes específico, o un período de tiempo desde el momento actual como "últimas 24 horas" o "la semana pasada". Algunas visualizaciones toman en cuenta los datos de todos los tentáculos de un monitor, otras piden seleccionar primero un tentáculo específico.

Ya que los formularios de las visualizaciones generalmente son similares entre sí, el framework incluye algunos formularios que pueden ser reutilizados o extendidos para lograr cualquier comportamiento deseado. Todos los formularios de visualizaciones heredan de un formulario base llamado *BaseResultForm*, este tiene como atributo el

[1] los *slugs* se construyen reemplazando todos los espacios en el nombre por guiones (-) y los caracteres especiales por sus equivalentes en ASCII

identificador del monitor con la finalidad de facilitar la construcción de formularios ajustados al monitor. Por ejemplo, la clase *LinkSelectionForm* crea un menú para seleccionar de entre la lista de enlaces del monitor, todos los formularios predefinidos pueden ser vistos en la tabla 5.3.2.

Los formularios no son mas que clases que podemos importar con el método definido en el Algoritmo 10, la tabla *ResultView* guarda una referencia a la clase del formulario en la columna *form_class*.

Clase	Clase Base	Descripción
BaseResultForm	Form	Formulario base para todas las visualizaciones
LinkSelectionForm	BaseResultForm	Formulario para seleccionar enlace de un monitor
SampleForm	LinkSelectionForm	Formulario para seleccionar un número de muestras a graficar
DateRangeForm	BaseResultForm	Formulario para seleccionar un período de tiempo a graficar
ElapsedForm	BaseResultForm	Formulario para seleccionar un período de tiempo a partir del momento actual
LinkMultiselectForm	BaseResultForm	Formulario para seleccionar un conjunto de enlaces de un monitor

Tabla 5.1: Formularios predefinidos

Una característica de los formularios es que se les puede "adjuntar" archivos JavaScript y hojas de estilo CSS, luego la plantilla que dibuja el formulario puede vincular estos archivos y así obtener cualquier comportamiento deseado en el formulario. Por ejemplo, el formulario de los mapas de calor de RTT incluye Javascript para mostrar y esconder campos según las selecciones del usuario.

Cuando el usuario hace click en el botón "PLOT" comienza el proceso de validación del formulario, una rutina JavaScript toma el control, extrae los valores seleccionados del formulario y los envía al servidor a través de una petición POST, el servidor valida el formulario y genera una respuesta JSON:

- Si el formulario es válido, el servidor codifica un URL de la visualización con los parámetros en formato GET y un "URL directo".
- Si el formulario es inválido el archivo JSON contiene un código HTML que luego se incrusta en el tope del formulario e indica los errores de validación.
- Si no se recibe respuesta del servidor o el código de estado de la respuesta es distinto de "200 OK", entonces la rutina JavaScript muestra un mensaje indicando al usuario que ocurrió un problema.

Para desplegar las visualizaciones se usa un elemento HTML llamado iframe. Este elemento permite incrustar páginas web dentro de otras a partir de un URL dado. La rutina JavaScript le pasa al iframe el URL retornado en el archivo JSON y el iframe a su vez carga el URL que devuelve una página conteniendo la visualización.

En el caso de visualizaciones como los mapas de calor o periodos de actividad continua que pueden tardar largos periodos de tiempo en computarse, para ofrecer al usuario retroalimentación es recomendable mostrar algún tipo de animación de carga o indicador de progreso de la operación. Para implementar esta funcionalidad, se muestra una animación de carga inmediatamente después de que el usuario hace click en el botón "PLOT" o "Re-compute", luego es posible aprovechar una señal emitida por el iframe cuando obtiene respuesta del servidor para ocultar esta animación.

La característica de re-computar visualizaciones funciona de manera análoga al flujo normal de visualización, con la única diferencia de que se solicita al cargador genérico que ignore el caché y force el computo de la visualización.

5.3.3 Cargador genérico de visualizaciones

El cargador genérico de visualizaciones maneja la tarea de retornar visualizaciones listas en formato HTML. El cargador recibe el *slug* correspondiente a la visualización y una cadena de parámetros GET y no computa las visualizaciones inmediatamente, sino que primero se asegura de que la visualización exista en el caché.

Se puede decir que una visualización existe en el caché cuando los parámetros de dicha visualización son los mismos que los parámetros GET pasados al cargador. Para

facilitar la búsqueda en el caché, las visualizaciones se identifican a través de un código *hash* que se calcula a partir de la lista de parámetros, luego se puede buscar el archivo que contiene la visualización por si nombre en la carpeta correspondiente. Ya que la búsqueda en un directorio depende de la cantidad de archivos en el, dividimos el caché por carpetas para acelerar las consultas, las carpetas tienen como nombre el *slug* de la visualización a la que pertenecen, por lo tanto un archivo en el caché se identifica como: $CACHE\_ROOT/ < slug > / < hash > .html$

Cuando una búsqueda en el caché no obtiene resultados, entonces el sistema pasa a computar la gráfica, para esto, se guarda una referencia a la función que computa la visualización en la base de datos, y esta se carga dinámicamente como ya se vio en el Algoritmo 10. Las funciones de cómputo de visualizaciones se guardan dentro del paquete *visual*, en un módulo con el nombre $< test > \_visual.py$ donde test es el nombre de módulo de la prueba a la que pertenece dicha visualización.

El único requisito para las funciones de computo es retornar una cadena de caracteres válida en formato HTML (sin embargo los formatos CSV y JSON también son soportados). En caso de retornar algún otro tipo objeto (o retornar nulo) el usuario verá la representación en *unicode* de dicho objeto en la Interfaz de Visualización.

5.3.3.1 Plantillas genéricas

Las plantillas son archivos que se usan para introducir el contenido variable de una página web dinámica a través del uso de un micro-lenguaje, el desarrollador tiene la libertad que escribir cualquier código HTML que desee, así como vincular cualquier dependencia como hojas de estilo, imágenes o archivos JavaScript. Para este trabajo se ha usado principalmente HighCharts y Google Maps, para dibujar la mayoría de las visualizaciones.

El desarrollador debe seguir una serie de pautas al escribir plantillas para visualizaciones, ninguna de ellas es obligatoria pero son recomendables para asegurar que el usuario tenga una experiencia consistente:

- La plantilla debe ajustarse responsivamente al ancho de la pantalla.
- Siempre se deben usar HighCharts o Google Maps sobre otras bibliotecas que

tengas las mismas funcionalidades o similares.

- Las visualizaciones deben estar preferiblemente sobre fondo blanco.
- En caso de que no se encontraran datos para el período seleccionado, se debe mostrar un mensaje de alerta en vez de una visualización en blanco.
- Se deben usar las hojas de estilo de Bootstrap al dibujar visualizaciones que estén compuestas de elementos HTML.

Para simplificar la tarea de escribir código repetitivo, se ha incluido un conjunto plantillas predefinidas, algunas de ellas pueden ser extendidas para escribir nuevas visualizaciones, otras son plantillas listas para usar que aceptan alguna estructura de datos y resultan en una visualización, las plantillas predefinidas se pueden ver en la Tabla 5.2.

5.4 Integración de nuevas pruebas

Habiendo entendido todos los componentes que hacen posible el monitoreo de redes, solo queda explicar como usarlos para integrar rápidamente una prueba en un entorno de producción. El *workflow* de integración de pruebas consiste de dos etapas: la etapa de desarrollo y la etapa de despliegue, esto puede ser visto en la Figura 5.1.

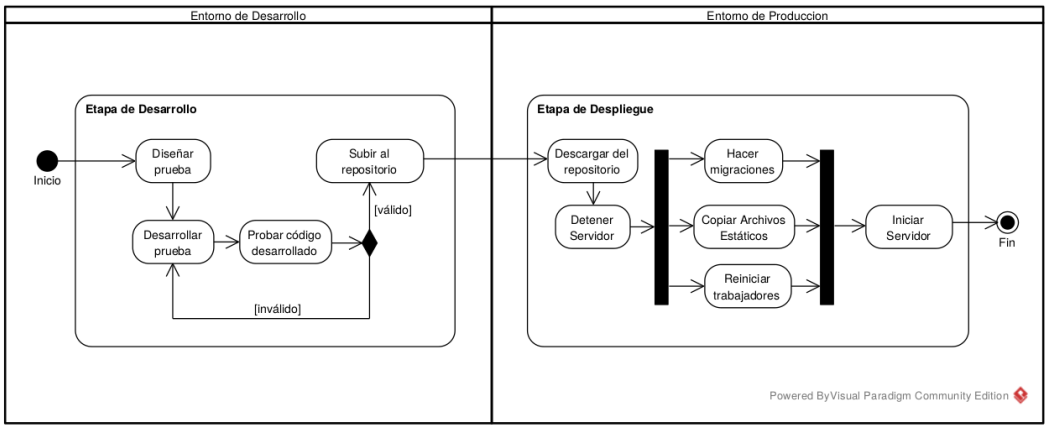

Figura 5.1: El *workflow* de integración de pruebas

Plantilla	Descripción
base-visual.html	Plantilla base para todas las visualizaciones
base-highcharts.html	Plantilla base para todas las visualizaciones con HighCharts
base-highstock.html	Plantilla base para todas las visualizaciones con HighStock (módulo de Hihcharts para dibujar *stockcharts*)
base-googlemaps.html	Plantilla base para visualizaciones con Google Maps.
generic-heatmap.html	Plantilla base para mapas de calor de tipo <hora,tiempo,valor>
generic-stock.html	Plantilla base para *stockcharts* de tipo *tiempo*, *valor*
generic-stock-multi.html	Plantilla base para *stockcharts* de tipo <tiempo,valor>con múltiples series.
generic-line.html	Plantilla base para gráficas de linea de tipo <time,value>
generic-line-multi.html	Plantilla base para gráficas de linea de tipo <time,value>con múltiples series,

Tabla 5.2: Plantillas Predefinidas

5.4.1 Etapa de Desarrollo

Durante la etapa de desarrollo se diseñan, implementan y validan las pruebas y visualizaciones que serán parte del sistema en el futuro próximo. La etapa de desarrollo ocurre en un entorno de desarrollo que se configura para ser lo más parecido posible al entorno de producción y así evitar potenciales inconsistencias que produzcan en errores al momento del despliegue.

Para desarrollar código fácilmente en distintos entornos se usa un repositorio de código git[2]. Los cambios hechos en el entorno de desarrollo pueden ser descargados en el entorno de producción y aplicados al servidor web, los trabajadores Celery y la base de datos. Todos los componentes deben compartir la misma versión del código antes de ir a producción para evitar comportamientos inesperados durante su interacción.

[2]git https://git-scm.com/

Cuando se desea desarrollar una prueba, el primer paso es entender y describir los requisitos de la prueba, para facilitar el entendimiento de esta sección usaremos como ejemplo el diseño una prueba que determine la pérdida de paquetes usando el comando ping en modo *flood* (inundación). Para eso haremos que ping envíe paquetes tan rápido como pueda hasta pasar una cierta cantidad de paquetes luego determinaremos cuantos se perdieron durante la prueba.

El módulo de prueba se muestra a continuación, cada ejecución de la prueba anexa una linea en formato <marca de tiempo,paquetes enviados, paquetes recibidos> al archivo de trazas correspondiente:

```
1
2  def parse(out,link_id):
3      time = str(time.strftime("%d_%m_%y_%H"))
4      filename = os.path.join(DATA_FOLDER,"pktloss","link_%s_%s.txt"%(link_id,time))
5      output_file = open(filename,'a')
6
7      lines= out.split('\n')
8
9      for line in lines:
10         if "packet loss" in line:
11             tokens = line.split()
12             sent = tokens[0]
13             recieved = tokens[3]
14             timestamp = time.time() #gets current time
15             output_file.write(str(timestamp)+" "+sent+" "+recieved+"\n")
16             break
17     output_file.close()
18
19 def run(kwargs):
20 """executes ping command for each of the test links specified in the config file
21 parameters:
22 -count number of icmp packages to send
23 -max_duration: maximum test durationg
```

```
24 """
25     count = kwargs["count"]
26     max_duration = kwargs["maximum duration"]
27     config = get_config()
28     if config is None:
29         return
30     links = config["links"]
31 
32     for link in links.values():
33         if not link["status"]:
34             continue
35         try:
36                 args = ["ping",link["ip"],"-f","-c",str(count),"-w","str(max_duration)"]
37             output = check_output(args)
38             parse(output,link["id"])
39         except Exception as e:
40             log.error(str(e))
```

Tras tener definido el módulo de prueba y el formato del archivo de trazas, pasamos a desarrollar el código que estará del lado de la aplicación web. El primer paso es definir el almacén de datos que alojará las trazas de esta prueba. Para esto, declaramos la clase *TracePaketLoss*, esta consiste de un marca de tiempo (*time*), el identificador de enlace (*link*), el número de paquetes enviados (*sent_packets*), el número de paquetes recibidos (*recieved_packets*), el porcentaje de pérdida de paquetes (*percentage*) y un índice que indexa las columnas *time* , *link* y *percentage*. A continuación se muestra la definición de dicha clase:

```
1 
2 class TracegPaketLoss(models.Model):
3     link = models.ForeignKey(Link)
4     time = models.DateTimeField()
5     sent_packets = models.IntegerField
6     recieved_packets = models.IntegerField()
7     percentage = models.SmallIntegerField()
```

```
8
9     def save(self, *args, **kwargs):
10        self.percentage = self.recieved/float/(self.sent)
11        super(TracegPaketLoss, self).save(*args, **kwargs)
12
13    class Meta(TraceBase.Meta):
14        index_together = [
15                ["link", "time","percentage"],
16            ]
17      unique_together = ("time", "link")
```

Cuando hacemos cambios a la estructura de la base de datos en el código de la aplicación, debemos crear archivos llamados *migrations* que describen dichos cambios y permiten asentarlos en la base de datos fácilmente a través de un comando. Para crear *migrations*, Django incluye el comando *makemigrations*, la primera vez que se ejecuta este comando, se crea una migración inicial que describe la estructura completa de la base de datos. Llamadas posteriores a *makemigrations* calcularán las diferencias a partir del estado anterior, conformando un historial de cambios que luego pueden ser aplicados en el entorno de producción.

El siguiente paso es crear el *parser* del archivo de trazas, puesto que el archivo de trazas viene en formato CSV es muy sencillo separar cada linea por el carácter ',', convertir las cadenas de caracteres en sus tipos de dato correspondiente y guardar cada traza en la base de datos, esta función se puede observar a continuación:

```
1
2   def parse_packet_loss(f,filename,monitor):
3
4       tokens = filename.split('/')
5       filename_tokens = tokens[len(tokens)-1].split('_')
6       link_id = int(filename_tokens[1])
7       timezone = monitor.timezone
8
9       for line in f.readlines():
10          tokens = line.split()
```

```
11        if len(tokens) < 3: continue
12        time, sent, received = tokens[0], tokens[1], tokens[2]
13        try:
14            sent = int(sent)
15            received= int(received)
16            time = datetime.datetime.fromtimestamp(float(time),timezone)
17        except Exception: continue
18        trace = TracePacketLoss(link_id=link_id,time=time,sent_packets=sent,received_packets=received)
19
20        try: trace.save()
21        except IntegrityError: pass
```

Ahora que tenemos los elementos necesarios para alojar los datos obtenidos de las pruebas solo nos queda definir las visualizaciones que harán uso de esas pruebas. Para efectos de este ejemplo, definiremos una visualización que muestre una gráfica de stock del porcentaje de pérdida de paquetes desde el momento actual hasta una cantidad dada de tiempo en el pasado como 24 horas.

A continuación se puede observar el código de la visualización, primero tomamos del diccionario de parámetros el número de horas seleccionadas y luego hacemos una consulta a la base de datos extrayendo todas las trazas que se encuentren en el período de tiempo. Seguidamente recorremos el arreglo para localizar las marcas de tiempo a la zona horaria del monitor, finalmente llamamos al método *render_to_string* que dibuja la plantilla, en este caso usamos la plantilla genérica *generic-stock.html*.

```
1  def generate_packetloss_sample_stock(parameters):
2      link_id = parameters["link"]
3      period = int(parameters["period"])
4      link = Link.objects.get(id=link_id)
5      min_time = datetime.datetime.now() - datetime.timedelta(hours=period)
6
7      rows = TracePacketLoss.objects.values_list('time', 'elapsed').filter(time__gt=min_time,
8                                                                           link_id=link_id,
9                                                                           elapsed__gt=0)
10     array = []
```

```
    for row in rows:
        array.append((row[0].astimezone(link.monitor.timezone),row[1]))

    return render_to_string("default_stock.html",{"rows":array,
            "title":"Elapsed Time Stock Chart",
            "series_name":"Sample Elapsed Time"})
```

Habiendo definido todas las clases y funciones necesarias, solo falta introducir en la base de datos los registros necesarios para que el sistema sepa de la nueva prueba y donde buscar las funciones correspondientes cuando necesite importarlas. Para esto, usamos archivos llamados *fixtures* que permiten definir los registros que queremos introducir a la base de datos en formato JSON y luego insertarlos en el entorno de producción con el comando *loaddata*.

El *fixture* incluirá entonces, un registro para la prueba, los registros para los parámetros de la prueba y la visualización. A continuación se puede ver el *fixture* para la prueba de pérdida de paquetes:

```
[
  {
    "model": "main.test",
      "pk": 5,
      "fields": {
        "name": "Packet Loss",
        "module_name": "pktloss",
        "description": "Sends a burst of ICMP packages in quick succession and counts the number
         of lost packets",
        "is_periodic": true,
        sync_function: "octopusmonitor.management.parsing.parse_packet_loss"
      }
  },
  {
      "model": "main.parameter",
      "pk": 15,
      "fields": {
```

```
18            "name": "Count",
19            "description": "Number of packets to send",
20            "type": "i", (integer)
21            "test_id": 5,
22            "is_global": true,
23            "default_value": "1000",
24            "required": true
25          }
26    },
27    ...
28    {
29      "model": "main.result_view",
30      "pk": 26,
31      "fields": {
32        "name": "Packet loss % Stockchart",
33        "description": "Displays a stockchart of the last packet loss data",
34        "type": "k", (stock chart)
35        "test_id": 15,
36        "form_class": "octopusmonitor.main.forms.ElapsedForm",
37        "generating_function" "octopusmonitor.visual.packet_loss_visual.generate_packetloss_sample_stock":
38      }
39    }
40  ]
```

5.4.2 Etapa de Despliegue

En la etapa de despliegue se aplicarán, en el entorno de producción, todos los cambios realizados en la etapa de desarrollo. Esta etapa consiste en una serie de pasos intencionalmente simples y rápidos para evitar largas interrupciones de servicio mientras que estos son aplicados. Después de que finaliza la etapa de despliegue la nueva prueba estará disponible para que los usuarios la incluyan a sus monitores.

La etapa de despliegue comienza descargando la última versión del código del repositorio y parando el servidor web para evitar cualquier petición durante este

período, es buena idea planificar los despliegues para momentos de baja carga del servidor, y notificar a los usuarios del sistema.

Para actualizar el servidor web y los trabajadores basta con reemplazar el código que estos ejecutan, sin embargo, para actualizar la base de datos se hace uso de archivos de migraciones que se crean en la etapa de desarrollo. las migraciones pueden ser aplicadas con el comando *migrate*, para saber cual fue la última migración aplicada Django mantiene una tabla para llevar cuenta del estado de la base de datos, así es posible saber cuales de las migraciones deben ser aplicadas. Se debe tener especial cuidado al ejecutar migraciones, algunas migraciones incluyen operaciones costosas como agregar indices a tablas existentes o actualizar todas las entradas de una tabla, también es importante tomar en cuenta que las migraciones no incurran en conflictos con datos contenidos en la base de datos, por ejemplo, aplicar una restricción de unicidad sobre una columna con valores repetidos.

En la Sección 3.1.3 se dijo que la aplicación web Django no se encarga de servir los archivos estáticos al cliente, es por esto que estos deben ser copiados a la carpeta correspondiente ya sea en el propio servidor web o en un servidor externo, para esto Django incluye el comando *collectstatic* que busca en cada paquete los archivos estáticos y los copia a dicho directorio.

Finalmente, se reinician los componentes que lo requieran de modo que estos puedan importar y ejecutar el nuevo código. Solo es necesario reiniciar el servidor web y los trabajadores, los demás micro-servicios como *broker* de mensajería, planificador y *backend* de resultados no requieren ser reiniciados pues son indiferentes al código de la aplicación.

Capítulo 6

Conclusiones y Recomendaciones

6.1 Conclusiones

Durante el desarrollo de este trabajo, construimos un sistema de monitoreo de redes de propósito general, ligero y extensible, el sistema permite recolectar y almacenar datos sobre el estado de la red a partir de pruebas planificadas y observar los resultados obtenidos a largo plazo. Para el diseño e implementación del sistema se usó una metodología en espiral en que se agregaron nuevas características incrementalmente a cada uno de los componentes del sistema.

Los logros mas relevantes desde un punto de vista computacional se nombran a continuación:

- Se desarrolló una aplicación web que permite a usuarios autenticados manejar de forma remota sus monitores de red, recolectar datos y finalmente observar y explorar los datos obtenidos a partir del monitoreo.

- Construimos un monitor de red con la cantidad de código mínimo para planificar tareas de monitoreo periódico de acuerdo al plan de monitoreo definido en la aplicación web. El monitor escrito en Python es ligero e ideal para desplegar en dispositivos de bajo costo (ver Capitulo 4.2).

- Establecimos un esquema de sincronización a costo cero y tolerante a fallas entre la aplicación web y el monitor a través del uso de cuotas de almacenamiento en

la nube y peticiones gratuitas al API de Dropbox.

- Para la aplicación web se diseñó una arquitectura a cuatro capas capaz de manejar eficientemente las distintas tareas de la aplicación web, esta arquitectura permite escalar cada uno de sus componentes tanto como sea necesario facilitando el manejo de mayores cargas de trabajo y usuarios.

- Para la aplicación web se diseñó e implementó una interfaz sencilla y fácil de usar que se ajusta de manera fluida a la resolución de pantalla de cualquier dispositivo.

- Se desarrolló un *framework* para integrar fácilmente nuevas pruebas y visualizaciones al sistema. Se encontraron los puntos comunes y las tareas repetitivas relacionadas con la implementación y despliegue de nuevas pruebas en un sistema en fase de producción (ver Capítulo 5).

- Se desarrolló una batería de pruebas para que fueran parte del sistema, así como para determinar los requisitos y pormenores del *framework* de integración de pruebas:

  - **Prueba de Ping:** fue la primera prueba implementada y se usa para determinar la latencia y disponibilidad de los enlaces, esta prueba se usó para inferir congestión en la red y periodos de actividad e inactividad.
  - **Prueba de Httping:** similar a la prueba anterior, se usa para determinar el tiempo de respuesta de servidores web usando peticiones HTTP.
  - **Prueba de Traceroute:** se usó para determinar la alcanzabilidad de un nodo a través de un enlace e inferir detalles como congestión a nivel de nodos intermedios. Su implementación ayudó a determinar los requisitos de pruebas con resultados no estructurados.
  - **Prueba con Iperf:** se usó para obtener datos del *throughput* de los enlaces, así como para determinar los requisitos del sistema cuando fuera necesaria la colaboración de los nodos monitoreados (enfoque cliente-servidor).

- Varios métodos de visualización de datos fueron conceptualizados e implementados y probaron ser una manera intuitiva de inferir hechos sobre el

comportamiento y estado de la red. Un ejemplo de esto, fueron los mapas de calor de latencia que muestran al mismo tiempo patrones de congestión como áreas con colores calientes (rojos y amarillos), y además muestran periodos de inactividad como espacios en blanco.

Además del diseño y desarrollo del sistema, también se ejecutaron pruebas para determinar el rendimiento, estabilidad y escalabilidad tanto de la aplicación web como de los monitores en escenarios reales de producción incluidos casos de uso de la vida real.

Un monitor de pruebas fue desplegado para probar algunos servicios críticos de REDULA tales como ula.ve, saber.ula.ve y un servidor de RESIDE. Este monitor de pruebas se mantuvo conectado a la nube y recolectando datos constantemente durante periodos de semanas y meses y no mostró problemas de rendimiento o estabilidad.

La aplicación web fue desplegada en un servidor de pruebas, la integración y prueba constante de nuevas características en este entorno de producción fue crucial en el desarrollo del *workflow* de integración de pruebas, la aplicación web está disponible en el siguiente URL: http://www.octopusmon.it/octopusmonitor.

El esquema de sincronización de datos a través de la nube usando el API de Dropbox probó ser una manera confiable de transferir datos entre los monitores y la aplicación web. A pesar de que existe un límite de peticiones diarias que puede hacerse al API por usuario, este nunca fue alcanzado incluso con las máximas frecuencias de sincronización; se observó un máximo de 3602 peticiones por día, cada monitor conectado a la nube usará unas 3000 peticiones diarias. Al momento de escribir este documento, se han sincronizado mas de 1161533 trazas usando este mecanismo, cada una de ellas representando el resultado de la ejecución de una prueba.

Se realizaron pruebas de rendimiento al sistema para aproximar la cantidad de usuarios concurrentes que este sería capaz de atender en el hardware disponible. Descubrimos que la aplicación web sería capaz de atender 768 peticiones **dinámicas** por minuto lo que equivale a 1105920 por día, sin embargo solo sería capaz de atender 32 peticiones por minuto dado que estas incluyan la descarga de archivos estáticos, por lo tanto concluimos que para asegurar la escalabilidad del sistema es conveniente delegar esta tarea a otro servidor o un servicio de entrega de contenidos.

A pesar de que hacer análisis sobre los datos recolectados esta fuera del alcance de este trabajo, se observaron algunos patrones sobre la latencia, disponibilidad y estabilidad de los enlaces monitoreados. Por ejemplo, se encontró un patrón claro de menor latencia y mayor estabilidad en el período entre las 12am y las 8am, lo cual corresponde al patrón de uso esperado de la red, encontramos constantes periodos de inactividad para ula.ve al igual que tiempos de respuesta mas largo en comparación con saber.ula.ve, además encontramos que ambos sitios rechazan paquetes del protocolo ICMP, por lo que se monitorearon a través de HTTP, esto solo para nombrar algunos resultados relevantes.

6.2 Recomendaciones

Como trabajos futuros, proponemos la implementación de algoritmos de sincronización que además de introducir datos en la base de datos sean capaces de detectar condiciones críticas a partir de los datos recolectados y alertar proactivamente al usuario emitiendo notificaciones a través de la aplicación web, correo electrónico o SMS. Las alertas podrían ser anexadas a las pruebas y podrían funcionar como parte del *framework* de integración de pruebas.

También sería posible la implementación de nuevos esquemas de sincronización con los monitores que permitan el *streaming* de datos en tiempo real desde los monitores hasta el navegador del usuario, ya sea permitiendo que el servidor web sirva para negociar una conexión directa entre ellos, o que el servidor web sirva como relé a través del cual se pasen los datos.

Apéndice A

Casos de uso de la aplicación web.

A.0.1 Usuario no autenticado

Tabla A.1: Caso de uso – Ver Página Principal

UN-01	*Ver página principal*
Descripción	El usuario desea visitar la página principal del sistema.
Secuencia normal	**Paso** **Acción**
	1 El usuario escribe el URL del sitio y presiona *enter*.
	2 El servidor retorna la página principal.
Excepciones	**Paso** **Acción**
	3 Si el servicio no está disponible el usuario ve un mensaje de error 503.
Postcondición	Pantalla principal del sistema.

Tabla A.2: Caso de uso – Inicio de sesión

UN-02	*Iniciar sesión*
Descripción	El usuario desea ingresar al sistema.
Precondición	Pantalla principal del sistema.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el enlace de *login*.
	2 El servidor muestra el formulario de *login*.
	3 El usuario ingresa sus credenciales.
	4 El usuario hace click en el botón *login*.
Excepciones	**Paso** **Acción**
	3 Si las credenciales son incorrectas el servidor informa del error.
Postcondición	Usuario autenticado y Pantalla "Home".

Tabla A.3: Caso de uso – Registro de usuario

UN-03	*Registro de usuario*
Descripción	El usuario desea registrarse para ingresar al sistema.
Precondición	pantalla principal del sistema.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el enlace de registro.
	2 El servidor muestra el formulario de registro.
	3 El usuario ingresa su nombre de usuario
	4 El usuario ingresa su contraseña
	5 El usuario ingresa su correo electrónico
	6 El usuario hace click en el botón *submit*
	7 El servidor envía un correo electrónico de confirmación
Excepciones	**Paso** **Acción**
	7 Si el nombre de usuario o correo electrónico son incorrectos el servidor informa del error.
	8 Si las contraseñas no coinciden el servidor informa del error.
Postcondición	usuario registrado y correo enviado.

Tabla A.4: Caso de uso – Confirmación de usuario

UN-04	*Confirmación de usuario*
Descripción	El usuario desea confirmar su cuenta.
Precondición	Correo de confirmación enviado (UN-03).
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el enlace de confirmación
	2 El servidor activa la cuenta del usuario
Excepciones	**Paso** **Acción**
	3 Si el código de confirmación está vencido la activación falla.
Postcondición	Cuenta de usuario activa y Pantalla de *Login*.

Tabla A.5: Caso de uso – Recuperar contraseña

UN-05	*Recuperar contraseña*
Descripción	El usuario ha olvidado su contraseña y desea recuperarla.
Precondición	Pantalla de *login*.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en "*forgot password*"
	2 El usuario ingresa su correo electrónico
	3 El servidor envía un correo electrónico de recuperación de contraseña
	4 El usuario hace click en el enlace de recuperación
	5 El usuario ingresa una nueva contraseña
Excepciones	**Paso** **Acción**
	6 Si las contraseñas no coinciden el servidor informa del error.
Postcondición	Contraseña restablecida.

A.0.2 Usuario

Tabla A.6: Caso de uso – Cambiar contraseña

UA-01	*Cambiar contraseña*
Descripción	El usuario desea cambiar su contraseña.
Precondición	Pantalla de cuenta de usuario.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en "cambiar contraseña"
	2 El usuario ingresa su nueva contraseña
	3 El usuario confirma su nueva contraseña
	4 El servidor actualiza la contraseña
Excepciones	**Paso** **Acción**
	6 Si las contraseñas no coinciden el servidor informa del error.
Postcondición	Contraseña actualizada.

Tabla A.7: Caso de uso – Vincular cuenta de Dropbox

UA-02	*Vincular cuenta de Dropbox*
Descripción	El usuario desea vincular su cuenta de Dropbox con Octopus Monitor.
Precondición	Pantalla de cuenta de usuario.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en "vincular cuenta de Dropbox" 2 El servidor retorna un enlace a Dropbox y un formulario 3 El usuario hace click en el enlace 4 El usuario ingresa a Dropbox 5 El usuario da permisos a la aplicación de Octopus Monitor 6 Dropbox muestra un código al usuario 7 El usuario ingresa el código en el formulario 8 El servidor envía el código a Dropbox 9 Dropbox retorna el código de acceso del usuario 10 El servidor actualiza el perfil del usuario
Excepciones	**Paso** **Acción** 5 Si el usuario no da permisos a la aplicación el flujo termina.
Postcondición	Cuenta de Dropbox vinculada.

Tabla A.8: Caso de uso – Ver monitores

UA-03	*Ver monitores*
Descripción	El usuario desea ver la lista de sus monitores.
Precondición	Pantalla "Home".
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en "*monitors*" en el menú lateral 2 El servidor retorna la lista de los monitores
Postcondición	Pantalla de monitores.

Tabla A.9: Caso de uso – Crear monitor

UA-04	*Crear monitor*
Descripción	El usuario desea crear un monitor.
Precondición	Pantalla de monitores.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el botón de agregar monitor
	2 El servidor envía el formulario de datos básicos
	3 El usuario ingresa los datos básicos del monitor
	3 El usuario hace click en "*next*"
	4 El servidor envía el formulario de localización
	5 El usuario ingresa los datos de localización
	6 El usuario hace click en "*next*"
Excepciones	**Paso** **Acción**
	2 Si los datos no son validos el servidor informa del error.
Postcondición	Pantalla de instrucciones de instalación del monitor.

Tabla A.10: Caso de uso – Ver monitor

UA-05	*Ver monitor*
Descripción	El usuario desea ver los detalles de un monitor.
Precondición	Pantalla de monitores.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el enlace "*view*" de un monitor
Postcondición	Pantalla del monitor.

Tabla A.11: Caso de uso – Editar monitor

UA-05	*Editar monitor*
Descripción	El usuario desea editar los detalles de un monitor.
Precondición	Pantalla del monitor.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el botón "*edit*" del monitor
	2 El servidor envía el formulario de edición
	3 El usuario ingresa los datos
	4 El usuario hace click en el botón "*done*"
Excepciones	**Paso** **Acción**
	3 Si los datos no son validos el servidor informa del error.
Postcondición	Pantalla del monitor.

Tabla A.12: Caso de uso – Eliminar monitor

UA-06	*Eliminar monitor*
Descripción	El usuario desea eliminar un monitor.
Precondición	Pantalla del monitor.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el botón "*delete*" del monitor
	2 El servidor envía un dialogo de confirmación
	3 El usuario confirma que desea continuar
	4 El servidor elimina el monitor y todos las entradas relacionadas en el base de datos
Excepciones	**Paso** **Acción**
	3 Si el usuario no confirma el flujo termina.
Postcondición	Pantalla de monitores.

Tabla A.13: Caso de uso – Ver Tentáculos

UA-07	*Ver Tentáculos*
Descripción	El usuario desea ver la lista de tentáculos para un monitor.
Precondición	Pantalla del monitor.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el enlace "*tentacles*" del sub-menú lateral del monitor
Postcondición	Pantalla de Tentáculos.

Tabla A.14: Caso de uso – Registrar tentáculo

UA-08	*Registrar tentáculo*
Descripción	El usuario desea registrar un tentáculo.
Precondición	Pantalla de tentáculos.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el botón "*register tentacle*" 2 El servidor envía el formulario de registro de tentáculo 3 El usuario ingresa los datos del tentáculo 4 El usuario hace click en "submit"
Excepciones	**Paso** **Acción** 3 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de tentáculos.

Tabla A.15: Caso de uso – Editar tentáculo

UA-09	*Editar tentáculo*
Descripción	El usuario desea editar un tentáculo.
Precondición	Pantalla de tentáculos.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el ícono de lápiz del tentáculo en la lista.
	2 El servidor envía el formulario de edición de tentáculo
	3 El usuario ingresa los datos del tentáculo
	4 El usuario hace click en "*submit*"
Excepciones	**Paso** **Acción**
	3 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de tentáculos.

Tabla A.16: Caso de uso – Editar localización del tentáculo

UA-10	*Editar tentáculo*
Descripción	El usuario desea editar la localización un tentáculo.
Precondición	Pantalla de tentáculos.
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el ícono del marcador de mapa del tentáculo en la lista.
	2 El servidor envía el formulario de localización de tentáculo
	3 El usuario ingresa los datos de localización tentáculo
	4 El usuario hace click en "*submit*"
Excepciones	**Paso** **Acción**
	3 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de tentáculos.

Tabla A.17: Caso de uso – Ver plan de monitoreo

UA-11	*Ver plan de monitoreo*
Descripción	El usuario desea ver el plan de plan de monitoreo.
Precondición	Pantalla del monitor.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en "*tests*" en el menú lateral de navegación. 2 El servidor envía un menú para seleccionar el tipo de prueba a observar 3 El usuario selecciona un tipo de prueba y hace click en "*view test plan*"
Postcondición	Pantalla de Plan de Monitoreo para una prueba.

Tabla A.18: Caso de uso – Planificar prueba periódica

UA-12	*Planificar prueba periódica*
Descripción	El usuario desea planificar una prueba de tipo periódico.
Precondición	Pantalla de plan de Monitoreo.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el botón "*schedule periodic test*". 2 El servidor envía una lista de pruebas disponibles para el monitor 3 El usuario selecciona una prueba de la lista. 4 El servidor envía el formulario de planificación y configuración de prueba 5 El usuario ingresa la fecha inicial, final y el intervalo entre pruebas 6 El usuario ingresa los parámetros de la prueba 7 El usuario hace click en el botón "submit" 8 El servidor guarda la prueba y sube un mensaje al buzón del monitor
Excepciones	**Paso** **Acción** 5 Si la fechas son inválidas o el intervalo no es un número el servidor informa del error. 6 Si alguno de los parámetros es invalido el servidor informa del error
Postcondición	Pantalla de detalles de la prueba.

Tabla A.19: Caso de uso – Planificar prueba de una sola ejecución

UA-13	*Planificar prueba*
Descripción	El usuario desea planificar una prueba para un momento específico.
Precondición	Pantalla de plan de Monitoreo.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el botón "*schedule test*". 2 El servidor envía una lista de pruebas disponibles para el monitor 3 El usuario selecciona una prueba de la lista. 4 El servidor envía el formulario de planificación y configuración de prueba 5 El usuario ingresa la fecha de ejecución de la prueba 6 El usuario ingresa los parámetros de la prueba 7 El usuario hace click en el botón "*submit*" 8 El servidor guarda la prueba y sube un mensaje al buzón del monitor
Excepciones	**Paso** **Acción** 5 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de detalles de la prueba.

Tabla A.20: Caso de uso – Editar prueba

UA-14	*Editar prueba*
Descripción	El usuario desea modificar los parámetros de una prueba planificada.
Precondición	Pantalla de detalles de la prueba.
Secuencia normal	**Paso** **Acción** 1 El usuario edita los campos que desea cambiar o re-definir. 2 El usuario hace click en el botón "*Submit*" 3 El servidor guarda la prueba y sube un mensaje al buzón del monitor
Excepciones	**Paso** **Acción** 1 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de detalles de la prueba.

Tabla A.21: Caso de uso – Habilitar/Deshabilitar prueba

UA-15	*Habilitar/Deshabilitar prueba*
Descripción	El usuario desea habilitar o deshabilitar una prueba.
Precondición	Pantalla de detalles de la prueba.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el enlace *enable/disable test*. 2 El servidor cambia el estado de la prueba y sube un mensaje al buzón del monitor
Postcondición	Pantalla de detalles de la prueba.

Tabla A.22: Caso de uso – Seleccionar visualización

UA-16	*Seleccionar visualización*
Descripción	El usuario desea ver una visualización de datos para un monitor.
Precondición	Pantalla del monitor.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en "*results*" en el menú lateral de navegación. 2 El servidor filtra las visualizaciones según las pruebas que se han activo para el monitor y envía la lista 3 El usuario busca una visualización en la lista y hace click sobre el enlace correspondiente
Postcondición	Pantalla de visualización de resultados.

Tabla A.23: Caso de uso – Computar visualización

UA-17	*Computar visualización*
Descripción	El usuario desea computar una visualización.
Precondición	Pantalla de visualización de resultados.
Secuencia normal	**Paso** **Acción** 1 El usuario introduce los parámetros de configuración de la visualización en el formulario. 2 El usuario hace click en el botón "*PLOT*" 3 El servidor valida el formulario y envía un URL directo de cómputo de visualización 4 El navegador solicita el URL en segundo plano 5 El servidor busca en el caché o computa la visualización 6 El servidor retorna la visualización 7 El navegador enmarca la visualización en la interfaz
Excepciones	**Paso** **Acción** 1 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de visualización de resultados.

Tabla A.24: Caso de uso – Re-computar visualización

UA-18	*Re-computar visualización*
Descripción	El usuario desea computar una visualización para incluir nuevos datos sin pasar por el caché.
Precondición	Pantalla de visualización de resultados.
Secuencia normal	**Paso** **Acción** 1 El usuario introduce los parámetros de configuración de la visualización en el formulario. 2 El usuario hace click en el botón "*Re-compute*" 3 El servidor valida el formulario y envía un URL directo de re-cómputo de visualización 4 El navegador solicita el URL en segundo plano 5 El servidor computa la visualización 6 El servidor retorna la visualización 7 El navegador enmarca la visualización en la interfaz
Excepciones	**Paso** **Acción** 1 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de visualización de resultados.

Tabla A.25: Caso de uso – Ver enlace directo de una visualización

UA-19	*Ver enlace directo de una visualización*
Descripción	El usuario desea ver el enlace directo de una visualización para compartir.
Precondición	UA-17, UA-18.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el botón "*View direct link*" 2 El navegador muestra el enlace directo de la visualización
Postcondición	Pantalla de visualización de resultados.

Tabla A.26: Caso de uso – Ver visualización en nueva pestaña

UA-20	*Ver visualización en nueva pestaña*
Descripción	El usuario desea ver la visualización a tamaño completo en una nueva pestaña.
Precondición	UA-17, UA-18.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el enlace "*View in new tab*" 2 El navegador abre una nueva pestaña con el URL de la visualización
Postcondición	Pantalla de Visualización.

Tabla A.27: Caso de uso – Ver detalles de sincronización

UA-21	*Ver detalles de sincronización*
Descripción	El usuario ver el horario de sincronización y la información sobre las últimas sincronizaciones.
Precondición	Pantalla del monitor
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el enlace "*Sync*" en el sub-menú del monitor 2 El servidor devuelve la Pantalla de sincronización
Postcondición	Pantalla de sincronización

Tabla A.28: Caso de uso – Sincronizar monitor

UA-22	*Sincronizar monitor*
Descripción	El usuario desea sincronizar un monitor inmediatamente.
Precondición	Pantalla de sincronización
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el botón "*Sync now*" 2 El servidor encola la sincronización del monitor 3 El servidor devuelve la pantalla de sincronización con un mensaje indicando al usuario que espere.
Postcondición	Pantalla de sincronización

Tabla A.29: Caso de uso – Definir horario de sincronización

UA-23	*Definir horario de sincronización*
Descripción	El usuario desea definir el horario de sincronización de un monitor.
Precondición	Pantalla de sincronización
Secuencia normal	**Paso** **Acción**
	1 El usuario selecciona un esquema de sincronización
	2 El navegador muestra el formulario para el esquema seleccionado
	3 El usuario introduce los datos.
	4 El usuario hace click en "*Submit*".
	5 El servidor actualiza la base de datos.
	6 El planificador detecta los cambios y ajusta el tiempo de la próxima sincronización del monitor
Excepciones	**Paso** **Acción**
	3 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla de sincronización

Tabla A.30: Caso de uso – Crear reporte

UA-24	*Crear reporte*
Descripción	El usuario desea crear un nuevo reporte.
Precondición	Pantalla de reportes
Secuencia normal	**Paso** **Acción**
	1 El usuario hace click en el botón "Create Report"
	2 El servidor envía un formulario de crear reporte
	3 El usuario introduce el nombre del reporte.
	4 El usuario introduce el cuerpo del reporte
Excepciones	**Paso** **Acción**
	3 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla del reporte

Tabla A.31: Caso de uso – Editar reporte

UA-25	*Editar reporte*
Descripción	El usuario desea editar un reporte existente.
Precondición	Pantalla del reporte
Secuencia normal	**Paso** **Acción**
	1 El usuario hacer click en el botón "*editar*"
	2 El servidor envía el formulario de edición de reporte
	3 El usuario introduce los cambios al reporte
	4 El servidor guarda los cambios
Excepciones	**Paso** **Acción**
	2 Si los datos no son válidos el servidor informa del error.
Postcondición	Pantalla del reporte

Tabla A.32: Caso de uso – Cambiar privacidad del reporte

UA-26	*Cambiar privacidad del reporte*
Descripción	El usuario desea hacer un reporte público o privado.
Precondición	Pantalla del reporte
Secuencia normal	**Paso** **Acción**
	1 El usuario hacer click en el botón "*Public*" o "*Private*"
	2 El navegador envía una petición asíncrona al servidor
	3 El servidor guarda los cambios y devuelve una respuesta exitosa
	4 El navegador actualiza la interfaz mostrando el nuevo estado de privacidad
	5 El servidor guarda los cambios
Excepciones	**Paso** **Acción**
	2 Si el servidor no responde se alerta al usuario.
Postcondición	Pantalla del reporte

Tabla A.33: Caso de uso – Agregar visualización a un reporte

UA-27	*Agregar visualización a un reporte*
Descripción	El usuario desea agregar una visualización a un reporte.
Precondición	Pantalla del reporte
Secuencia normal	**Paso** **Acción** 1 El usuario hacer click en el botón "*Add Result*" 2 El usuario computa una visualización (UA-17) 3 El usuario hace click en el botón "*Add to Report*" 4 El navegador muestra un formulario para agregar comentarios al reporte 5 El usuario introduce comentarios sobre la visualización 6 El usuario somete el formulario 7 El servidor agrega la visualización al reporte
Excepciones	**Paso** **Acción** 5 Si los datos introducidos son inválidos el servidor alerta el problema.
Postcondición	Pantalla de Visualización

Tabla A.34: Caso de uso – Editar visualización de un reporte

UA-28	*Editar visualización de un reporte*
Descripción	El usuario desea editar los comentarios de una visualización en un reporte.
Precondición	Pantalla del reporte
Secuencia normal	**Paso** **Acción** 1 El usuario hacer click en los comentarios del reporte 2 El navegador muestra un formulario para editar los comentarios 3 El usuario hace los cambios al texto 4 El hace click en el botón "*accept*" 5 El servidor actualiza el reporte y devuelve una respuesta exitosa 6 El navegador actualiza la interfaz
Excepciones	**Paso** **Acción** 3 Si los datos introducidos son inválidos el servidor alerta el problema.
Postcondición	Pantalla del Reporte

Tabla A.35: Caso de uso – Eliminar visualización de un reporte

UA-29	*Eliminar visualización de un reporte*
Descripción	El usuario desea eliminar una visualización en un reporte.
Precondición	Pantalla del reporte
Secuencia normal	**Paso** **Acción** 1 El usuario hacer click en el enlace "*remove*" correspondiente a la visualización 2 El servidor actualiza el reporte 3 El servidor envía el reporte sin la visualización eliminada
Postcondición	Pantalla del Reporte

Tabla A.36: Caso de uso – Eliminar reporte

UA-30	*Eliminar reporte*
Descripción	El usuario desea eliminar un reporte.
Precondición	Pantalla del reporte
Secuencia normal	**Paso** **Acción**
	1 El usuario hacer click en el botón "*delete*"
	2 El servidor elimina el reporte
Postcondición	Pantalla de Reportes

Apéndice B

Casos de uso del monitor de red

Tabla B.1: Caso de uso – Iniciar monitor

TM-01	*Iniciar monitor*
Descripción	El usuario desea iniciar el monitor de red TPS.
Secuencia normal	**Paso** **Acción**
	1 El usuario invoca el cliente con el comando "start".
	2 El cliente crea el proceso daemon y retorna.
Excepciones	**Paso** **Acción**
	3 Si el archivo `.pid` existe entonces el cliente muestra un mensaje de error indicando que el monitor ya se esta ejecutando.

Tabla B.2: Caso de uso – Detener monitor

TM-02	*Detener monitor*	
Descripción	El usuario desea detener el monitor que se esta ejecutando en modo daemon.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario invoca el cliente con el comando "stop".
	2	El cliente abre el archivo `.pid` y envía la señal SIGTERM al proceso daemon.
	3	El monitor maneja la excepción deteniendo su ejecución.
Excepciones	**Paso**	**Acción**
	4	Si el archivo `.pid` no existe, el cliente informa al usuario que el monitor no se esta ejecutando.

Tabla B.3: Caso de uso – Reiniciar monitor

TM-03	*Reiniciar monitor*	
Descripción	El usuario desea reiniciar el monitor que se esta ejecutando en modo daemon.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario invoca el cliente con el comando "restart".
	2	El cliente abre el archivo `.pid` y envía la señal SIGTERM al proceso daemon.
	3	El monitor maneja la excepción deteniendo su ejecución.
	4	El cliente crea el proceso daemon y retorna.
Excepciones	**Paso**	**Acción**
	5	Si el archivo `.pid` no existe, el cliente informa al usuario que el monitor no se esta ejecutando.

Tabla B.4: Caso de uso – Ver Plan de Monitoreo

TM-04	*Ver Plan de Monitoreo*	
Descripción	El usuario desea ver el plan de ejecución del monitor.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario invoca el cliente con el comando "jobs".
	2	El cliente abre la base de datos y retira la información.
	3	El cliente muestra por pantalla las tareas planificadas y los enlaces monitoreados y retorna.
Excepciones	**Paso**	**Acción**
	5	Si la base de datos no se ha creado muestra una lista vacía.

Bibliografía

[1] J. Saldana, A. Arcia-Moret, B. Braem, E. Pietrosemoli, A. Sathiaseelan, and M. Zennaro, "Alternative Network Deployments. Taxonomy, characterization, technologies and architectures.," Active Internet-Draft (gaia RG) , November 2015.

[2] A. Arcia-Moret, J. Gomez, and A. Sathiaseelan, "Octopus: A zero-cost architecture for stream network monitoring," in *ACM DEV*, (London, UK), pp. 1 – 2, December 2015.

[3] J. F. Kurose and K. W. Ross, *Computer Networking. A Top-Down Approach.* Pearson, 2013.

[4] C. Mikeka, M. Thodi, J. Mlatho, J. Pinifolo, D. Kondwani, L. Momba, M. Zennaro, and A. Arcia-Moret, "Preliminary performance assessment of tv white spaces technology for broadband communication in malawi," in *Humanitarian Technology: Science, Systems and Global Impact (HumTech2014)*, (Boston, USA), 2014.

[5] M. Porcar, A. Arcia, J. Gomez, E. Velasquez, and M. Hernandez, "Malawinet network monitor.," tech. rep., Universidad de los Andes, 2015.

[6] B. Vahl, T. Luque, F.H.and Huhn, and C. Sengul, "Network monitoring and debugging through measurement visualization," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium*, (San Francisco, CA), pp. 1 – 6, IEEE, 2012.

[7] S. Sonntag, J. Manner, and L. Schulte, "Netradar-measuring the wireless world," in *Modeling & Optimization in Mobile, Ad Hoc & Wireless Networks (WiOpt), 2013 11th International Symposium on*, pp. 29–34, IEEE, 2013.

[8] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato, "Bismark: a testbed for deploying measurements and applications in broadband access networks," in *2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 383–394, 2014.

[9] RIPE, NCC, "Ripe atlas." [Página web en línea]. Disponible en : https://atlas.ripe.net, 2010.

[10] Solarwinds Cloud, "Pingdom - website monitoring." [Página web en línea] Disponible en https://www.pingdom.com/, 2015.

[11] Uptime Robot Bilisim A.S., "Uptime robot." [Página web en línea]. Disponible en : https://uptimerobot.com/, 2015.

[12] I. Grigorik, *High Performance Browser Networking*. O'Reilly, 2013.

[13] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, p. 40, 2011.

[14] Silver Matrix LLC, "Dslreports home: Broadband isp reviews news tools and forums." [Página web en línea]. Disponible en : http://www.dslreports.com/speedtest/, 2015.

[15] NetConnex Ltd, "Thinkbroadband :: Uk broadband speed test." [Página web en línea]. Disponible en : http://www.thinkbroadband.com/speedtest.html, 2015.

[16] S. D. Strowes, "Passively measuring tcp round-trip times," *Communications of the ACM*, vol. 56, no. 10, pp. 57–64, 2013.

[17] C. Demichelis and P. Chimento, "RFC3393: IP Packet Delay Variation Metric for IP Performance Metrics," RFC 3393, RFC Editor, November 2002.

[18] Ookla, "Speedtest.net by ookla." [Página web en línea]. Disponible en : http://www.speedtest.net/, 2014.

[19] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate, "End-to-end wan service availability," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 2, pp. 300–313, 2003.

[20] Pervasifm, "Data visualization." [Página web en línea]. Disponible en : http://www.pervasif.com/index.php/news-a-event/capabilities/data-visualization, 2012.

[21] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[22] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu, "Cloud storage as the infrastructure of cloud computing," in *Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on*, pp. 380–383, IEEE, 2010.

[23] E. Bascón Pantoja, "El patrón de diseño modelo-vista-controlador (mvc) y su implementación en java swing," *Acta Nova*, vol. 2, no. 4, p. 493, 2004.

[24] Django Software Foundation, "The web framework for perfectionists with deadlines." [Página web en línea]. Disponible en : https://www.djangoproject.com/, 2015.

[25] A. Solem, "Celery: Distributed task queue." [Página web en línea]. Disponible en : http://www.celeryproject.org/, 2015.

[26] S. Sanfilippo and P. Noordhuis, "Redis." [Página web en línea]. Disponible en : http://redis.io/, 2015.

[27] Oracle, "Mysql 5.7 :: Reference manual 1.3.1 what is mysql?." [Página web en línea]. Disponible en : http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html, 2015.

[28] S. W. Ambler, "Mapping objects to relational databases: O/r mapping in detail." [Página web en línea]. Disponible en : http://www.agiledata.org/essays/mappingObjects.html, 2013.

[29] Dropbox, Inc, "Acerca de dropbox." [Página web en línea]. Disponible en : https://www.dropbox.com/about, 2015.

[30] Highsoft AS, "Interactive javascript charts for your webpage — highcharts." [Página web en línea]. Disponible en : http://www.highcharts.com/, 2015.

[31] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye, "Ip geolocation databases: Unreliable?," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 53–56, 2011.

[32] A. Grönholm, "Advanced python scheduler - apscheduler 3.1.0.dev1 documentation." [Página web en línea]. Disponible en : https://apscheduler.readthedocs.org/en/latest/, 2015.

[33] D. Vega, L. Cerda-Alabern, L. Navarro, and R. Meseguer, "Topology patterns of a community network: Guifi. net," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, pp. 612–619, IEEE, 2012.

[34] Monitis, "Online visual traceroute." [Página web en línea]. Disponible en : http://www.monitis.com/traceroute/, 2010.

[35] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: illuminating the edge network," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 246–259, ACM, 2010.

[36] J. Nielsen, *Usability engineering*. Elsevier, 1994.

[37] The Apache Software Foundation, "Apache jmeter." [Página web en línea]. Disponible en : http://jmeter.apache.org, 2014.

[38] B. Finney, "Standard daemon process library ," Tech. Rep. 3143, January 2009.