



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

Octubre 2015

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, 43 páginas

Resumen: En el presente proyecto se plantea el desarrollo de un sistema de monitoreo distribuido de enlaces críticos de redes a través de un servicio web centralizado. Este servicio además pretende hacer 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 solución centralizada, de muy bajo costo y con componentes de hardware que puedan estar desatendidos.

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

Índice

1	Introducción	1
1.1	Antecedentes	1
1.2	Planteamiento del Problema	3
1.3	Justificación	4
1.4	Objetivos	4
1.4.1	Objetivos Generales	4
1.4.2	Objetivos Específicos	4
1.5	Metodología	5
1.6	Alcance	6
1.7	Estructura del Documento	6
2	Marco Teórico	7
2.1	Monitoreo de Redes	7
2.2	Principio Fin-a-Fin	7
2.3	SMTP	7
2.4	Métricas de calidad de un enlace	7
2.4.1	Latencia	7
2.4.2	Tiempo de ida y vuelta	8
2.4.3	Perdida de paquetes	9
2.4.4	Jitter	9
2.4.5	Throughput	10
2.4.6	Uso del ancho de banda	11
2.4.7	Disponibilidad	11

2.4.8	Bufferbloat	12
2.5	Big Data	13
2.6	Data Visualization	13
2.7	Computación en la nube	13
2.7.1	IaaS	13
2.7.2	PaaS	13
2.7.3	SaaS	13
2.8	Herramientas usadas para el desarrollo del sistema	13
2.8.1	Modelo Vista Controlador (MVC)	13
2.8.2	Django	15
2.8.3	Celery	16
2.8.4	Redis	18
2.8.5	Bases de Datos	18
2.8.6	Json (JavaScript Object Notation)	20
2.8.7	Dropbox	20
2.8.8	Diseño web adaptable	22
2.8.9	Highcharts	22
2.8.10	Google Maps	22
2.8.11	freegeoip	22
2.8.12	Ajax	22
2.8.13	APScheduler	22
2.8.14	Ping	22
2.8.15	Requests	22
2.8.16	Traceroute	22
2.9	Estado del Arte	22
2.9.1	Tabla comparativa del estado del arte	22
3	Monitor de Red "Octopus Tentacle"	23
3.1	Diseño del Monitor	24
3.1.1	Arquitectura	24
3.1.2	Diagrama de actividades	25
3.2	Componentes	27

3.2.1	Cliente	27
3.2.2	Hilo Principal	27
3.2.3	Manejador de plugins	27
3.2.4	Planificador	28
3.2.5	Almacenamiento Compartido	28
3.3	Estructura del archivo de configuración	29
3.4	Pruebas Implementadas	31
3.4.1	Ping	32
3.4.2	Httping	33
3.4.3	Traceroute	35
3.5	Casos de uso	36
4	Aplicación Web "Optopus Monitor"	38
4.1	Diseño del sistema	39
4.1.1	Arquitectura	39
4.1.2	Estructura de la base de datos	39
4.1.3	Flujo de navegacion	39
4.1.4	Diseño de Pantallas	39
4.2	Componentes	39
4.2.1	Recolector de datos	39
4.2.2	Generación de visualizaciones	39
4.3	Casos de Uso	39
4.4	Caching de gráficas	39
4.5	Pruebas de Rendimiento	39
5	Framework de integración de pruebas	40
6	Conclusiones y Recomendaciones	41
	Bibliografía	42

Capítulo 1

Introducción

1.1 Antecedentes

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. El benchmarking tiene la desventaja de tener que inyectar tráfico lo cual puede entorpecer el funcionamiento normal de la red ejemplos de herramientas de benchmarking son ping, iperf y traceroute.

El monitoreo pasivo tiene la ventaja de que nos puede dar una muy buena 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 y wireshark; en ambos casos hay que resaltar que es difícil tener una imagen completa de la realidad 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. 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 [1].

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

La red de TVWS (TV White Spaces o Espacios blancos en el espectro radioeléctrico) de Malawi fue implementada en el marco de un proyecto para llevar Internet a áreas rurales en países en vías de desarrollo utilizando soluciones de bajo costo a través de los espacios en blanco en el espectro radioeléctrico específicamente en la banda UHF (siglas del inglés Ultra High Frequency, 'frecuencia ultra alta') [2].

Muchas veces esta red debe dejarse desatendida durante largos periodos de tiempo ya que sus nodos son de difícil acceso o es muy costoso tener a profesionales dedicados que se encarguen de su mantenimiento, este escenario hace evidente la necesidad de una solución de monitoreo de redes a distancia y 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 la red de Malawi se implementó un sistema en dos partes: un monitor de red instalado en la estación base (BS) en Malawi y un servicio web remoto, 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 por sus siglas en inglés) 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 [3].

A pesar de que este sistema recolecta información útil y presenta gráficas muy sencillas de entender, su programación no permite agregar nuevos nodos, esto trae como consecuencia que debe ser modificado manualmente cuando la red se expande, dando lugar a la necesidad de que el servicio web se pueda expandir para dar servicio

a múltiples monitores remotos simultáneamente.

Por estos motivos proponemos la construcción de un sistema de monitoreo a gran escala que llamaremos Octopus Monitor, para hacerlo totalmente configurable y robusto además de agregar una interfaz de configuración vía web que permita manejar usuarios, agregar monitores remotos, agregar nodos y modificar los parámetros de las pruebas, todo esto apoyándonos en un sistema de archivos compartidos a través de la nube.

Mientras que el mayor valor de Malawinet Monitor es la visualización de grandes volúmenes de datos que se pueden obtener a partir de pruebas de bajo impacto de tráfico (ping, traceroute), se han realizado otros trabajos en el área de monitoreo de redes como Bowlmap; 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 inglés). Este sistema tiene una alta flexibilidad ya que permite realizar cambios en sus pruebas existentes así como agregar pruebas totalmente nuevas y a su vez generar las visualizaciones necesarias para el análisis de dicha información, además tiene la ventaja de solo transmitir la información necesaria para cada actualización lo que acelera las peticiones y permite la visualización de data en tiempo real [4].

1.2 Planteamiento del Problema

Las redes de computadoras se componen de un conjunto de nodos interconectados sujetos a numerosos factores que escapan de nuestro control y sobre los que muchas veces no tenemos conocimiento, en otras palabras la red puede llegar a ser impredecible y no ofrece garantías sobre el servicio que ofrece; algunas aplicaciones dependen de una alta disponibilidad y estabilidad por lo que es esencial para un administrador de red tener información del estado de la red, para diagnosticar, solucionar problemas y asegurar la calidad de servicio.

Existen muchos otros ejemplos en los que es importante tener datos del estado de la red como en redes de bajo costo en las que pueden ocurrir largas interrupciones de servicio o para un cliente de un servicio de alojamiento web que desea saber si su sitio web está disponible y que tan rápido responde; plataformas como Pingdom [5] o

UptimeRobot [6] 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 (es decir, las fuentes de información) y la cantidad de datos aumenta a través del tiempo, sumado a esto solo podemos capturar información a partir de los nodos externos de la red, por lo que en la mayoría de los casos no es posible tener una imagen completa de los enlaces a monitorear.

A pesar de que Malawinet Network Monitor podría ofrecer estadísticas de tiempo de ida y vuelta (RTT) y disponibilidad de un enlace solo a partir de las trazas capturadas con Ping, se desea además implementar un marco de trabajo que permita agregar nuevas pruebas automatizadas que ayuden a obtener una imagen más completa de la red.

1.3 Justificacion

1.4 Objetivos

1.4.1 Objetivos Generales

Construir un servicio web de monitoreo de redes de bajo costo para países en vías de desarrollo con almacenamiento de datos en la nube de tipo PaaS que sea de fácil instalación y permita configurar múltiples monitores remotos para ajustarse a cambios en las características de las redes a monitorear y la carencia de personal in sitio.

1.4.2 Objetivos Específicos

- Desarrollar un servicio de monitoreo de bajo costo para países en vías de desarrollo 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 de sistemas de bajo costo y alta disponibilidad en la nube para almacenamiento y transferencia de datos.
- Integrar los distintos subsistemas que conforman el servicio de monitoreo.
- 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.

1.5 Metodología

En este trabajo se seguirá 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 y se lo desarrolla.
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 realizarán cuatro ciclos, el primero corresponde 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 consiste en permitir el monitoreo de una cantidad arbitraria de monitores remotos permitiendo a múltiples usuarios manejar sus monitores remotos desde el servicio web y observar las visualizaciones.

El tercer ciclo corresponde 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. Lidar con aplicaciones que requieren enfoque cliente solo [ping] o cliente-servidor [iperf]).

El cuarto ciclo consiste 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 países en vías de desarrollo.

1.6 Alcance

1.7 Estructura del Documento

Capítulo 2

Marco Teórico

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

2.1 Monitoreo de Redes

2.2 Principio Fin-a-Fin

2.3 SMTP

2.4 Métricas de calidad de un enlace

A continuación se explican las métricas mas comúnmente utilizadas para determinar la calidad de un enlace.

2.4.1 Latencia

La latencia es el tiempo que le toma a un paquete o mensaje viajar desde su origen hasta el punto destino [7]. 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,

la latencia total será la suma de cada uno de los siguientes factores para cada uno de los enrutadores que atraviese el paquete[7]:

- **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.

2.4.2 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 que hace que sea arriesgado establecer esa proposición.

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[8].

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.

2.4.3 Pérdida 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 WiFi), o cuando son desechados por un nodo de la red, los paquetes pueden ser desechados si se determina que están corruptos 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 [8] 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 NOTA: Nosotros alguna vez hablamos de esto, valdría la pena nombrar el RFC que lo explica.

2.4.4 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[1], sin embargo todos los factores mencionados en la sección 2.4.1 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 termino es a veces preferido por ser mas preciso [9]

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[1]

2.4.5 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 [1], el throughput puede compararse al caudal de un río, donde mientras mas ancho sea el río mas agua puede fluir a través de el.

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 wifi y el ISP resulta ser el cuello de botella en la comunicación. En la practica 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.

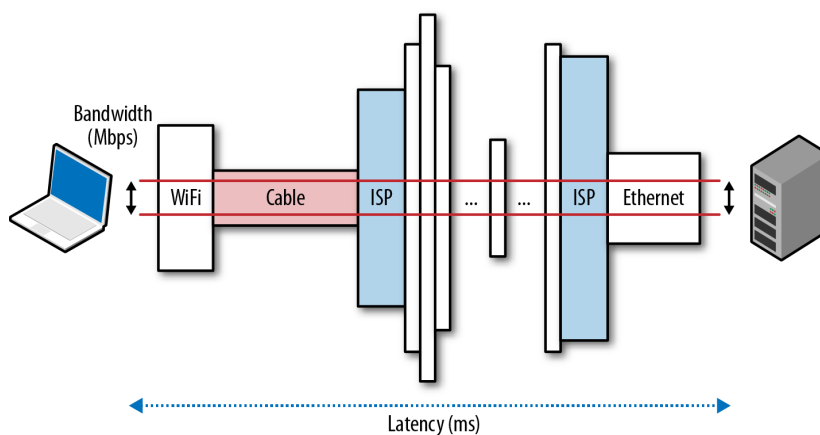


Figura 2.1: Latencia y ancho de banda – Imagen tomada del sitio: <http://chimera.labs.oreilly.com/books/12300000000545/ch01.html>

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[1], 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.net [10] 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.

2.4.6 Uso del ancho de banda

Se le llama uso del ancho de banda a la medida de los datos que fluyen a través de un enlace, generalmente se busca optimizar el uso del ancho de banda para aprovechar al máximo los recursos de red.

Más allá de solo determinar la cantidad de datos o la velocidad del flujo a través de un enlace, también es útil hacer análisis detallados del uso del ancho de banda para descubrir patrones de uso de la red, por ejemplo una universidad podría desear saber que porcentaje del ancho de banda está siendo utilizado para visitar sitios de streaming, a una empresa le gustaría saber cuánto tiempo pasan sus empleados en redes sociales, un analista desea saber cuáles son los sitios web más visitados en una red, etc.

NOTA PARA EL PROFESOR: ayuda, no encontré ninguna cita que me ayudara a definir estos conceptos, sin embargo me parece que es importante incluir esto en el marco teórico.

2.4.7 Disponibilidad

Un servicio está disponible para un cliente cuando dicho cliente puede comunicarse con él, análogamente un servicio no está disponible para un cliente cuando no puede comunicarse con él, ya sea por un problema en el nodo final o en la red [11].

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 [11], 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, y un 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 [11] 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, o cuando la latencia es tal que hace que una comunicación en tiempo real no sea posible.

2.4.8 Bufferbloat

Bufferbloat es la existencia de buffers excesivamente grandes y generalmente llenos presentes en Internet [12]; 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 [12].

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 como [13] [14] 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.

2.5 Big Data

Big Data es la tendencia de recolectar y almacenar cantidades masivas de datos de forma constante permitiendo tomar decisiones basadas en el análisis de estos datos. No se ha definido formalmente una cantidad a partir de la cual se esta hablando de Big Data, pero usualmente se relaciona con datos en el orden de los petabytes y exabytes [15].

El concepto de Big Data esta estrechamente relacionado con el concepto de stream computing, que es el procesamiento de flujos de datos continuos que podrían estarse recibiendo indefinidamente, en contraposición al enfoque clásico de

2.6 Data Visualization

2.7 Computación en la nube

2.7.1 IaaS

2.7.2 PaaS

2.7.3 SaaS

2.8 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.

2.8.1 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 [16].

Este patrón de diseño es especialmente popular en el marco de aplicaciones web ya

que su abstracción permite escribir software altamente desacoplado y fácil de mantener y escalar.

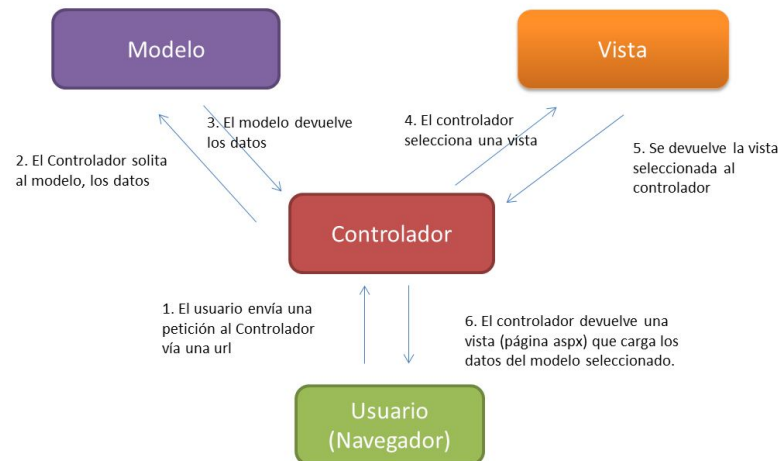


Figura 2.2: Ejemplo de la relación entre los componentes de una aplicación MVC – Imagen tomada del sitio: <http://mind42.com/mindmap/75870b0c-b70c-4e83-9223-67de79330cd3>

Modelo

Según [16] *“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 [16] .

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 caldera es posible tener una vista que dibuje gráficamente el nivel de la caldera y un termómetro con su temperatura 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 modelo anterior de esta forma sería posible monitorear en tiempo real el estado de la caldera a partir de sensores que mantengan actualizado el modelo.

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 línea que separa el controlador y la vista de forma distinta, por ejemplo en algunas implementaciones el controlador se encarga tanto de actualizar el modelo y las vistas como de responder a los mensajes externos, es decir que el controlador es el encargado de ejecutar toda la lógica, y la vista meramente contiene la presentación de los datos, en otras implementaciones como veremos en la próxima sección la vista es la encargada tanto de seleccionar los datos que van a mostrarse así como de desplegar la presentación, esta última es una variación de MVC a veces llamado MTV (Modelo-Template-Vista)

2.8.2 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 "baterías 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; de esta manera el desarrollador puede concentrarse en escribir la lógica que es específica a su aplicación y dejar que Django maneje los aspectos repetitivos y muchas veces tediosos de la pila de desarrollo web.

Django está 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 los datos son presentados.

Cuando Django recibe una petición esta 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 de 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 (llamada template) para generar la respuesta que será devuelta, en la figura 2.3 puede verse el ciclo de petición-respuesta tal como se ha descrito.

Las respuestas generadas por Django pueden ser tanto páginas HTML con CSS y Javascript pensadas para la interacción con el usuario así como respuestas en formato json o xml para la construcción de APIs que pueden ser usados para la comunicación máquina-máquina, esto es especialmente útil cuando se desea desarrollar aplicaciones en otras plataformas como Android o iOS que también pueden ser manejadas por el mismo servidor.

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 enorme comunidad de colaboradores y amplia documentación.

2.8.3 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 más tarde. Celery en realidad no implementa la mayoría de sus

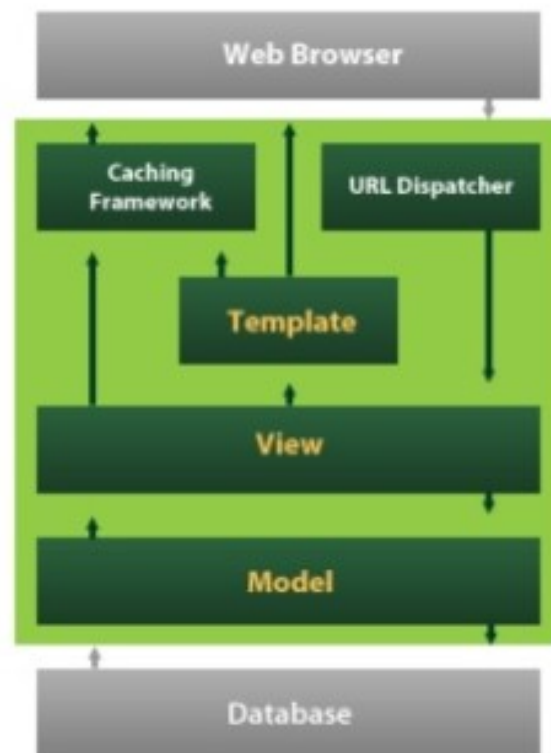


Figura 2.3: Ciclo de petición-respuesta de Django

componentes, sino que define un protocolo de comunicación entre una serie de componentes (también llamados micro-servicios) que son:

- Bróker de mensajería: el message broker es el encargado almacenar las tareas que deben ser ejecutadas, el message broker consiste de una o mas colas de prioridad, de esta manera es posible controlar finamente que tareas deben ser ejecutadas con mayor urgencia, y por que ejecutores.
- Planificador: el planificador es el encargado de encolar tareas previamente planificadas para que sean ejecutadas en algún momento específico, el planificador no puede asegurar que las tareas sean ejecutadas exactamente en el tiempo planificado ya que el tiempo de planificador depende del tamaño de la cola y de las tareas que ya estén siendo ejecutadas por los ejecutores.

- Workers (Ejecutores): los ejecutores son los encargados de ejecutar las tareas; estos se subscriben a una o mas colas y consumen los mensajes según la prioridad establecida en la cola finalmente escriben los resultados obtenidos en un result backend
- Result Backend: sirve como un almacen donde se guardan los resultados de las tareas ejecutadas durante un periodo de tiempo dado, esto es especialmente de esta manera es posible consultar los resultados obtenidos por un tarea

Uno de los problemas comunes en el desarrollo de aplicaciones web es la ejecución de tareas largas o altamente bloqueantes que mantienen ocupado al servidor web durante periodos prolongados y peor aun mantienen al usuario final esperando, para evitar este escenario el servidor web debe tener una forma de delegar este tipo de tareas para ser ejecutadas después y responder inmediatamente.

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

2.8.4 Redis

Redis es un almacén de estructuras de datos en memoria que puede ser usado como base de datos, cache o bróker de mensajería; se caracteriza por su excelente velocidad de respuesta ya que todos sus datos están en memoria, sin embargo sus datos no persisten en caso de fallas o errores inesperados. Redis puede ser usado tanto como bróker de mensajes o como result backend en Redis.

2.8.5 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 [17] 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.

Mysql

Mysql es un sistema manejador de bases de datos relacionales (RDBMS por sus siglas en ingles) de codigo 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 disponible. Mysql posee características para ejecutarse en clusters de maquinas junto con un motor de replicacion para obtener una alta escalabilidad [17].

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 [18] 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 util tener una interfaz que transforme los datos entre estos tipos de 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 esta capa de abstracción permite desacoplar el código de la aplicación de los detalles específicos de cada RDBMS.

2.8.6 Json (JavaScript Object Notation)

Json (JavaScript Object Notation) o notación de objetos javascript 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 máquinas y más ligero que XML por lo que se ha popularizado para el desarrollo de APIs.

Json soporta dos tipos de estructuras de datos:

1. Colecciones de pares {nombre,valor} comparable a un diccionario o tablashash.
2. Listas ordenadas de valores, similar a las listas o vectores que existen en virtualmente cualquier lenguaje de programación

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

Los tipos de datos soportados por json son:

- Number: números flotantes de doble precisión en formato Javascript
- String: cadenas de caracteres unicode encerradas en dobles comillas
- Boolean: true o false
- Arreglo: secuencia de valores ordenados
- Objeto: colección no ordenada de pares {nombre,valor}
- null: nulo o vacío

2.8.7 Dropbox

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

Dropbox es usado por aproximadamente 400 millones de personas y 100000 organizaciones[19] 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 cierta cuota de almacenamiento a partir del cual el pago escala según la cantidad de almacenamiento necesario.

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 la plataforma de otra aplicación (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.

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

2.8.8 Diseño web adaptable

2.8.9 Highcharts

2.8.10 Google Maps

2.8.11 freegeoip

2.8.12 Ajax

2.8.13 APScheduler

2.8.14 Ping

2.8.15 Requests

2.8.16 Traceroute

2.9 Estado del Arte

2.9.1 Tabla comparativa del estado del arte

Capítulo 3

Monitor de Red "Octopus Tentacle"

Octopus Tentacle (OT) es un monitor de red ligero, diseñado para ajustarse a las limitaciones de sistemas de bajo costo y con la intención de ser desplegado en las redes que se desea monitorear y ser dejado desatendido durante periodos arbitrarios de tiempo.

OT ejecuta pruebas en intervalos regulares, recogiendo datos sobre distintas métricas, los resultados de las pruebas son guardados en un espacio de almacenamiento compartido en la nube para que la aplicación web sea capaz de retirar estos datos posteriormente.

Uno de los objetivos principales de Octopus Tentacle es ejecutar las pruebas en el momento preciso, ya que el posterior análisis de los datos exige que las muestras se tomen con un patrón regular y conocido, para esto se hace uso de un planificador, cada prueba se ejecuta en su propio hilo, siendo posible que varias pruebas se ejecuten simultáneamente.

El monitor OT posee una arquitectura basada en plugins que hace posible agregar nuevas pruebas en tiempo de ejecución, sin necesidad de reiniciar el monitor, eliminando potenciales interrupciones a pruebas que ya se están ejecutando. Esto también implica que no importa cuantas pruebas estén disponibles, el monitor solo tiene en memoria aquellas que son parte del conjunto de pruebas a ejecutar.

El comportamiento del monitor viene dado por un archivo de configuración en formato json, este archivo se aloja en el almacenamiento compartido en la nube y funge de interfaz entre los monitores remotos y la aplicación web. OT revisa regularmente este archivo y lo compara con su estado interno haciendo los cambios que sean necesarios.

3.1 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 resultados 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 la manera mas fiel posible la planificación dada.
- **Ejecutable en equipos de bajo costo** el monitor debe incluir el código mínimo para su funcionamiento y tener un uso muy bajo de memoria.
- **Configuración remota** el monitor debe tener algún protocolo para actualizar su estado interno a partir de cambios hechos en la aplicación web.
- **Bitacora** ya que el monitor se ejecuta como un proceso daemon, se desea tener una bitácora donde se puedan leer mensajes sobre ciertos eventos importantes del monitor.

3.1.1 Arquitectura

Como se puede observar en la figura 3.1, Octopus Tentacle tiene una arquitectura basada en componentes altamente desacoplados con una interfaz bien definida para facilitar el desarrollo de la aplicación.

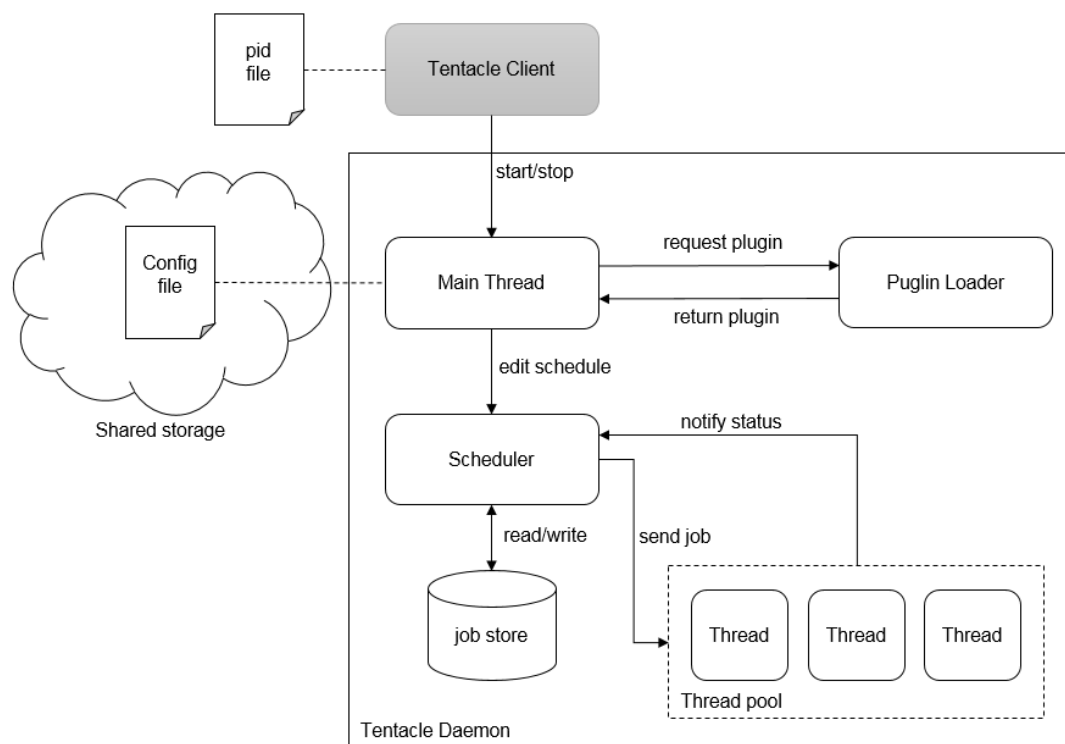


Figura 3.1: Diagrama de Componentes Octopus Tentacle

3.1.2 Diagrama de actividades

La ejecución del monitor Tentacle comienza en el cliente, el cliente es el encargado de instanciar el proceso daemon y lo hace a través de un método bien conocido de doble fork. Para asegurarse de no ejecutar múltiples instancias del monitor se revisa un archivo .pid, si el archivo existe significa que el monitor se está ejecutando y el cliente informa del error al usuario.

Después de que el proceso está daemonizado comienza la fase de inicialización, para esto se crea el planificador y se lee el archivo de configuración insertando en el almacén de tareas del planificador cada prueba a ejecutar, para esto primero hay que importar el código de cada prueba a través del manejador de plugins.

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 este pasa a ejecutarse como un subproceso y se encarga

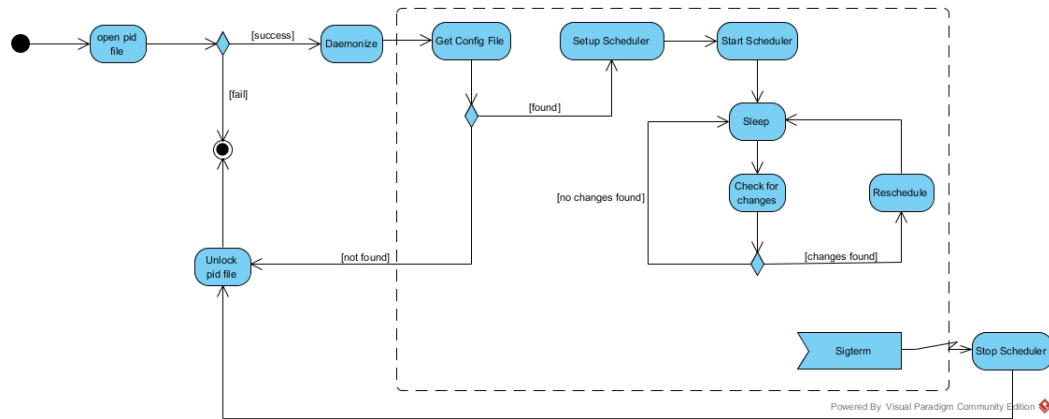


Figura 3.2: Diagram de actividades de Tentacle

de iniciar las tareas en el momento justo. Cuando el planificador inicia una tarea se la pasa al ejecutor en este caso un grupo de subprocesos previamente inicializados, cuando un subproceso termina de ejecutar una tarea informa al planificador, esto con la finalidad de implementar politicas para limitar la cantidad de tareas del mismo tipo que se estan ejecutando simultaneamente.

El hilo principal revisa periódicamente el archivo de configuración y lo compara con el almacen de tareas, existen tres tipos de cambios relevantes: (1) una prueba se ha insertado en la configuracion y debe cargarse a memoria y planificarse (2) una prueba se ha eliminado y debe ser eliminada de la planificacion (3) el intervalo entre pruebas de una prueba ha cambiado y debe replanificarse.

El monitor solo puede detenerse a través de una señal del sistema operativo, en este caso SIGTERM, el cliente utiliza el archivo pid para determinar el id 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 3.2.

3.2 Componentes

3.2.1 Cliente

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

Se puede invocar el cliente con los siguientes comandos:

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)

Tabla 3.1: Comandos del cliente Octopus Tentacle

3.2.2 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 a partir del archivo de configuración y se asegura de mantener el estado del planificador al día en caso de que se observen cambios en el mismo.

3.2.3 Manejador de plugins

Este modulo contiene la lógica para cargar plugins de forma dinámica, cada plugin se identifica a través de un "nombre de modulo" único, si no se encuentra un paquete que coincida con dicho nombre se escribe un mensaje de error en la bitácora y se omite esta entrada.

El único requisito para que un modulo sea un plugin valido es que este implemente una función "run" que sirva como punto de partida para una prueba.

3.2.4 Planificador

Para la implementación del planificador se hizo uso de APScheduler, una biblioteca que permite ejecutar de forma periódica código python; el planificador se puede ejecutar como un subprocesso de modo que es posible agregar, eliminar y re-planificar tareas en cualquier momento desde el hilo principal de la aplicación.

APScheduler consiste de un conjunto de componentes configurables que se pueden extender o re-usar para obtener cualquier comportamiento deseado, a continuación se explica su funcionamiento y como se usaron en el marco de esta aplicación.

Triggers (Gatillos)

Los triggers contienen la lógica para determinar en que momento se debe ejecutar una tarea, la biblioteca incluye un trigger en intervalos periódicos que se ha usado para esta aplicación.

Executors (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 subprocessos (thread pool), la cantidad de hilos en el grupo esta dado por el numero de pruebas que estaremos ejecutando de modo que siempre se tenga al menos un hilo disponible cuando se inicia una prueba.

Almacén de tareas (Job store)

El almacén de tareas guarda las tareas planificadas, el comportamiento por defecto es guardar las tareas en memoria, pero existen distintos tipos de almacenes como redis (ver sección 2.8.4) y bases de datos; el comportamiento por defecto es preferido ya que solo estaremos manejando un conjunto pequeño de tareas y no es necesario ningún tipo de persistencia en caso de que el monitor se detenga.

3.2.5 Almacenamiento Compartido

El almacenamiento compartido se encarga de alojar el archivo de configuración del monitor así como los resultados de las pruebas, por cada prueba que se esté ejecutando

se crea una carpeta donde se alojan sus respectivos archivos de trazas.

Para mantener el numero de archivos en el almacenamiento compartido razonablemente pequeño se crea para cada enlace un archivo por hora y todas las trazas que se generen en ese periodo se anexan al archivo correspondiente; mientras el costo de alojar archivos en la nube no depende del numero de archivos sino del espacio total de disco en uso, existe un limite de peticiones diarias por usuario que debemos evitar superar. Este tema se explicará a profundidad en la sección ??.

El trabajo de mantener el almacenamiento compartido sincronizado con la nube se delega a una aplicación de terceros, esto no solo facilita el desarrollo de la aplicación, que sencillamente guarda los archivos en disco, sino que ademas permite cambiar con mucha facilidad el servicio de almacenamiento en la nube que se esté usando con mínimos cambios al monitor.

3.3 Estructura del archivo de configuración

El archivo de configuración determina el comportamiento del monitor de red tentacle, un requisito primordial de este archivo es que sea sencillo de leer y modificar tanto por la aplicación web como el propio monitor, por esto se eligió el formato json, que puede ser decodificado de forma trivial a estructuras de datos python.

El archivo de configuración consiste de un diccionario que tiene la siguiente estructura:

```
{
    "sleep_time": sleep_time_1 ,
    "tests":{
        "test_1":{
            "parameter_1":value_1 ,
            "parameter_2":value_2 ,
            ...
        },
        "test_2":{
            "parameter_1":value_1 ,
```

```

        "parameter_2": value_2 ,
        ...
    },
    ...
},
"links":{
    "link_1":{
        "status":true|false ,
        "ip":" ip_1",
        "id":id_1
    },
    "link_2":{
        "status":true|false ,
        "ip":" ip_2",
        "id":id_2
    },
    ...
},
"id":2
}

```

Como se puede observar, el diccionario tiene las siguientes entradas:

- **sleep time:** determina el tiempo que el hilo principal del monitor espera antes de verificar si hay cambios en el archivo de configuración
- **tests:** un diccionario cuyas claves son las pruebas a ejecutar, los valores son a su vez diccionarios con los parámetros de las pruebas
- **links:** la lista de los enlaces monitoreados, cada enlace es a su vez un diccionario con las siguientes entradas:
 - **status:** determina si se deben realizar pruebas sobre este enlace.
 - **ip:** dirección ip de enlace.

- **id:** identificador único del enlace .
- **id:** el identificador único del monitor

3.4 Pruebas Implementadas

Gracias a la arquitectura de Tentacle, 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 lee el archivo de configuración
2. Se obtienen los enlaces a monitorear y los parámetros globales
3. Por cada enlace a monitorear se crea un hilo donde se ejecutará la prueba en sí.
4. Se leen los parámetros específicos al enlace (si los hay)
5. Se ejecuta un comando externo externo como ping o traceroute o se usa una biblioteca python para evaluar alguna métrica del enlace.
6. (opcional) si se ejecuta un comando externo se hace un parsing para extraer los resultados relevantes de la salida del programa
7. Se guardan los resultados en un archivo de trazas; generalmente el resultado de una prueba está representado por una linea en archivo de trazas, sin embargo el desarrollador tiene libertad total sobre el formato que utilice

Las pruebas implementadas hasta ahora no inyectan una cantidad de trafico considerable a la red por lo que ejecutarlas en paralelo para cada enlace monitoreado no debería afectar los resultados, sin embargo también es posible implementar pruebas que se ejecuten en secuencia esto es especialmente útil si se desea medir, por ejemplo, el throughput del enlace y se desea minimizar el efecto de otros flujos de datos en la red.

3.4.1 Ping

Esta prueba hace uso del comando ping para obtener datos de la latencia en un enlace, como ya se menciona en la sección 2.8.14 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 ejecutar el comando ping para cada uno de los enlaces monitoreados, ejecutamos el comando con la opción -D para que ping imprima cada resultado de latencia con una marca de tiempo entre corchetes, un ejemplo de la salida de ping se puede ver en la figura 3.3.

Es muy sencillo extraer los datos relevantes de la salida de ping, para esto recorreremos la salida descartando las líneas que no comiencen con el carácter '[', separamos la salida en palabras, la palabra en la posición 0 corresponde a la marca de tiempo, luego buscamos las palabras que comiencen por "icmp_seq o icmp_req y time" para obtener numero de secuencia icmp y tiempo de ida y vuelta respectivamente.

```
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 3.3: Ping output

Independientemente del número de sondas que se envíen elegiremos solo un resultado de latencia por prueba (la mediana), 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 el protocolo.

Parametro	Tipo	Descripcion
Numero de sondas	Entero	Número de sondas icmp a enviar.
Timeout	Float	Tiempo a esperar por una respuesta antes de asumir una sonda como perdida.
Intervalo entre sondas	Float	Tiempo entre el envío de cada sonda individual.

Tabla 3.2: Parametros de la prueba ping

3.4.2 Httping

Esta prueba usa el protocolo HTTP para hacer un HEAD Request y obtener el tiempo de respuesta y código de estatus HTTP, a diferencia de la prueba esta no ejecuta un comando externo sino que llama a una función de la biblioteca 'requests' que hace la petición directamente por lo que no es necesario ningún tipo de parsing.

La tabla 3.4.2 muestra los parámetros de la prueba httping, note que los parámetros path y port son específicos a cada enlace, los parámetros de este tipo se guardan en el archivo de configuración uniendo el nombre del parámetro y el id del enlace usando guión bajo como carácter de separación, un ejemplo del archivo de configuración para un monitor ejecutando esta prueba se puede ver a continuación:

```
{
    "sleep_time":10,
    "tests":{
        "httping":{
            "port_8":80,
            "port_9":80,
            "path_8":"","
            "path_9":"/octopusmonitor/",
            "path_7":"","
            "interval":300,
            "port_7":80,
            "timeout":5
```

```

    }
  },
  "links":{
    "RESIDE":{
      "status":true ,
      "ip": "150.185.138.59" ,
      "id":9
    },
    "ULA Site":{
      "status":true ,
      "ip": "150.185.168.156" ,
      "id":8
    },
    "Saber ULA":{
      "status":true ,
      "ip": "190.168.5.17" ,
      "id":7
    }
  },
  "id":2
}

```

Parametro	Tipo	Descripcion
Timeout	Float	Tiempo a esperar por una respuesta
Path	String	Cadena de caracteres que se adjunta al ip o nombre de dominio del enlace, especialmente útil para probar servicios específicos de una aplicación o servicio web..
Port	Entero	Especifica el puerto al que se envía la petición HTTP.

Tabla 3.3: Parametros de la prueba ping

3.4.3 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 numero 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 3.4 en el salto 9 se observa que obtenemos una respuesta de la dirección ip 154.54.31.230 y dos de 154.54.47.154

```
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 reacciu-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 3.4: Salida del comando traceroute

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 tracerouteparser.py¹, que extrae la información de la cabecera (ip destino y nombre de dominio), así como una lista de hops (saltos), cada hop es a su vez una lista de probes (sondas), cada sonda tiene dirección ip, nombre de dominio, rtt y anotaciones; ya que el ip destino es conocido, solo se guarda en el archivo de trazas la lista de saltos en formato json.

El formato esta compuesto de la siguiente manera:

```
1 [
2     [
3         {
```

¹tracerouteparser.py es cortesía del proyecto Netalyzr: <http://netalyzr.icsi.berkeley.edu>


```

4         "anno": anno_1 ,
5         "rtt": rtt_1 ,
6         "ipaddr": ipaddr_1 ,
7         "name": name_1
8     } ,
9     {
10        "anno": anno_2 ,
11        "rtt": rtt_2 ,
12        "ipaddr": ipaddr_2 ,
13        "name": name_2
14    } ,
15    ...
16 ],
17 ...
18 ]

```

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

Las parámetros de esta prueba son los siguientes:

Parametro	Tipo	Descripcion
Numero de sondas	Entero	Número de sondas a enviar por cada valor de TTL.
TTL Maximo	Entero	Numero maximo de saltos antes de asumir que el nodo no es alcanzado.

Tabla 3.4: Parametros de la prueba con traceroute

3.5 Casos de uso

Ya que el monitor de red Tentacle funciona de forma automatizada y todas las acciones de configuración y visualización de los datos recolectados por el se realizan en la aplicación web, solo se tienen dos casos de uso para el monitor.

Tabla 3.5: Caso de uso – Iniciar monitor

<i>MU-01</i>	<i>Iniciar monitor</i>	
<i>Descripción</i>	El usuario desea iniciar el monitor de red tentacle.	
<i>Secuencia normal</i>	Paso	Acción
	1	El usuario invoca el cliente con el comando "start".
	2	El cliente crea el proceso daemon y retorna.
<i>Excepciones</i>	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 3.6: Caso de uso – Detener monitor

<i>AD-01</i>	<i>Detener monitor</i>	
<i>Descripción</i>	El usuario desea detener el monitor que se esta ejecutando en modo daemon.	
<i>Secuencia normal</i>	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.
<i>Excepciones</i>	Paso	Acción
	4	Si el archivo pid no existe, el cliente informa al usuario que el monitor no se esta ejecutando.

Capítulo 4

Aplicación Web "Optopus Monitor"

4.1 Diseño del sistema

4.1.1 Arquitectura

4.1.2 Estructura de la base de datos

4.1.3 Flujo de navegacion

4.1.4 Diseño de Pantallas

4.2 Componentes

4.2.1 Recolector de datos

4.2.2 Generación de visualizaciones

Cálculo de Mapas de Calor de RTT

Cálculo de horas activas

Cálculo de días activos

Cálculo de periodos de actividad continúa

4.3 Casos de Uso

4.4 Caching de gráficas

4.5 Pruebas de Rendimiento

Capítulo 5

Framework de integración de pruebas

Capítulo 6

Conclusiones y Recomendaciones

Bibliografía

- [1] J. F. Kurose and K. W. Ross, *Computer Networking. A Top-Down Approach*. Pearson, 2013.
- [2] M. Zennaro, E. Pietrosevoli, J. Mlatho, M. Thodi, and C. Mikeka, “An assessment study on white spaces in malawi using affordable tools,” in *Global Humanitarian Technology Conference (GHTC), 2013 IEEE*, (San Jose, CA), pp. 265 – 269, IEEE, 2013.
- [3] J. Gomez, M. Porcar, M. Hernandez, and E. Velasquez, “Malawinet network monitor,” tech. rep., Universidad de los Andes, 2015.
- [4] 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.
- [5] S. Cloud, “Pingdom - website monitoring.” [Página web en línea] Disponible en <https://www.pingdom.com/>, 2015.
- [6] U. R. B. A.S., “Uptime robot.” [Página web en línea]. Disponible en : <https://uptimerobot.com/>, 2015.
- [7] I. Grigorik, *High Performance Browser Networking*. O’Reilly, 2013.
- [8] S. D. Strowes, “Passively measuring tcp round-trip times,” *Communications of the ACM*, vol. 56, no. 10, pp. 57–64, 2013.

-
- [9] C. Demichelis and P. Chimento, “RFC3393: IP Packet Delay Variation Metric for IP Performance Metrics,” RFC 3393, RFC Editor, November 2002.
- [10] Ookla, “Speedtest.net by ookla.” [Página web en línea]. Disponible en : <http://www.speedtest.net/>, 2014.
- [11] 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.
- [12] J. Gettys and K. Nichols, “Bufferbloat: Dark buffers in the internet,” *Queue*, vol. 9, no. 11, p. 40, 2011.
- [13] S. M. LLC, “Dslreports home: Broadband isp reviews news tools and forums.” [Página web en línea]. Disponible en : <http://www.dslreports.com/speedtest/>, 2015.
- [14] N. Ltd, “Thinkbroadband :: Uk broadband speed test.” [Página web en línea]. Disponible en : <http://www.thinkbroadband.com/speedtest.html>, 2015.
- [15] R. B. Fragoso, “¿que es big data?,” *IBM developerWoks*, 2012.
- [16] E. B. Pantoja, “El patrón de diseño modelo-vista-controlador (mvc) y su implementación en java swing,” *Acta Nova*, vol. 2, no. 4, p. 493, 2004.
- [17] 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.
- [18] 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.
- [19] I. Dropbox, “Acerca de dropbox.” [Página web en línea]. Disponible en : <https://www.dropbox.com/about>, 2015.