Universidad del Bío-Bío, Concepción, 2016.

Facultad de ciencias empresariales

UNIVERSIDAD DEL BÍO-BÍO

“Simulación en NS3 de una red híbrida P2P para soportar Cloud Collaboration”

Proyecto de título para optar al título profesional de

Ingeniería civil informática.

Autores

Juan Pablo San Martín Moreno
Ramón Andrés Soto Avendaño

Profesor guía

Patricio Galdames Sepúlveda

Resumen

Este proyecto se presenta para dar conformidad a los requisitos exigidos por la Universidad del Bío-Bío para optar al título de Ingeniero Civil en Informática. Nuestro proyecto se titula “Simulación en NS3 de una red híbrida P2P para soportar Cloud Collaboration”, tiene como objetivo principal la simulación e implementación de una técnica de descarga de archivos en paralela de archivos desde una red de distribución de contenido para soportar un servicio de colaboración en la nube.

Para esta evaluación, se implementó en Java un software de descarga en paralelo de archivos que opera con HTTP. La técnica implementada es consciente de las variaciones de la tasa de transferencia obtenida desde los servidores de archivos. Además, se evaluó esta misma técnica en un simulador desarrollado en C++ con librerías de NS3. Se midió la efectividad de la técnica en términos de tiempo de descarga y des-balance en la carga de trabajo de los servidores con respecto a una técnica de descarga en paralelo no consciente de la tasa de transferencia y también contra la descarga del mejor servidor.

Índice General

Índice de Tablas

Índice de Ilustraciones

2 Introducción

En este proyecto, se presenta un estudio de aplicación de una técnica de descarga en paralelo de archivos, los cuales están replicados en diferentes servidores, compartiendo información por medio de Cloud Collaboration.

Cloud Collaboration, es una forma emergente de compartir archivos mediante la computación en la nube (Cloud Computing). Este tipo de tecnologías permiten a los usuarios subir, comentar y colaborar en documentos alojados en la nube y también acceder a estos desde cualquier lugar.

Un archivo, puede ser representado por un audio, video o material multimedia, el cual se encuentra alojado en algún sitio en Internet, él puede llegar a pesar varios Mbs o GBs, por lo que una descarga adaptativa, puede cumplir un propósito importante de reducir el tiempo de descarga y aprovechar de manera óptima el rendimiento del enlace.

Para lograr esto, es necesario generar una descarga en paralelo, la cual su principal característica es realizar conexiones concurrentes por medio de protocolos TCP, enviando la solicitud de descarga de trozos del archivo a los diferentes servidores donde se encuentra la copia.

Existen aplicaciones que operan de forma similar a la propuesta, servicios que utilizan topologías P2P y clientes que forman una red gigante de ordenadores con archivos alojados en su disco duro, cumpliendo la función de servidores.

En este proyecto se desarrolla una técnica ya estudiada llamada técnica de descarga en paralelo con variaciones conscientes del ancho de banda. Se realiza una estimación del tiempo de descarga de un trozo de archivo, del cual dependerá la cantidad de descarga para cada enlace en la siguiente iteración, de esta forma, se consigue evaluar el tiempo entre cada servidor y una métrica para poder definir rangos y tamaños de cada descarga.

El trabajo consta de dos etapas, la primera es un representación de la problemática en una simulación de eventos discretos, utilizando para este caso NS-3, el cual es un simulador de redes. La segunda etapa es implementar una aplicación en lenguaje JAVA que utilice la técnica propuesta y que trabaje con protocolo HTTP.

3 Definición del Proyecto

3.1 Objetivos del Proyecto

3.1.1 Objetivo General

- Simular e implementar aplicación con protocolo HTTP de descarga en paralelo de archivos para soportar Cloud Collaboration.

3.1.2 Objetivos Específicos

- Representar proyecto en ambiente de simulación de redes de eventos discretos NS3, con diferentes casos de aplicación.

- Desarrollar un estudio de la herramienta de simulación a utilizar, para conocer sus características y desarrollar las habilidades necesarias para el desarrollo del proyecto.

- Investigación del protocolo de intercambio de información a utilizar Http.

- Indagar en problemáticas o trabajos similares para poder obtener ideas de utilidad y ser aplicadas en el desarrollo del proyecto.

- Implementar una aplicación en lenguaje JAVA con el protocolo http.

3.2 Ambiente de Ingeniería de Software

3.2.1 Metodología de desarrollo

Se utilizará Metodología Incremental, debido al tipo de proyecto, donde a través de cada avance o iteración en las simulaciones, nos permite mejorar las versiones y añadiendo nuevas funcionalidades a estas. Permite la evolución de las entregas con valor, basándose en los resultados de entregas anteriores, mejorando en forma completa con cada avance.

Con este tipo de metodología se reduce la repetición de carga y permite separar o atacar puntos específicos en cada iteración. Optimizando la retroalimentación y el proceso de mejora continua.

De acuerdo a las cualidades de esta metodología podemos entregar versiones prototipo con funcionalidades básicas, para seguir mejorando y adaptando nuevas etapas.

3.2.2 Estándares de documentación.

Para documentar las diferentes etapas del proyecto, y tener un medio de interacción con el equipo de trabajo, se documentaran las etapas del proceso a través de una wiki (RedLine). La principal ventaja de esto, es poder lograr un orden y control de las diferentes versiones y los cambios en cada una de las etapas.

3.2.3 Herramientas

- Eclipse Versión Mars 2.0
- NetBeans Versión 8.0
- SublimeText Versión 3.0
- NS-3 versión 3.20

3.3 Definiciones, siglas y Abreviaciones

- ***P2P*:** Se define como una red de computadores en la que todos los integrantes de esta, pueden tomar los roles de servidores o clientes, para compartir archivos.[13]
- ***UDP*:** User Datagram Protocol, es un protocolo de nivel de transporte basado en el intercambio de datagramas, no es confiable. [14]
- ***TCP*:** Transmission Control Protocol, es un protocolo de transporte seguro, que entrega confiabilidad e integridad de transferencias de información.[15]
- ***MODELO OSI*:** Es un modelo de referencia para los protocolos de la red de arquitectura en capas.
- ***API*:** Interfaz de programación de aplicaciones, conjunto de subrutinas, funciones y procedimientos.
- ***HELPERS*:** Se definen como ayudantes para trabajar en la interfaz de programación de aplicaciones, para evitar el trabajo de bajo nivel. [16]

4 Estudio del Arte

4.1 Trabajos Relacionados

A través de una pequeña introducción relacionada con el tema de investigación a desarrollar, se presentan títulos que necesitan ser explicados para dar a conocer las bases de desarrollo del proyecto.

Primero que todo, debemos mencionar que en nuestro trabajo nos fue de mucha utilidad el trabajo realizado por el alumno Cristopher Barrientos Matamala [11], que desarrollo una investigación en "Descarga en paralelo de archivos sensible a variaciones del ancho de banda". Nuestro proyecto consiste en realizar una simulación e implementación de descarga en paralelo usando el protocolo HTTP, en cambio, la investigación de nuestro compañero fue la descarga en paralelo pero ocupando el protocolo FTP, donde no se desarrollaron las bases para simular múltiples escenarios en un entorno como NS3. Lo que significa que en nuestro trabajo trataremos un comportamiento de la red distinto, pero que gracias a esta investigación logramos abarcar el problema desde una mirada relativamente similar.

4.2 Simulación de redes

La simulación es una técnica muy utilizada e importante en la ingeniería, ya que nos permite modelar, observar y analizar sistemas de formas no complejas como la representación de un sistema real. A través de esta técnica es posible establecer y determinar el comportamiento de eventos o situaciones desconocidas, que podemos implementar con herramientas dedicadas a esto.

Para las simulaciones existen varios tipos de software que son capaces de representar lo que se quiere desarrollar.

4.2.1. Simulador de eventos discretos NS-2

NS-2 fue desarrollado en C++ y provee una interfaz de simulación a través de lenguaje de comandos orientados a objetos. El usuario describe una topología de red por medio de scripts OTcl, y luego el programa principal de ns-2 simula dicha topología utilizando los parámetros definidos. Ns-2 está diseñado para sistemas operativos Linux, FreeBSD, Solaris, Mac OS X y puede ejecutarse bajo Windows utilizando Cygwin. Fue licenciado bajo GPL versión 2 [17].

4.2.2 Simulador de eventos discretos NS-3

NS-3 es el sucesor de NS-2, nace el año 2005 a partir de Tom Henderson, el cual decidió realizar una nueva versión del popular NS-2, pero con la diferencia de partir desde cero, utilizando el lenguaje de programación C++.

La variante ns-3 surge en el año 2005, a partir del impulso que Tom Henderson, según la lista de correo del grupo de realizadores de ns-2 se decidió realizar una nueva versión desde cero, utilizando el lenguaje de programación C++. La infraestructura de ns-3 permite el desarrollo de modelos de simulación de alto desempeño, lo que habilita el uso de la herramienta como emulador. NS-3 soporta simulación de redes IP, no IP; así como redes inalámbricas tales como Wi-Fi, WiMAX, o LTE, además de un diferentes protocolos de ruteo entre los que se destacan OLSR y AODV [18].

4.2.3 OMNeT++

OMNet++ es un simulador basado en lenguaje C++ basado en eventos discretos utilizado para modelar redes de comunicaciones, multiprocesadores y otros sistemas distribuidos. Es una aplicación con arquitectura basada en módulos. Integra varias herramientas como un entorno de desarrollo y un analizador gráfico de resultados. Ofrece soporte para gráficos 3D y su última versión es OMNeT++5 beta 3. Dentro de las plataformas de funcionamiento se encuentra soporte para Windows, Linux, Mac OS X. Se distribuye bajo la Licencia Pública Académica, pero también cuenta con una versión comercial desarrollada por Opensim Ltd [19].

4.2.4 SimPy

SimPy es un simulador de eventos discretos basado en el lenguaje de programación Python. Proporciona los componentes necesarios para representar procesos, entidades y recursos. Se desarrolla en forma de código abierto bajo la licencia Publica General de GNU. Entre sus características sobresalen un potente editor integrado, un enfoque de modelado avanzado y potentes algoritmos de eventos discretos [20].

4.2.5 Qualnet

Es un simulador de red comercial escrito en C++, distribuido por GloMoSim , diseñado principalmente para simulaciones de redes inalámbricas, pero también puede representar redes de tipo cableadas y mixtas. También contiene una interfaz gráfica integrada para el análisis de los resultados de las simulaciones [21].

Simulador de Red	Licencia	Lenguajes	Interfaz gráfica
NS-2	Gratuita	C++ y TLC	No
NS-3	Gratuita	C++ y Python	No
OMNeT++	Gratuita	C++	Sí
Qualnet	Comercial	C++ y Parsec	No
SimPy	Gratuita	Python	No

Tabla 1 Comparativa de las características más relevantes de los simuladores expuestos

4.3 Computación en la Nube

Cloud Computing o computación en la nube es un paradigma que permite ofrecer una serie de servicios de computación a través de una red, la que generalmente es internet.

Se puede ver como un nuevo modelo de prestación de servicios, que se basa en la abstracción de la infraestructura que hace frente a las diferentes complejidades existentes. Este paradigma está tomando fuerza y está siendo implementado y potenciando rápidamente en los servicios creados en la nube.

Nuestro trabajo se enfoca a una rama de este paradigma, el cual es Cloud Collaboration. Por definición, Cloud Collaboration o Colaboración en la Nube, es una forma emergente de compartir archivos o información en una nube central de almacenamiento.

De todas formas, debemos definir primero lo que es Cloud Computing, para lograr un entendimiento y recopilación de información importante. Según diferentes autores, las visiones del significado de este paradigma son algunas de las que mencionaremos:

Según NIST (National Institute of Standards and Technology – U.S. Department of Commerce):

- La computación en la nube es un modelo que permite, un acceso ubicuo conveniente, un conjunto de recursos informáticos configurables (redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar rápidamente y poner en libertad con un mínimo esfuerzo de gestión o interacción del proveedor de servicios.[22]

Otra definición entregada por Amazon web Services es:

- Cloud Computing se refiere a la entrega bajo demanda de recursos informáticos y aplicaciones a través de Internet con un sistema de precios basado en el consumo realizado.

Según una definición del sitio web Dictionary.com es:

- Es la informática basada en Internet, en el que grandes grupos de servidores remotos están conectados en red a fin de permitir el intercambio de tareas de procesamiento de datos,

almacenamiento de datos centralizados, y el acceso en línea a los servicios o recursos informáticos.

Una definición de Cisco Systems:

- Recursos TI que se abstraen de la infraestructura subyacente y se brindan bajo demanda y a escala en un entorno multiusuario. [23]

En pocas palabras, y tratando de resumir y tomar la idea de cada definición descrita, la computación en la nube se inclina hacia la definición descrita pos NIST. Tecnologías de virtualización, portabilidad, interoperabilidad, han evolucionado para compartir los recursos mostrando los pasos a seguir hacia una nueva era.

4.3.1 Principales Características

- *En la Demanda del Autoservicio*

Un consumidor puede unilateralmente definir las capacidades, tales como el tiempo de almacenamiento y los servidores de red, según sean sus necesidades, de forma automática y sin requerir de la interacción humana con cada proveedor del servicio.

- *Amplias Redes de Acceso*

Las capacidades están disponibles en la red y se acceden a través de mecanismos estándar que promueven el uso de plataformas disponibles para los clientes (portátiles, Smartphones, Tablets, etc). Esto también permite la independencia del dispositivo y la ubicación, ya que en tiempos actuales podemos acceder a internet desde cualquier sitio solo contando con una red inalámbrica o un plan de datos.

- *La Agrupación de Recursos*

Los recursos informáticos del proveedor se agrupan para servir a múltiples consumidores mediante un modelo multiusuario, con diferentes recursos físicos y virtuales asignados dinámicamente y reasignados de acuerdo a la demanda de los consumidores.

- *Mantenibilidad*

Al estar instalas las aplicaciones o software en la nube, no es necesario contar con estos instalados en el ordenador o dispositivo que se utiliza, esto permite acceder y modificar desde prácticamente cualquier lugar.

- *Costos*

La inversión inicial y la mayor flexibilidad permite a las usuarios pagar solo por lo que se usa, esto reduce los gastos de entrada, ya que la infraestructura es proporcionada por otras partes.

4.3.2 Ventajas

Son muchas las ventajas que este paradigma ofrece, a continuación de mencionan de forma breve algunas de ellas:

- La integración que presenta este tipo de tecnología permite implementar con rapidez y facilidad los servicios.
- La energía utilizada en las nubes, varía dependiendo de la necesidad de los clientes, por lo tanto solo se utiliza lo necesario por el usuario, lo que trae consigo una ganancia considerable, tomando en cuenta el alto nivel de consumo que generan los centro de datos o datacenters.
- La implementación de esta tecnología permite reducir el factor de riesgo, ya que para comenzar no es necesario contar con un capital muy abultado. También al ser más rápida la integración, en días u horas puede estar implementada.
- La recuperación de errores y caídas de sistemas, reducción de los tiempos de inactividad, son características que proporciona este tipo de infraestructura.

4.3.3 Desventajas

- Al concentrar todos los datos en la nube, genera un cierto tipo de dependencia hacia los proveedores del servicio. El fundador de Free Software Foundation, menciona que este tipo de tecnología, en vez de crear un ambiente favorable, pone en riesgos la libertad de los usuarios, su privacidad y datos personales, dejándolos en manos de terceras personas. También hace

una afirmación relacionada a que la computación en la nube es “simplemente una trampa para obtener sistemas propietarios”.

- La seguridad depende netamente de las capacidades de los proveedores que ofrecen los servicios.
- Dependencia de la Red en su totalidad.
- En un futuro, la sobrecarga de clientes presentes con el servicio, generará un aumento de utilización de infraestructura, por lo cual, si la empresa no tiene visiona de crecimiento y modernización, provocara un problema de escalabilidad a largo plazo.

4.3.4 Servicios

4.3.4.1 Software como servicio

Es una aplicación completa ofrecida como servicio, su definición en inglés es “software as a service (SaaS). Este tipo de aplicaciones son accesibles desde cualquier navegador web. El usuario final no tiene control sobre ella.

4.3.4.2 Plataforma como servicio

Es la encapsulación de un ambiente de trabajo, empaquetando los módulos y complementos que conforman las funcionalidades de la plataforma. En inglés “Platform as a service (Pass)”. Este modelo de plataforma ofrece un ambiente de desarrollo y brinda herramientas para todas las fases, pero no controla la infraestructura.

4.3.4.3 Infraestructura como servicio

Es un servicio que entrega almacenamiento y servicios de cómputo. Todo lo relacionado con hardware, en inglés “infrastructure as a service (IaaS).

Existen varios tipos de infraestructuras dedicadas para ciertos tipos de usuarios o consumidores, a continuación se detallan:

4.3.5 Tipos de Cloud o Nubes

- Nubes Públicas

Varios clientes se encuentran ejecutando tareas en el mismo servidor o red. Este tipo de nube es mantenida por terceras personas, las cuales no se encuentran relacionadas con la organización.

- Nubes Privadas

En este tipo de nubes se proveen los datos, donde se controla todo lo que se ejecuta. Es administrada por personal que es propietario de la compañía.

- Nubes Comunitarias

Por definición del Instituto Nacional de Estándares y Tecnología, es la nube que sirve o tiene como finalidad cumplir una función o propósito común relacionado con la comunidad, política, seguridad o servicio.

- Nubes Híbridas

La infraestructura de este tipo de nube está compuesta por dos o más tipos de nubes (privada,

Comunitario o pública).

Un servicio de colaboración en la nube permite que dos o más personas puedan compartir y trabajar juntos a la vez en los documentos, contenidos multimedia, y otros tipos de datos. Una solución sencilla para proporcionar una infraestructura de almacenamiento compartido está en utilizar un servidor dedicado. Este servidor mantiene un directorio que contiene a todos los miembros comparten los derechos para editar un archivo en particular. Cuando un miembro colaborador o grupo elegido actualiza un archivo, este servidor intenta enviar tan pronto como sea posible esta nueva versión del archivo a todos los grupos de compañeros uno por uno. [25]

- Google

Ofrece servicios basados en la nube, los cuales permiten crear desde simples sitios web hasta aplicaciones de alta complejidad. Cuenta con servicios de Cálculo, Almacenamiento, Big Data, etc.

- Amazon

Amazon es una compañía que ha evolucionado considerablemente con el paso del tiempo, cuenta con una gran cantidad de servicios basados en infraestructuras y plataformas.

- Dropbox

Dropbox es un servicio de almacenamiento en la nube, que utiliza la infraestructura como servicio (IaaS). Ofrece una solución de almacenamiento en línea, accesible desde cualquier lugar a través de Internet.

- Sugarsync

Es un servicio que genera copias de seguridad en la nube de almacenamiento de datos y proporciona soluciones integrales para la gestión de datos personales y profesionales.

- Rackspace

Empresa dedicada a entregar servicios de almacenamiento en la nube, algunos de ellos son: Servidores dedicados, nubes privadas, públicas, híbridas, todo tipo de infraestructura. Una de las más completas.

4.4 Protocolo HTTP

HTTP o Hypertext Transfer Protocol es un protocolo de comunicación que permite realizar transferencias de información en la web. También es el encargado de definir la sintaxis y la semántica que utilizan los elementos de la arquitectura web. Este protocolo no tiene un estado, por lo cual no guarda ningún tipo de información sobre alguna conexión anterior. Desde sus inicios cuenta con varias versiones, de las cuales cada versión más reciente es compatible con las características de las versiones anteriores. El uso de este protocolo está definido en los documentos llamados RFC, dependiendo de la versión del protocolo existe un documento RFC con un digito identificador.[1]

En sí, Http es un protocolo basado en transacciones que sigue el esquema petición respuesta, operaciones sencillas donde un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado.

Los mensajes enviados tienen la característica de ser en texto plano, lo que lo hace más legible y fácil de depurar. Para realizar las peticiones, se define una serie de métodos que pueden utilizarse, los que aumentan a medida que se crean nuevas versiones. Los principales métodos o verbos, como suelen llamarse, son los siguientes:

HEAD: Método que solicita una respuesta idéntica a la petición GET, pero sin el cuerpo del mensaje.

GET: Solicita una representación del recurso especificado y lo transmite a través de la URL.

POST: Método que envía datos para ser procesados por el recurso identificado.

PUT: Método que sube, carga o realiza un upload de un recurso especificado.

DELETE: Borra el recurso especificado.

TRACE: Método que solicita al servidor devolver todos los datos contenidos en el mensaje de petición.

OPTIONS: Devuelve los métodos HTTP soportados por el servidor.

CONNECT: Se utiliza para comprobar al acceso a un host.

4.4.1Versiones de HTTP

HTTP 0.9 (1991): Obsoleta, solo soportada un método (GET).

HTTP/1.0 (1996): Primera versión que especifica su versión en las comunicaciones. Permite los métodos GET, POST y HEAD.

HTTP/1.1 (1999): Es la versión más usada actualmente. Permite enviar múltiples peticiones a la vez por la misma conexión.

HTTP/2 (2015): Esta versión no modifica la semántica de aplicación de versiones HTTP anteriores, pero cuenta con mejoras en cómo se empaquetan los datos y el transporte.

4.4.2Respuestas del protocolo HTTP

Cada vez que enviamos una petición a un servidor, el protocolo nos devuelve una respuesta, la cual consta de unos códigos con formato numérico de 3 dígitos.

Cada uno de estos códigos tiene un significado concreto.

- *Códigos 1xx:* Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.
- *Códigos 2xx:* Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- *Códigos 3xx*: Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- *Códigos 4xx:* Errores causados por el cliente. Indica que ha habido un error en el proceso de la petición a causa de que el cliente ha hecho algo mal.
- *Códigos 5xx:* Errores causados por el servidor. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor.

4.5 Protocolos de Transporte

En el cuarto nivel del modelo OSI, se encuentra la capa de transporte, la cual es la encargada de la transferencia libre de errores de los datos entre el emisor y receptor y de mantener el flujo existente en la red. La principal función de esta capa es otorgar confiabilidad entre las máquinas de origen y destino.

Sus protocolos son TCP y UDP, el primero orientado a conexión y el segundo sin conexión.

4.5.1Protocolo TCP

Transmission Control Protocol (TCP) fue creado entre los años 1973 y 1974 por Vint Cerf y Robert Kahn y es uno de los protocolos más usados en Internet. TCP garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron, sus principales características son:

- Orientado a la conexión

  Cuando dos máquinas establecen una conexión para intercambiar datos, los extremos se sincronizan para manejar el flujo de paquetes y adaptarse a la congestión de la red.

- Revisión de Errores

  TCP utiliza como técnica de comprobación de errores una técnica llamada checksum, la cual verifica que los paquetes no sean corruptos.

- Acuses de Recibo

  Cuando el receptor recibe el paquete, este genera un tipo de aviso transmitiendo que recibió correctamente el paquete.

- Control de Flujo

  Al transmitir los acuses de recibo, se puede alertar cuando los paquetes presentan problemas, por ejemplo, si se está desbordando el buffer, los acusos de recibo deberían indicar que está pasando y bajar la tasa de transferencia o dejar de transmitir.

- Servicio de Recuperación de paquetes

  Si el paquete no está definido con un acuso de recibo positivo, el receptor puede pedir la retransmisión de este.

Offsets Octeto		0								1								2								3							
Octeto	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Puerto de origen																Puerto de destino															
4	32	Número de secuencia																															
8	64	Número de acuse de recibo (si ACK es establecido)																															
12	96	Longitud de Cabecera				Reservado				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Tamaño de Ventana														
16	128	Suma de verificación																Puntero urgente (si URG es establecido)															
20 ...	160 ...	Opciones (Si la Longitud de Cabecera > 5, relleno al final con "0" bytes si es necesario) ...																															

Ilustración 1 Cabecera de protocolo TCP (Wikipedia)

4.5.2 Protocolo UDP

User Datagram Protocol (UDP) es un protocolo de transporte que se basa en el intercambio de datagramas. Permite el uso sin tener que establecer una conexión previ esto debido a que contiene mucha información en su interior. A diferencia de TCP, en este protocolo no se puede obtener información relacionada con la llegada de los paquetes, si ha llegado bien o no a su destino, o que se haya enviado un paquete antes que otro, en palabras resumidas no nos otorga garantías. Sus principales características son:

- Es un protocolo mínimo de nivel de transporte orienta a mensajes (datagramas).
- No otorga garantías en la entrega de los paquetes.
- No es fiable, no ordena los paquetes ni presenta un control del flujo.
- Provoca poca carga adicional en la red ya que es sencillo y emplea cabeceras muy simples.
- No trabaja con bytes, a diferencia de TCP. Este protocolo solo utiliza paquetes o datagramas enteros.

+	Bits 0 - 15	16 - 31
0	Puerto origen	Puerto destino
32	Longitud del Mensaje	Suma de verificación
64	Datos	

Ilustración 2 Cabecera de protocolo UDP (Wikipedia)

4.6 Redes P2P (De distribución de contenido)

P2P significa "Peer to Peer" (Par a Par o Igual a Igual), no es una red ni un software, más bien está definido como una estructura de red o una forma de organización lógica. Esto significa que P2P no define un protocolo en específico o reglas para su uso, P2P únicamente indica la manera en que se deben de realizar las conexiones y la organización de nodos, pero dejando a la implementación definir detalles de coordinación (protocolos), estructura y seguridad (autentificación, sesión, etc.).

En las redes P2P se considera a cada Nodo un "peer" o "par" dado que todos son considerados iguales dentro de la red, al contrario de las redes cliente-servidor, en P2P los recursos provienen de cada uno de los nodos y se utilizan conexiones Ad-hoc, una conexión de este tipo es una conexión dedicada a resolver un problema específico (en este caso comunicación de los nodos).

P2P es muy sencillo de entender y normalmente no hay que adentrarse en detalles, pero a diferencia de otro tipo de conexiones (por ejemplo FTP, HTTP, etc.) hay que conocer algunos términos y conceptos para poder usar los sistemas P2P de manera eficiente. [13]

La diferencia entre una red de distribución de contenido y una red P2P tradicional es la estabilidad de la red y que existe solo un administrador a cargo de los servidores.

4.6.1 Características

- Escalabilidad: Las redes peer to peer poseen una cobertura mundial dentro de las cuales millones de usuarios las utilizan día a día. Cuantos más usuarios estén conectados y compartiendo archivos mejor será el funcionamiento.

- Robustez: al ser redes descentralizadas, si se producen fallos en algunos de los nodos de la red, esto no tiene repercusiones en el resto de los nodos.

- Descentralización: son redes entre nodos iguales por lo que son descentralizadas
  Repartición de costes entre usuarios: existe un intercambio constante de información, se comparte información a cambio de información entre los nodos.

- Seguridad: no es una característica muy desarrollada en las P2P, al existir código malicioso en algunos nodos, algunos mecanismos de seguridad son: cortafuegos, cifrado, comentarios etc.

- Anonimato: En estas redes siempre funciona el anonimato de los usuarios, un pequeño porcentaje da a conocer su verdadera identidad.

4.6.2 Clasificación

Se puede realizar una clasificación acorde a su grado de centralización:

- Redes centralizadas: Todos los intercambios de información se realizan a través de un único servidor que sirve de punto de enlace entre el resto de los nodos distribuyendo y reenviando la información y las peticiones de los clientes. Carecen de privacidad entre usuarios y de escalabilidad.
- Redes semi-centralizadas: Existe un servidor central del que administra los recursos de banda ancha, enrutamientos y comunicación entre nodos pero sin saber la identidad de cada nodo y sin almacenar información alguna, solo actúa de coordinador. El resto de los nodos almacena la información, mejorando de esta forma la escalabilidad de la red.
- Redes descentralizadas: Son las redes más comunes, y las que se utilizan en la actualidad, además de cumplir con las características antes mencionadas. Las comunicaciones son directamente de usuario a usuario con ayuda de un nodo (que es otro usuario) quien permite enlazar esas comunicaciones.

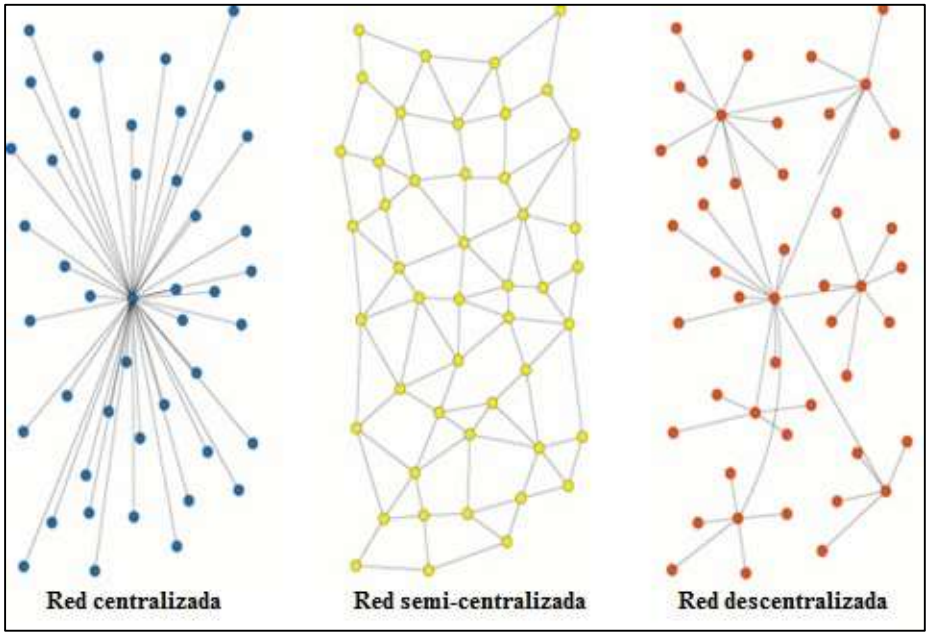


Ilustración 3 Clasificación redes P2P según su centralización (Wikipedia).

Según su tipo de estructura:

- *Estructuradas:* Existen categorías de nodos que permiten un control sobre la estructura de la red.
- *Desestructurado:* Las conexiones y la estructura general son arbitrarias.

Las redes P2P también se pueden clasificar según la generación:

- *1º Generación:* las primeras redes P2P que eran centralizadas.
- *2º Generación:* aparecen las primeras redes semi-descentralizadas.
- *3º Generación:* son las redes más recientes, totalmente descentralizadas.

Además, podemos clasificar las redes de acuerdo a la forma en que realizan las búsquedas y administran los archivos (en caso de ser transferencias), podemos distinguir 2 grandes grupos:

- *Redes Globales:* En este tipo de redes existe solamente una red de servidores que proveen servicio a todos los usuarios, normalmente esta operada por una compañía u organización, aunque también puede ser puramente aleatoria, ejemplos de estas redes son Freenet, Kazaa, Gnutella y OpenFT.
- *Redes Privadas:* En este tipo de redes no existe una red principal, solo existen muchas redes basadas en el protocolo que pueden o no estar interconectadas entre si, ejemplos: bittorrent y DirectConect.

Usos de P2P

- Transferencia de Correos: el email se transmite usando una red P2P entre los MTA (mail transport agents).
- Multimedia: Existen aplicaciones para visualizar video en demanda a través de redes P2P como democracy player, skype también se apoya en parte por un sistema P2P.
- Investigación: Existen varios proyectos que buscan resolver problemas complejos utilizando P2P y cómputo distribuido.

- Anonimato y Libertad de Expresión: Existen redes P2P cuya finalidad es proporcionar anonimato a los usuarios.

- Transferencia de Archivos grandes: Hay muchas organizaciones que alientan a los usuarios a distribuir sus archivos en redes P2P, tales como distribuciones Linux, fundaciones de software libre, sistemas abiertos, etc.

Además varios protocolos se apoyan en sistemas P2P para agilizar procesos, así como algunos P2P se apoyan en redes cliente servidor, un ejemplo es en protocolos de enrutamiento (usados en el Internet), donde la información sobre las conexiones de los host se intercambia entre cada router formando una red P2P.

Ejemplos de redes protocolos y software P2P

- *Hotline Connect:* desarrollada por Adam Hinkley para MaC OS por HCL con el objetivo de convertirse en una plataforma de distribución de archivos para empresas y particulares.
- *Napster:* Creada por Shawn Fanning y Sean Parker para el intercambio de archivos MP3 en redes semidescentralizadas.
- *eDonkey:* protocolo5 de redes semidescentralizada del que surgiría eMule.
- *Gnutella:* uno de los primeros protocolos en usar redes P2P totalmente descentralizadas.
- *Skype:* Software utilizado para llamadas en internet VoIP6 desarrollado por Janus Friis y Niklas Zennström creadores de Kazaa.
- *Freenet:* red de distribución de información sin censura y que defiende el anonimato.
- *Spotify:* software para la transferencia de archivos de audio que combina un servidor streaming7 y la transferencia P2P.
- *BitTorrent:* protocolo para el intercambio de archivos forzando a todos los que descargan un fichero a compartirlo también con otros. El servidor está centralizado.

4.7 Descarga Paralela

Para reducir los tiempos de descarga de archivos de gran tamaño en web existen técnicas que implementan métodos viables que se presentan como soluciones al problema. Varios investigadores se han centrado en desarrollar técnicas que mejoren esto, para nuestro caso destacan Byers [4], Funasaka [3] y Rodríguez [2].

Byers propone una técnica la cual consiste en la división de un archivo en n partes y luego generar k partes redundantes con as n partes iniciales, esto mejora la disponibilidad del contenido.

Luego, la propuesta de Rodríguez, toma en cuenta los recursos disponibles en la red, evaluando los servidores. Basándose en los resultados de la evaluación de los recursos, crear una tasa histórica que sea asignada de forma adecuada para cada servidor hasta completar la transferencia del archivo.

Funasaka, propuso tomar en cuenta la sensibilidad del ancho de banda, asignando una cantidad de tamaño determinada a descargar desde cada servidor dependiendo de su calidad de ancho de banda.

La técnica que nosotros implementaremos, desarrollada por Barrientos [11], consiste en tomar el tamaño total de un archivo replicado o alojado en varios servidores, para luego, dividir en partes iguales dependiendo de un número de etapas definidas. En cada iteración realizada, se evaluara el ancho de banda de cada enlace para realizar una reasignación de paquetes a transmitir en la iteración siguiente.

En pocas palabras, es una implementación que incluye las técnicas propuestas por Byers [4], Funakasa [3] y Rodríguez [2].

5 Estudio del Simulador

5.1 NS-3

NS3 (Network Simulator 3) es un simulador de redes de eventos discretos, principalmente utilizado como herramienta en la construcción de redes de telecomunicaciones. Este simulador, esta complemente escrito en C++, por lo que principalmente sus escenarios de comunicación son creados en este mismo lenguaje. También es compatible con lenguaje Python.

Al igual que su antecesor (NS-2), consta de un núcleo que está conformado por librerías. Cada una de estas librerías cuenta con los métodos necesarios para representar un programa, protocolo u objeto.

Para simular una red, es necesario crear un script en C++ o Python, donde se instancian los elementos a utilizar, sus parámetros y características.

Se definió el uso de este simulador, ya que trae los últimos desarrollos implementados y una serie de características como la modificación de su código fuente, implementación de manejo de eventos, y variadas funcionalidades.

Esta aplicación, es una versión complemente nueva en relación a su antecesor (NS2), mejorando sus fallos y creando un simulador desde cero.

NS3 es un proyecto relativamente nuevo, que tiene pocos años de desarrollo en comparación a su predecesor NS2.

NS-3 tiene implementada una API de Helpers que permiten simplificar la codificación, permitiendo escribir scripts con un manejo fino de todos los objetos, de forma eficiente y configurable, ofreciendo una interfaz amigable al programador.

5.2 Principales características

- Código: El núcleo de este simulador es C++. Simulaciones se pueden desarrollar en C++y Python.
- Desarrollo de modelos similares al mundo real.
- Paquetes con información de datos reales.
- Permite mezclar la emulación con la experimentación.
- Niveles físicos configurables.
- Escalable a varios escenarios.
- Amplio respalde de la comunidad de desarrollo, esto al ser una aplicación open source.

5.3 Definición de Helpers y Clases utilizadas

- *NodeContainer*
  NodeContainer es una especie de contendor que se utiliza para simplificar el uso de arreglos o conjuntos de nodos que necesitan algún tipo de configuración. Permite la simplificación de código.

- *ApplicationContainer*
  Normalmente las aplicaciones necesarias en un nodo se instalan con el helper de application. Este helper, toma un NodeContainer el cual tiene un atributo Ptr<nodo>. Para cada uno de estos nodos se crea una instancia de la aplicación y se añade al NodeContainer. ApplicationContainer funciona como contenedor de esto.

- *AnimationInterface*
  Es una interfaz con el animador de red, la cual proporciona funciones que facilitan la comunicación con alguna aplicación externa o interna para representar la simulación.

- *Address*
  Es una clase de dirección, que permite asignar un identificador y proporcionar métodos para instanciar direcciones.

- *InternetStackHelper*
  Agrega funcionalidades relacionadas con protocolos de transporte y de red.

- *PointToPointHelper*
  Utilizado para la creación de redes punto a punto, este helper permite instanciar una red de este tipo de forma rápida y segura.

- *Ipv4AddressHelper*
  Permite la asignación de direcciones en redes, asigna de forma automática incrementando dígitos de las direcciones IP.

- *Ipv4GlobalRoutingHelper*
  Permite realizar una pre-simulación de cálculo de la ruta estática en una topología de capa 3 IPv4.

- *PacketSinkHelper*
  Utilizado para realizar de manera más simple la creación de una instancia de una NS3. Se aplica un receptor de paquetes en un conjunto de nodos.

- *OnOffHelper*
  Crear una instancia de tal forma que permite trabajar de manera mucho más sencilla con OnOffApplications. Un OnOffAplication es un generador de tráfico que sigue un patrón de encendido / apagado. "On" y "Off", son los estados que se alternan. La duración de cada uno de estos estados se determina con la instancia onTime y las variables aleatorias con offTime. Durante el estado "Off", no se genera ningún tráfico. Durante el estado "On", se genera el tráfico CBR. Este tráfico CBR se caracteriza por la "velocidad de datos" especificada y "tamaño de paquete".

6 Definición del Proyecto

6.1 Simulación de una red híbrida P2P en NS3

En este tipo de red, híbrida P2P, tiene la peculiaridad de poder trabajar de dos formas, la primera, incorporar más de un servidor que gestiona los recursos y también de otra forma la cual es al estar unidos nodos con nodos establece una conexión directa entre ellos mismos, por lo cual es posible seguir compartiendo y descargando información.

La finalidad de crear una red de este tipo, es someterla a un escenario donde al existir una conexión nodo a nodo, con archivos replicados en varios equipos terminales, evaluar el estado de la red y el nivel de flujo de las conexiones entre un nodo receptor y un nodo transmisor. Luego dependiendo de la calidad del enlace, poder cambiar en tiempo real de descarga, los tamaños asignados de descarga en un principio después de la descarga de prueba.

Se simulará una red conformada por servidores, routers y equipos clientes, mediante una topología de red definida en los experimentos, para representar una transmisión de datos simulada, que permita obtener métricas y resultados para luego ser evaluados.

6.2 Implementación de aplicación Java con protocolo Http

Crear una aplicación en Java que permita realizar descargas paralelas de un archivo replicado en varios servidores. Tiene relación con el modo de descarga de las partes del archivo que se realizara en la simulación, ya que se quiere implementar un algoritmo que permita la modificación del tamaño de paquetes que se descargará desde cada servidor, luego unir el archivo y comprobar que este correcto sin daños.

6.3 Técnica Propuesta

En esta sección se detalla la técnica propuesta y sus pasos:

1-. Se crea una topología base, la cual se encuentra definida por Routers y equipos terminales. Esta topología, utiliza el protocolo de transporte TCP. Para simular un archivo, se trabajara con flujos de datos o paquetes, los cuales serán enviados desde los equipos servidores al equipo cliente.

2-. Al iniciar la simulación, se enviara un paquete desde los servidores a evaluar hacia el nodo cliente, con un tamaño definido por 10 Mb aproximadamente. Este paquete será denominado como paquete de muestra, con el cual podremos obtener una métrica para definir los tiempos de descarga para cada enlace y obtener la variable que necesitamos. Con esto podremos definir que servidor presenta un mejor enlace.

3-.Basándose en la transmisión del paquete de muestra, podemos obtener una estimación de velocidad de carga del enlace.

4-. Luego de obtener la estimación de carga, se asignan los tamaños de los paquetes que descargara cada servidor.

5-.Para cada iteración de los paquetes restantes, se realizara el mismo proceso de evaluación de calidad de los enlaces, esto servirá para poder modificar ante cualquier variación de los servidores, y volver a reasignar las particiones de los paquetes.

6-. Al momento de recibir todos los paquetes definidos, se implementa un monitor de eventos para poder verificar y comprobar los datos transmitidos por cada enlace y el tiempo de ejecución total de la simulación.

Implementación

1-. El primer paso es enviar desde los servidores el paquete de muestra para la evaluación del enlace.

Este paquete que se envía, denominado muestra, nos sirve como regulador del tiempo y tamaño del paquete a descargar.

2-.Lo primero que se produce es la medición del tiempo que demora en llegar el paquete completo al cliente. Para obtener esto de declarará la siguiente fórmula:

- $r_{ij} = (1-\gamma)* l_i/t_i + \gamma * r_{ij-1}$ (1)

Donde:

- R_{ij} = Corresponde a la tasa de descarga del servidor i en la iteración j.
- $\frac{L_i}{T_i}$ = Li Corresponde al tamaño de la muestra y Ti al tiempo de descarga de la muestra.
- γ = Es una variable general de 1/8, pero por simplicidad se aproxima a 0.

3-.Luego de obtener la tasa de descarga del servidor con el paquete de muestra, se procede a evaluar cada etapa para obtener la cantidad de Bytes que transmitirá cada enlace, para ello se utilizan las siguientes formulas:

- $\varepsilon = \Sigma\, r_{ij} \cdot X$ (2)

En donde:

- ε = Representa el tamaño en bytes de cada etapa.
- $\sum R_{ij}$ = Sumatoria de las tasas de descarga de cada servidor.
- X = Variable que representa la constante que al multiplicar con cada tasa de descarga entrega el valor de Bytes a descargar por cada servidor.

Despejando obtenemos: $$X = \frac{\varepsilon}{\sum R_{ij}} \quad (3)$$

4-. Para obtener el tamaño del rango de bytes a descargar por cada servidor se utiliza:

$$S_i = \varepsilon \times \frac{R_{ij}}{\sum R_{ij}} \quad (4)$$

Donde:

S_i =Corresponde al servidor con índice i.

5-. Luego de proceder al cálculo de estas operaciones, obtendremos los valores totales que deberá descargar cada servidor dependiendo de su etapa.

6-.Para cada iteración de las etapas, se procederá a realizar el mismo cálculo, para poder ir viendo en tiempo real como se comporta el problema.

Para visualizar de manera practica el problema, se tratará de ejemplificar la situación con el siguiente supuesto:

Tenemos una topología la cual está conformada por 3 equipos terminales y 4 Routers.

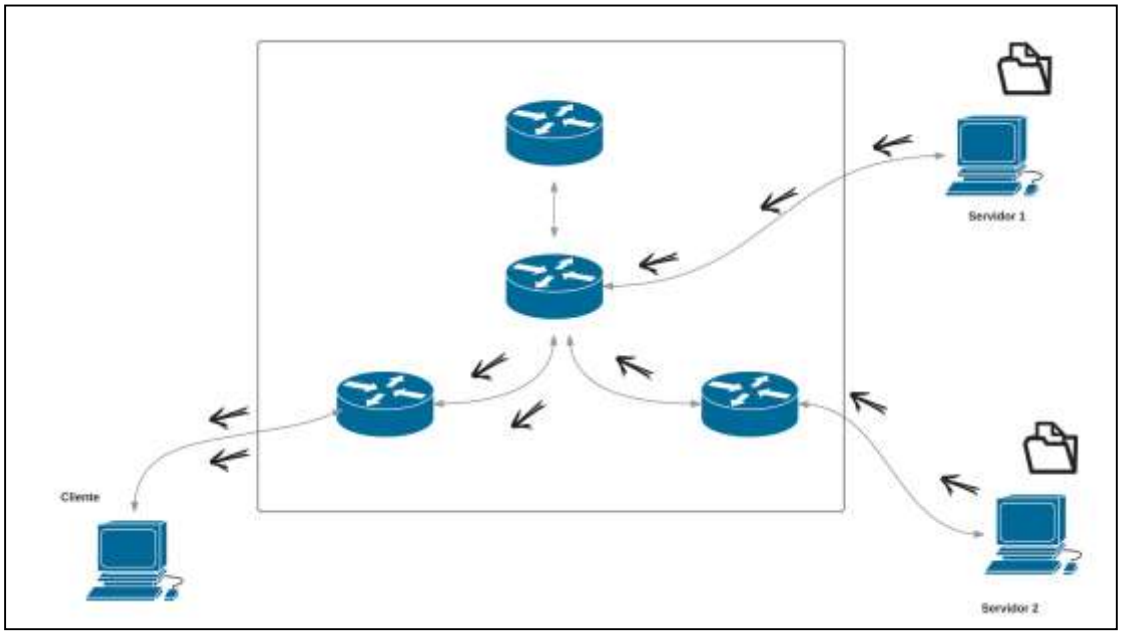


Ilustración 4 Caso de prueba caracterización de topología P2P

El equipo cliente necesita descargar un archivo que se encuentra replicado en el servidor 1 y servidor 2, supongamos que es un archivo que pesa 100Mb.

1-. Definimos la cantidad de etapas del proceso que necesitamos generar. Como la red consta de 2 servidores, dividiremos las etapas en 4, las cuales transmitirán paquetes de 25Mb cada una.

Necesitamos transformar los 25 Mb a Bytes, lo que nos arrojan 26214400 bytes.

2-. Antes que todo, se simula un envió de un paquete muestra, el cual definiremos un tamaño de 10 Mb, transformados son aproximadamente 10485760 bytes.

Resumiendo tenemos:

- 4 etapas de 26214400 Bytes cada una.
- Paquete de muestra de 10 Mb transformados pasan a 10485760 bytes.

3-. El paquete de muestra, tiene un tiempo de descarga para el servidor 1 de 35000 milisegundos (3.5 segundos) y para el servidor 2 de 55000 milisegundos (5.5 segundos).

Aplicando la fórmula 1, obtenemos lo siguiente:

R_{ij} Servidor 1 = 299,5931429

R_{ij} Servidor 2 = 190,6501818

4-. Tomando estos valores, podemos aplicar la formula (2) para obtener el valor de la constante X.

El valor de cada etapa es de 26214400 bytes, por lo cual tenemos:

ε = 26214400 Bytes

X = 26214400/ (R_{ij} servidor 1 + R_{ij} servidor 2)

X= 26214400 / (299,5931429 + 190,6501818)

X = 53472.22222

5-. Para obtener la información de cuantos bytes debemos descargar desde cada servidor, se utiliza la formula (4).

Servidor 1 = 26214400 * (299,5931429 / (299,5931429 + 190,6501818)

Servidor 1 = 16019 bytes a descargar por el servidor 1 en la etapa 1

Servidor 2 = 26214400 * (190,6501818 / (299,5931429 + 190,6501818)

Servidor 2 = 10194 bytes a descargar por el servidor 2 en la etapa 1

Suma de ambas descargas =26213000 bytes.

Por lo tanto, queda un margen de 1400 bytes que se deberán descargar en la próxima iteración.

Cada iteración deberá realizar la misma comprobación para así poder obtener el mejor rendimiento, por lo tanto el paquete muestra, en la segunda iteración, será representado por el paquete que fue descargado en la primera iteración.

7 Desarrollo de Simulación

7.1 Topología

En nuestro caso de simulación, se utilizara una topología de red híbrida P2P, es decir, una red con equipos interconectados entre sí, donde todos los nodos participantes de la red actúan como servidores y clientes a la vez. Este tipo de redes permite la comunicación y transferencia de información entre cualquier dispositivo conectado a la red.

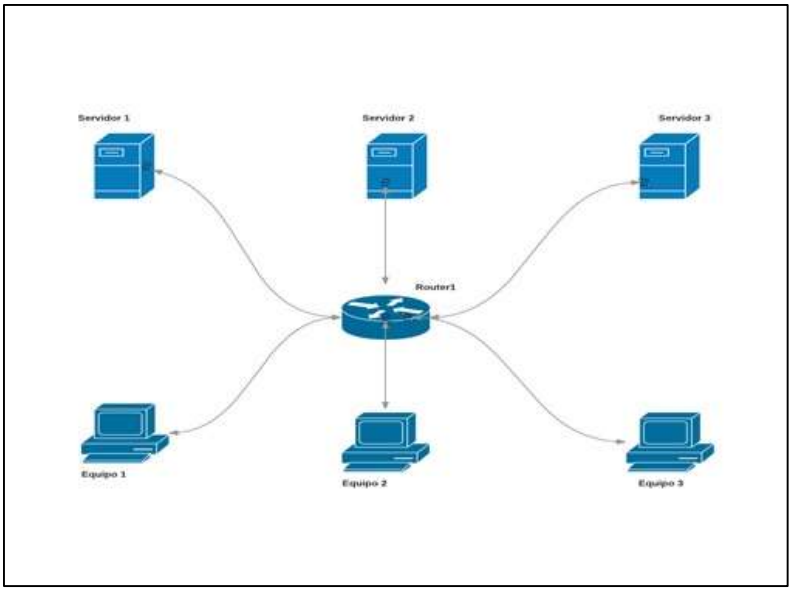


Ilustración 5 Ejemplo de topología implementada

7.2 Implementación de red con tráfico TCP

Existen varias formas de generar tráfico de red en NS-3, la primera utiliza sockets, los cuales se generarán en el nodo destino y nodo genera el recurso, luego bajo funciones de direccionamiento es posible implementar un recorrido de información desde un nodo a otro.

La segunda forma es por medio de la creación de una app (ApplicationContainer), la cual nos permite trabajar de manera simplificada, ya que se encarga del código de bajo nivel en el simulador.

Cabe mencionar, que la primera opción, permite obtener un mejor manejo de la información y podemos ver como se ejecuta el código.

8 Especificación de Requerimientos de Software

8.1 Alcances

El desarrollo de esta aplicación, tiene como objetivo principal implementar la técnica propuesta de descarga en paralelo de archivos descrita en el capítulo 8. Para poder llevar a cabo una correcta ejecución, el usuario debe indicar un mínimo de dos servidores donde se encuentre un archivo replicado en cada uno de estos.

El método de protocolo de transporte utilizado es HTTP, el cual simplifica la etapa de conexión al momento de solicitar el archivo al servidor.

La capacidad del software es de solo descargar un archivo a la vez, no funciona como un gestor de descargas que incorpora colas de espera para diferentes archivos.

Si una descarga es interrumpida, el software debe terminar de ejecutar los procesos que aún siguen activos. Al término de estos, reanudar el enlace de descarga caído.

8.2 Objetivo del software

Desarrollar en Lenguaje Java la técnica propuesta en el capítulo 8

8.3 Requisitos mínimos del software

Para el correcto funcionamiento de la aplicación, se debe contar con los requerimientos mínimos, ya sean de software como hardware.

Para la aplicación, está comprobada la ejecución en plataformas Linux y Windows.

8.3.1 Requisitos de sistema operativo

Para ejecutar la aplicación es necesario contar con la máquina virtual de Java instalada en el ordenador.

Basándose en esto, se describen las versiones de los sistemas operativos que cumplen con los requisitos:

Windows

- Windows 10 (7u85 y superiores)
- Windows 8.x (escritorio)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 SP2 y 2008 R2 SP1 (64 bits)
- Windows Server 2012 (64 bits) y 2012 R2 (64 bits)

Mac OS X

- Mac con Intel que ejecuta Mac OS X 10.7.3+, 10.8.3+, 10.9+
- Privilegios de administrador para la instalación

Linux

- Oracle Linux 5.5+
- Oracle Linux 6.x (32 bits), 6.x (64 bits)3
- Oracle Linux 7.x (64 bits)3 (7u67 y superiores)
- Red Hat Enterprise Linux 5.5+, 6.x (32 bits), 6.x (64 bits)3
- Red Hat Enterprise Linux 7.x (64 bits)3 (7u67 y superiores)
- Suse Linux Enterprise Server 10 SP2, 11.x
- Suse Linux Enterprise Server 12.x (7u75 y superiores)
- Ubuntu Linux 10.04 y superiores

8.3.2 Requisitos de Software

Nombre: Java Runtime Enviroment

Desarrollador: Oracle.

Versión: Version 8 Update 73 o posterior.

8.3.3 Requisitos de Hardware

Dependiendo del sistema operativo, existen diferencias de la configuración mínima con la que deben contar cada ordenador.

Windows:

- RAM: 128 MB; 64 MB para Windows XP (32 bits)
- Espacio en disco: 124 MB

Linux:

- Ram: 64MB.
- Espacio en disco: 58 MB.

8.3.4 Interfaces de comunicación

Se detallan los protocolos de comunicación necesarios para generar la comunicación desde el ordenador con los servidores relacionados en la solicitud de la descarga del archivo.

Nombre**:** Hypertext Transfer Protocol (“Protocolo de transferencia de hipertextos”)

Abreviación: HTTP

Es el protocolo de comunicación que permite las trasferencias de comunicación en la World Wide Web. Es un protocolo sin estado, el cual no guarda información sobre conexiones anteriores.

Nombre**:** Transmission Control Protocol

Abreviación: TCP

Es el protocolo que se usa para el nivel de transporte de los segmentos de una manera confiable libre de errores y sin pérdidas.

8.4 Requerimientos funcionales del sistema

Id	Nombre	Descripción
1	URL de servidores.	El sistema debe mostrar un aviso por pantalla al usuario, de que requiere al menos dos URL cuando este intenta iniciar la descarga con una o menos URL.
2	Integridad del archivo	El archivo descargado tiene que contener la misma suma de verificación (cantidad de Bytes iguales) que el archivo original para asegurar su integridad y completitud.
3	Interrupción de la descarga	El sistema debe mostrar un aviso por pantalla al usuario cuando se interrumpe la descarga con un servidor o sufre una caída completa.
4	Tiempos de descarga	El sistema debe mostrar un aviso por pantalla al usuario del tiempo efectuado en la descarga total del archivo y el tiempo del desbalance entre servidores.
5	URL invalidas	El sistema debe mostrar un aviso por pantalla al usuario cuando una URL no es válida o no logra conexión con el servidor.
6	Avance de la descarga	El sistema debe mostrar por pantalla el avance de la descarga al usuario constantemente.
7	Estado de la descarga	El sistema debe permitir al usuario iniciar, pausar, reanudar o cancelar una descarga.
8	Destino del archivo	El sistema debe permitir al usuario determinar donde se almacenara el archivo a descargar.

Tabla 2 Requerimientos funcionales

9 Análisis

9.1 Procesos de Negocios futuros

En esta sección, daremos a conocer el diagrama que modela al proceso de negocios, que es reconocido generalmente como BPMN por las siglas en ingles correspondientes a Bussiness Process Modeling Notation. Este diagrama consta en mostrar los procesos que se generan durante la ejecución de este proceso, en el que se inicia la aplicación y esta interactúa con otras entidades, que en este caso corresponde a Usuario y Servidores. El formato del diagrama de proceso de negocios es realizado en flujo de datos para ayudar al lector a entender el proceso de una manera más fácil y comprensible.

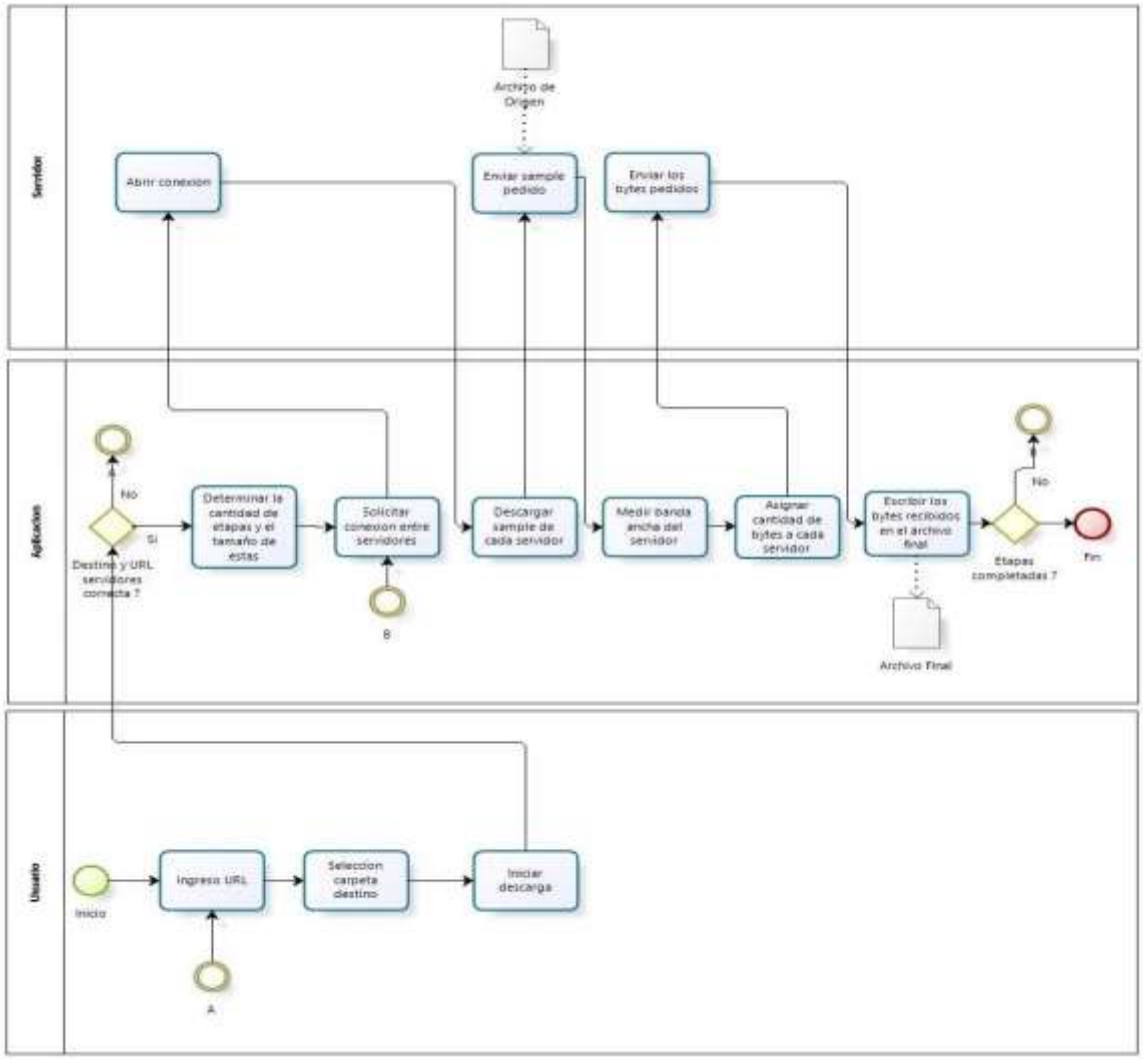


Ilustración 6 Diagrama Proceso de Negocios BPM

9.2 Diagrama de Flujo de Datos

En los siguientes diagramas de presentaras los distintos flujos de datos que interactúan con cada uno de los procesos pertenecientes al sistema y las entidades externas a este, los cuales son usuario y servidor.

El diagrama de flujo de datos o DFD, es constituido por 3 niveles. El primer nivel es llamado por diagrama de contexto, que simplemente es una vista amplia del sistema en un solo proceso. El segundo nivel es llamado por diagrama superior, que consta en dividir el sistema en sub procesos que son de importancia dentro del sistema. Por último, queda el tercer nivel que es llamado por diagrama de detalle, que su finalidad es detallar cada sub proceso que se identificó en el diagrama superior con un DFD.

9.2.1 Diagrama de Flujo de Datos nivel contexto

Este nivel está formado por un solo proceso que representa a todo el sistema. Este proceso se identifica con el número cero. La finalidad de este nivel es de dar a conocer al lector la interacción entre sistema y las entidades externas, dando a ver los flujos entrantes y salientes al sistema, sin mostrar los sub procesos que el sistema realiza.

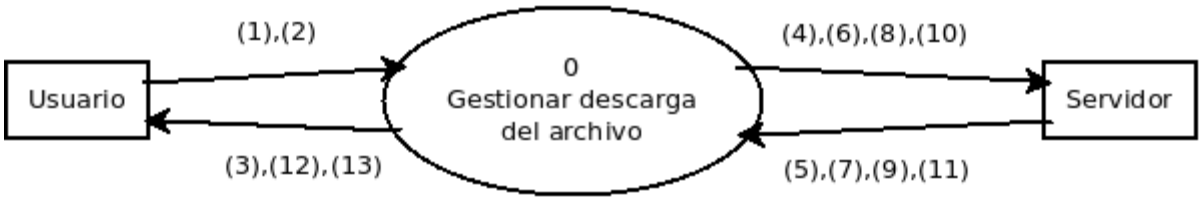


Ilustración 7 Dfd nivel de contexto

En donde:

(1): URL
(2): Solicitud inicio descarga
(3): Respuesta solicitud inicio descarga
(4): Solicitud de conexión puerto 80
(5): Respuesta solicitud conexión puerto 80
(6): Solicitud conexión de datos y puerto servidor

(7): Respuesta conexión de datos y puerto servidor
(8): Solicitud sample 10mb
(9): Respuesta solicitud sample 10mb
(10): Solicitud envió bytes de datos
(11): Respuesta solicitud envió de bytes de datos
(12): Información de tiempos de descarga y diferencia entre servidores
(13): Ruta destino del archivo

9.2.2 Diagrama de flujo de datos nivel superior

Este nivel tiene como objetivo especificar los procesos más importantes que se realizan en el sistema. Estos procesos se identifican desde el número 1 hasta N (donde N es el número de procesos identificados). En este nivel se podrán ver nuevos flujos que se forman por la interacción entre los distintos procesos que son parte del sistema.

1. Comprobar y gestionar datos ingresados: Se encarga de definir en que afecta la interfaz gráfica o GUI, que en este caso solicita los datos que se necesitan para poder dar comienzo al proceso de descarga y verifica que estos datos estén correctos.

2. Gestionar envió de comandos HTTP: Se encarga que de que se abra un socket por el puerto 80 para conectarse al servidor, para iniciar, pausar, reanudar y cancelar la descarga y recibir los datos a descargar por el puerto.

3. Gestionar etapas y tiempos: Se encarga de llevar a cabo una coordinación de la descarga. Control absoluto sobre etapas, repartición de bytes, asignación de tamaños y cantidad de etapas en que se hará la descarga y coordinación de hilos (threads) que se llevan a cabo de forma paralela.

4. Gestionar transferencia de datos: Se encarga de recibir los datos que se descargan, teniendo un control a lo que respecta a rangos de bytes que se obtienen en cada etapa y luego almacenarlos en el archivo final.

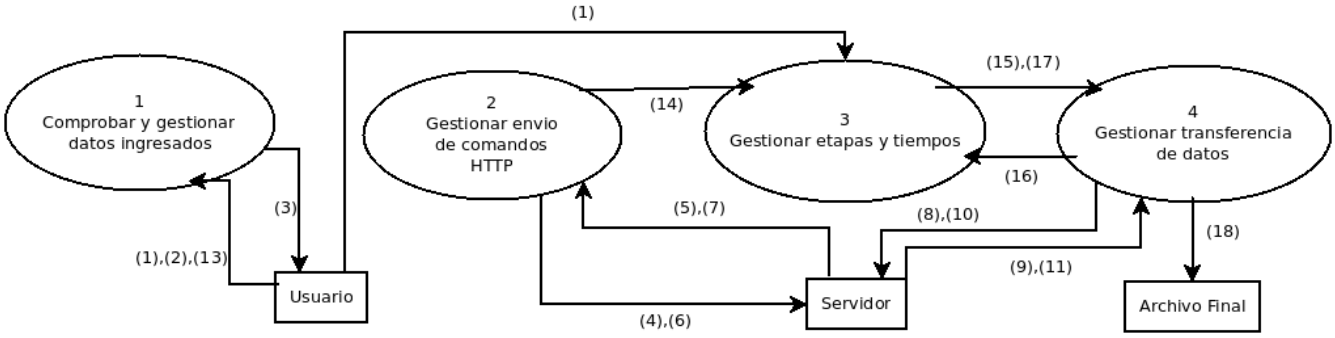


Ilustración 8 Dfd Nivel Superior

En donde:

(1): URL

(2): Solicitud inicio descarga

(3): Respuesta solicitud inicio descarga

(4): Solicitud de conexión puerto 80

(5): Respuesta solicitud conexión puerto 80

(6): Solicitud conexión de datos y puerto servidor

(7): Respuesta conexión de datos y puerto servidor

(8): Solicitud sample 10mb

(9): Respuesta solicitud sample 10mb

(10): Solicitud envió bytes de datos

(11): Respuesta solicitud envió de bytes de datos

(12): Información de tiempos de descarga y diferencia entre servidores

(13): Ruta destino del archivo

(14): Asentimiento de inicio y cantidad de servidores

(15): Solicitud descarga de sample por cada servidor

(16): Tiempos de descarga de sample por cada servidor

(17): Cantidad de bytes a descargar por cada servidor

(18): Bytes descargados

9.2.3 Diagrama de flujo de datos nivel detalle

En este nivel, cada proceso que se identificó en el nivel superior se debe realizar un DFD de detalle que muestre las interacciones de cada proceso. Estos procesos se identifican de la siguiente forma, el primer número corresponde al proceso al cual corresponden en el nivel superior y luego al que corresponde dentro del proceso a detallar. No se pueden crear nuevos flujos a las entidades y procesos distintos a los que se realizaron en el diagrama de nivel superior.

- Proceso 1: Comprobar y gestionar datos ingresados

  1.1 Comprobar el ingreso de los datos requeridos: Se encarga de validar las URL.

  1.2 Extraer datos desde las URL y enviarlos: Se encarga de identificar que contiene la URL para enviarlos al programa.

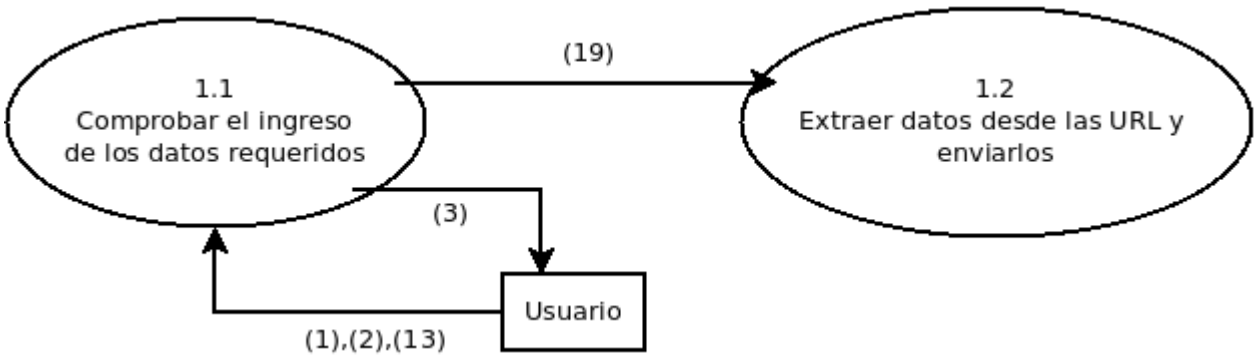


Ilustración 9 Dfd Nivel Detalle 01

En donde:

(1): URL
(2): Solicitud inicio descarga
(3): Respuesta solicitud inicio descarga
(13): Ruta destino del archivo
(19): URL comprobada

- Proceso 2: Gestionar envió de comandos HTTP

  2.1 Gestionar conexión de control: Se encarga de abrir y cerrar el socket java de control, esto se realiza a través del puerto 80.

  2.2 Gestionar conexión de datos: Se encarga de abrir y cerrar el socket java de datos, que sería el puerto dedicado para la descarga.

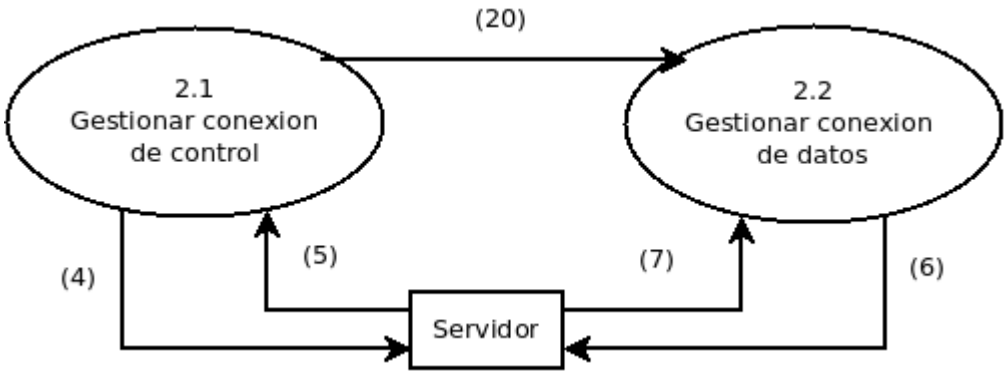


Ilustración 10 Dfd Nivel Detalle 02

En donde:

(4): Solicitud de conexión puerto 80
(5): Respuesta solicitud conexión puerto 80
(6): Solicitud conexión de datos y puerto servidor
(7): Respuesta conexión de datos y puerto servidor
(20): Asentamiento para enviar comando de modo pasivo y crear conexión

- Proceso 3: Gestionar etapas y tiempos

  3.1 Calcular la cantidad de etapas: Se encargar de determinar las veces en que se va a dividir el proceso de descarga del archivo.

  3.2 Calcular la cantidad de Bytes de cada servidor: Se encarga de recibir el tiempo de descarga de la muestra que se piden a cada servidor para asignar los bytes de manera proporcional a la tasa de transferencia de descarga por cada servidor.

3.3 Calcular el tamaño de cada etapa y llevar el conteo de etapas: Se encarga de ver el tamaño de cada división a descargar y llevar el un control para saber que etapa se ha descargado.

3.4 Contabilizar el tiempo total de ejecución y la diferencia de tiempo entre servidores: Calcula el tiempo que demoro cada servidor en descargar la muestra o etapa, además del tiempo que demoro finalizar la descarga del archivo de manera completa.

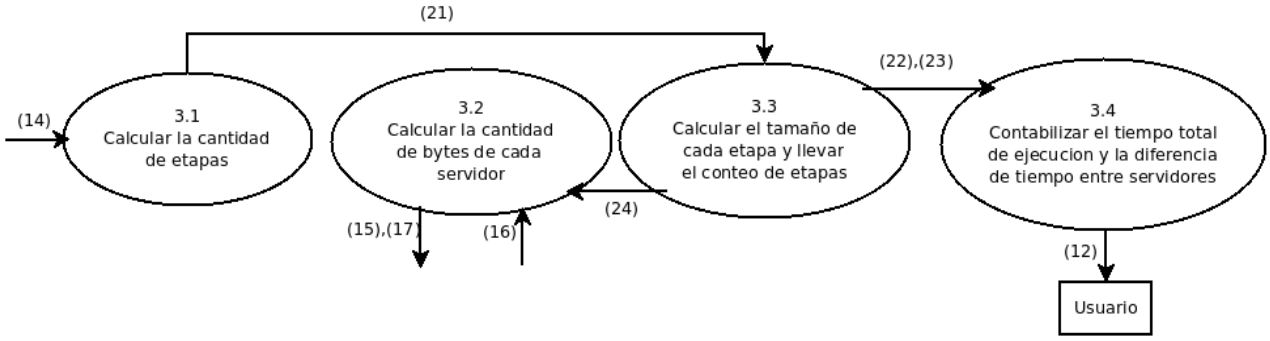


Ilustración 11 Dfd Nivel Detalle 03

En donde:

(12): Información de tiempos de descarga y diferencia entre servidores
(14): Asentimiento de inicio y cantidad de servidores
(15): Solicitud descarga de sample por cada servidor
(16): Tiempos de descarga de sample por cada servidor
(17): Cantidad de bytes a descargar por cada servidor
(21): Numero etapas
(22): Asentimiento número de etapas
(23): Asentimiento termino número de etapas
(24): Tamaño etapa

- Proceso 4: Gestionar transferencia de datos

4.1 Gestionar la medición de banda ancha de cada servidor: Se encarga de descargar la muestra de datos de cada servidor, teniendo el tiempo que demoro esta descarga.

4.2 Gestionar recepción de datos: Se encarga de la recepción de los datos a descargar y los guarda en el archivo final.

4.3 Solicitar una cantidad de bytes a cada servidor: Se encarga del buffer de datos, en el que se espera llenar el buffer con los bytes a descargar por cada servidor y luego pasa estos datos descargados al disco para así seguir recibiendo más bytes entrantes.

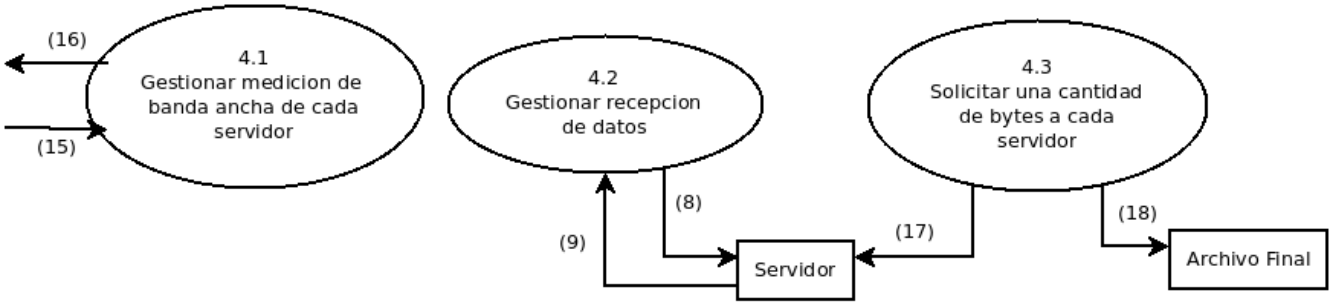


Ilustración 12 Dfd Nivel Detalle 04

En donde:

(8): Solicitud sample 10mb
(9): Respuesta solicitud sample 10mb
(15): Solicitud descarga de sample por cada servidor
(16): Tiempos de descarga de sample por cada servidor
(17): Cantidad de bytes a descargar por cada servidor
(18): Bytes descargados

9.3 Diagrama de casos de uso

En el contexto del Lenguaje de Modelado Unificado, el diagrama de casos de uso es un manera de mejorar el diagrama de comportamiento UML. Este lenguaje, que también es conocido por la abreviación UML da a conocer una notación grafica para representar casos de uso, que es llamada como modelo de casos de uso.

UML no ha especificado estándares para dar una forma escrita que muestre de una manera mejor los casos de uso, es por esto que hay gente que no puede entender que esta forma gráfica de representación aclara la naturaleza de un caso de uso o un conjunto de estos. Estos dos conceptos nombrados anteriormente (caso de uso y diagrama de casos de uso) están relacionados, pero, los casos de uso se centran mucho más en el detalle que los diagramas de casos de uso. En otras palabras, se deben detallar dos o más casos de uso para poder llegar a conocer cuál es la función de solo un caso de uso.

En el siguiente diagrama de casos de uso, se representa la interacción del usuario de la aplicación con los requerimientos funcionales que se han presentado en el ítem anterior.

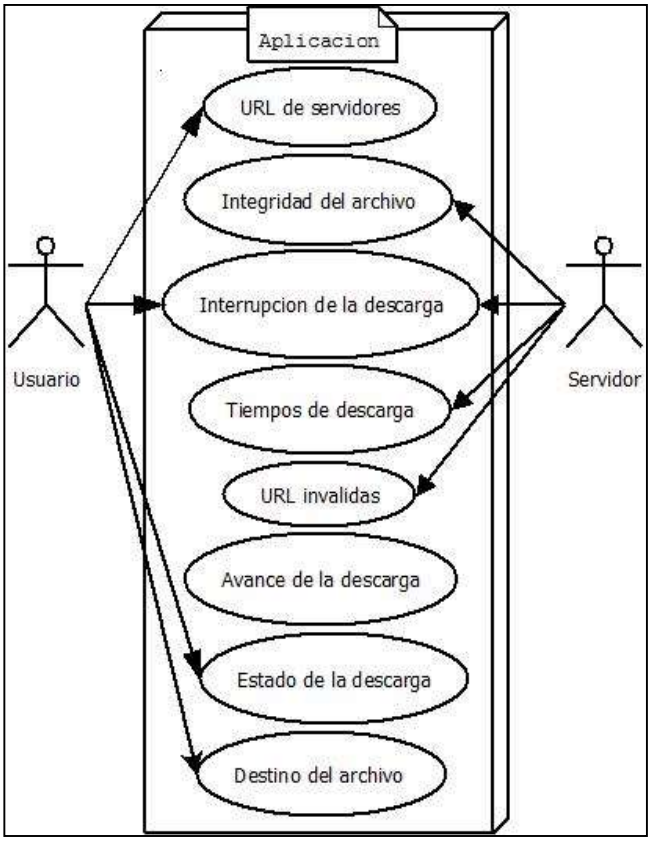


Ilustración 13 Caso de Uso general

10 Diseño Interfaz y Navegación

10.1 Diseño Interfaz gráfica del usuario

Se considera el diseño de la interfaz del escenario principal del programa.

Escenario Principal

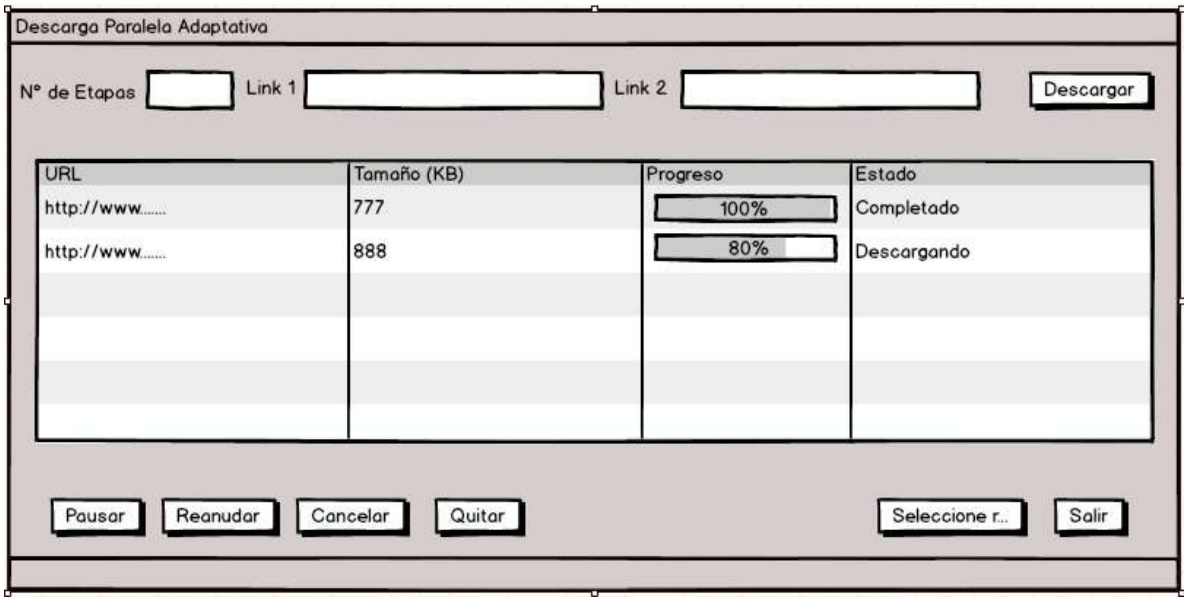


Ilustración 14Wireframe aplicación escenario principal

En el sector superior, consta de las opciones para el ingreso de texto, el primer cuadro pertenece al Numero de etapas en las cuales se procederá a evaluar el estado de la red y reasignar el tamaño de la descarga, el segundo y tercer recuadro es para el ingreso de URL de donde se encuentra el archivo, botón de descarga que es aquel que inicia el proceso.

En el sector central se puede ver una tabla en la cual se despliega la información del proceso. En el primer campo estarán las URL de descarga, segundo se encuentra el tamaño en KB de la etapa que se encuentra descargando, el tercer campo es el progreso de descarga en la cual se muestra una barra de progreso y finalmente el estado de la descarga, que será completado, pausado, cancelado y completado

En el sector inferior está compuesta de botones que afectan el estado de descarga, tenemos 6. Pausar, que pausa una descarga activa. Reanudar, activa una descarga pausada. Cancelar, que cancela una descarga, y quitar, que borra de la tabla la descarga. Y por último los botones de Seleccionar ruta de descarga, el cual el usuario selecciona donde quiere guardar el archivo a descargar y el botón salir, que es para cerrar el programa.

11 Pruebas Simulación

Para el desarrollo de las pruebas de rendimiento de la simulación en NS3, se realizan varios experimentos con diferentes números de etapas y diferentes calidades de enlace. Primero, comenzaremos definiendo la topología de red utilizada en esta simulación, y los valores de cada etapa transmitidos.

11.1 Topología de Red

La simulación estará compuesta por 7 Nodos, los cuales representarán a 3 servidores, 1 router y 3 clientes.

Solo se realizarán envíos al cliente N°1, desde los 3 servidores, pasando por el enlace del router hasta llegar a nuestro Nodo Cliente.

En la siguiente imagen se ve cómo está reflejada la topología en NS3.

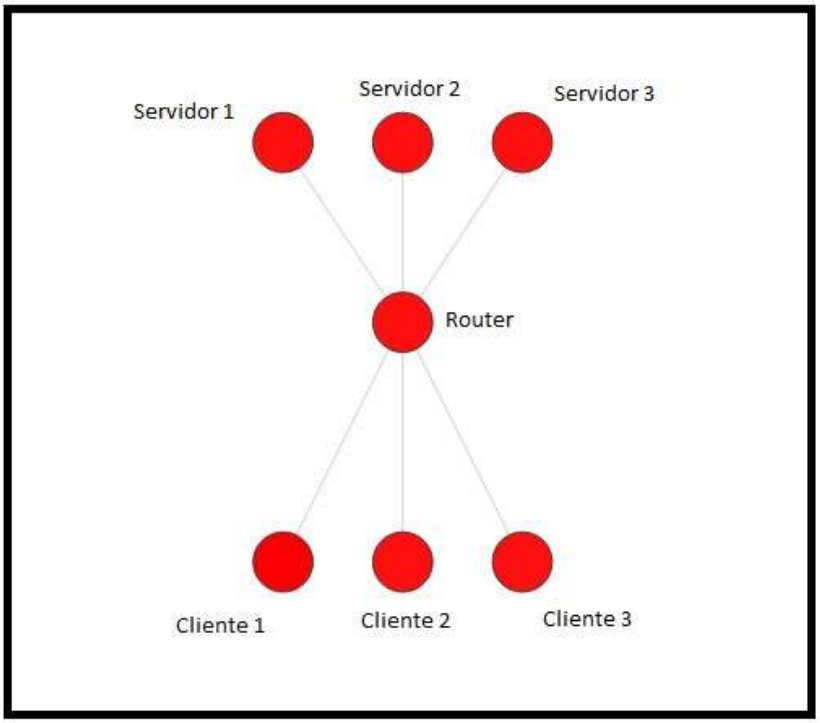


Ilustración 15 Imagen de topología en NS3

11.2 Información de Experimentos

Simularemos envíos de paquetes desde los 3 servidores, hacia un nodo cliente. Esto se realizará de dos formas diferentes, Paquetes con tamaños Dinámicos y Paquetes con tamaños Estáticos.

Cada simulación constará de un número de etapas definido y una velocidad de transmisión y delay, para ambos escenarios, idéntica. El tamaño final de cada simulación será el mismo para todas, aproximadamente 999Mb. Las etapas serán de 10 y 20 iteraciones. Cada una de estas etapas transmitirá 102297kilobytes (99 Mb aprox).

Nuestro primer experimento, será detallado de forma intensiva, para lograr visualizar y entender las variables significativas que intervienen en las situaciones, tamaños de los paquetes y los tiempos. Luego, las simulaciones siguientes, se detallarán con cuadros resúmenes para poder ver solamente los resultados.

11.3 Experimento N°1

La siguiente tabla muestra los datos que conforman la información de los enlaces de la topología, válida para los escenarios con envío de paquetes dinámicos y estáticos.

Topología		
Enlace S1 a R1	DataRate 15 Mbps	Delay 70 ms
Enlace S2 a R1	DataRate 15 Mbps	Delay 80 ms
Enlace S3 a R1	DataRate 15 Mbps	Delay 70 ms
Enlace R1 a C1	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C2	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C3	DataRate 5Mbps	Delay 6 ms

Tabla 3 Topología de Red Experimento 1

11.3.1 Simulación de Red P2P con Paquetes Dinámicos

Al inicio de la simulación, se define en la iteración 0, un paquete con tamaño idéntico para cada servidor de la red. Es decir, un valor definido que solo será válido en esta etapa. Luego, al finalizarla

transferencia, permitirá analizar el tiempo de descarga de cada servidor, para luego realizar una estimación de la tasa de descarga que será asignada en las etapas siguientes.

Luego, cada ciclo de iteración, dependiendo del tiempo de descarga del paquete anterior, se evaluará la calidad de transmisión y asignará el tamaño del paquete próximo en la nueva iteración. Dependiendo de la estabilidad de la descarga, se logrará aprovechar el servidor que presenta un tiempo menor en relación a los otros servidores en el tiempo de transmisión del paquete.

11.3.2 Desarrollo de Etapas Dinámicas

Etapa 0

Se asignan valores por defecto para transmitir en la red. Cada servidor transmitirá un paquete de tamaño 34099 kilobytes (33 Mb) hacia el Cliente 0.

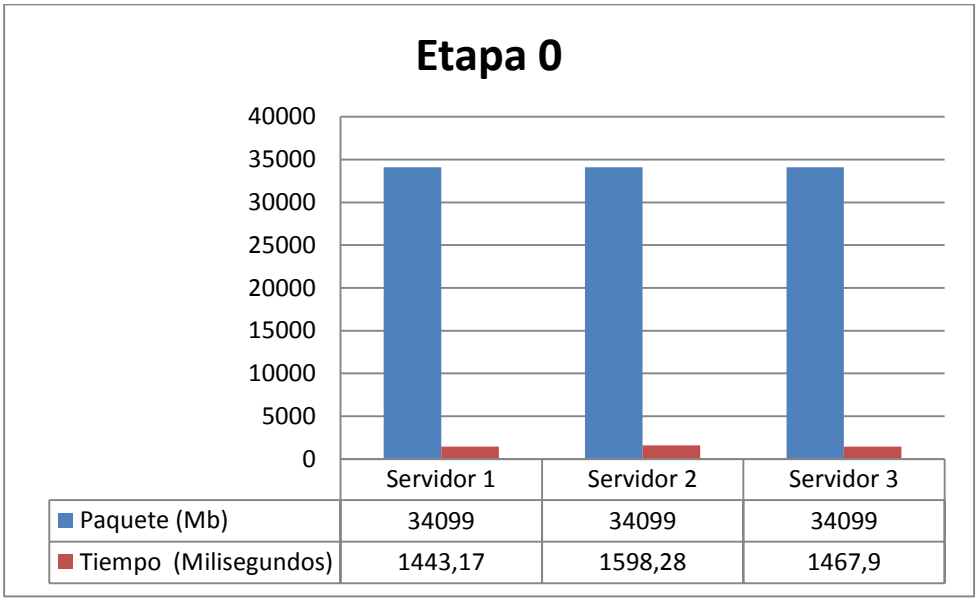


Ilustración 16 Etapa 0 Escenario Dinámico

Como se puede apreciar, en esta primera etapa, lo que se pretende es obtener los tiempos, para luego proceder a ejecutar un algoritmo de estimación para definir el tamaño del paquete para cada servidor en el próximo ciclo.

Se utilizan las fórmulas definidas en el Capítulo 6 “Definición del proyecto”.

Etapa 1

Basándose en los resultados de la primera iteración, se definen los nuevos paquetes y se vuelve a definir el tamaño para la próxima iteración, este proceso se repetirá hasta llegar al límite de etapas definidos en la simulación.

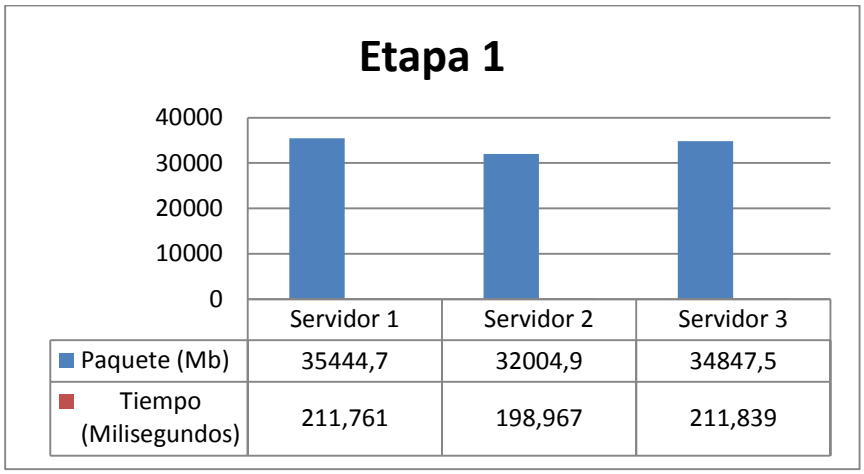


Ilustración 17 Etapa 1 Escenario Dinámico

Etapa 2

Como el servidor 2, obtuvo un tiempo de descarga menor en relación a los otros dos servidores en la etapa anterior, el tamaño a descargar en esta iteración es mayor al anterior.

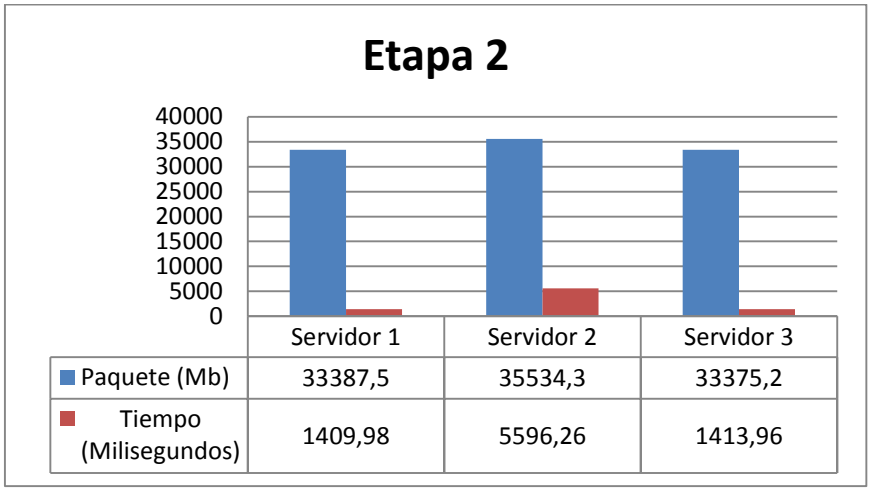


Ilustración 18 Etapa 2 Escenario Dinámico

Esto continúa sucesivamente, a modo de resumen se representan los gráficos de las etapas siguientes:

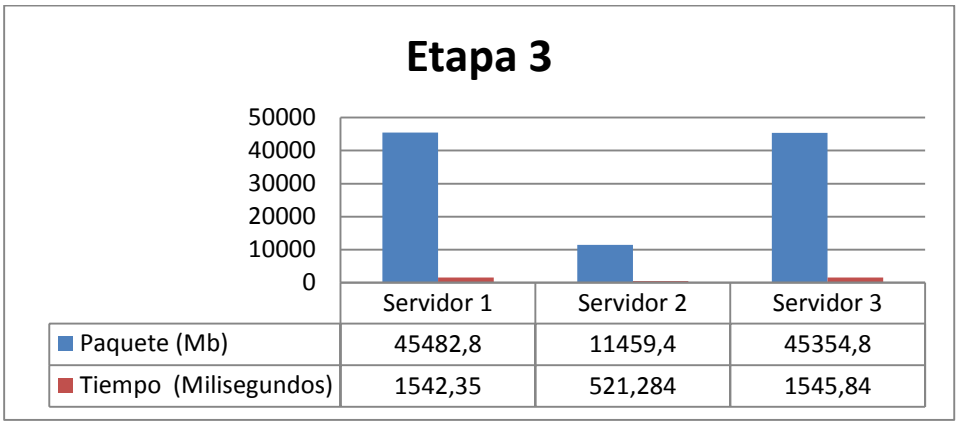


Ilustración 19 Etapa 3 Escenario Dinámico

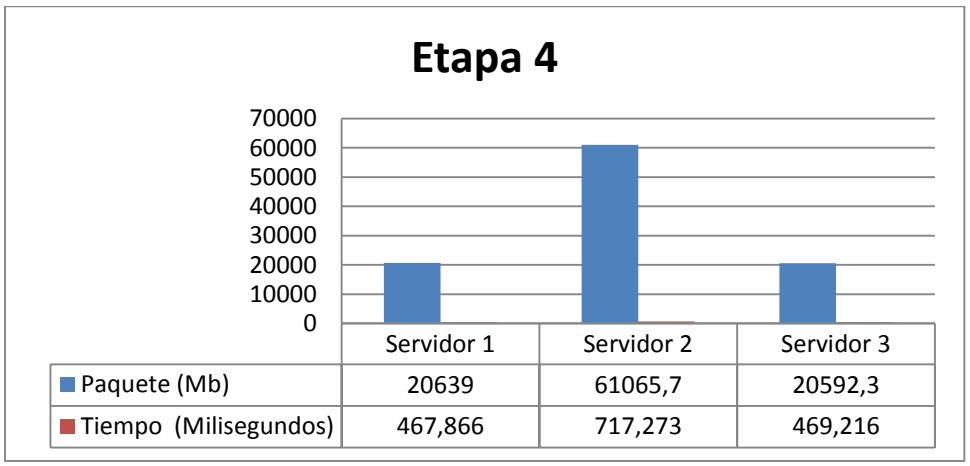


Ilustración 20 Etapa 4 Escenario Dinámico

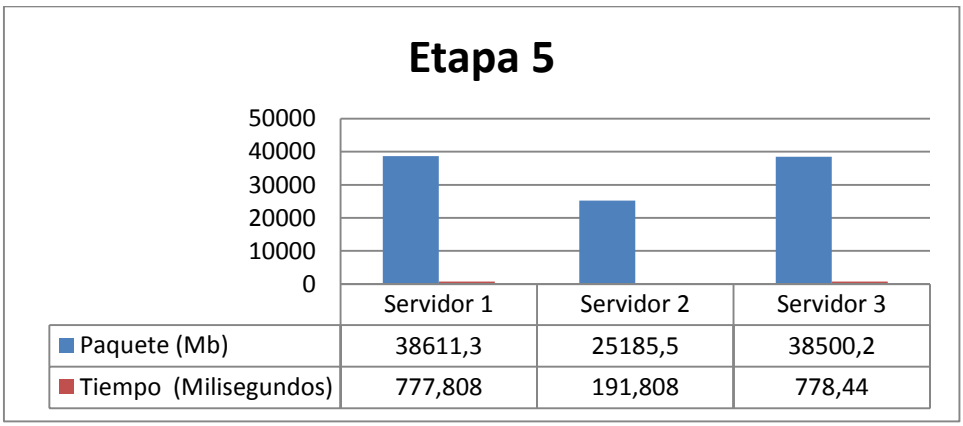


Ilustración 21 Etapa 5 Escenario Dinámico

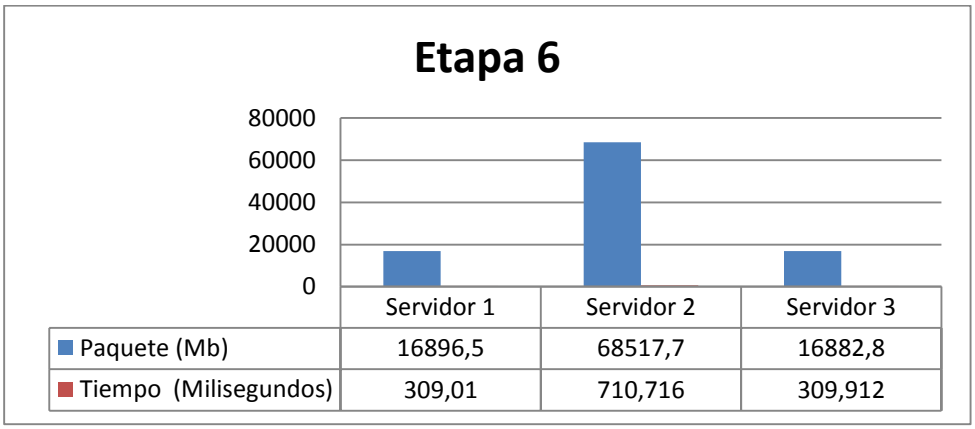


Ilustración 22 Etapa 6 Escenario Dinámico

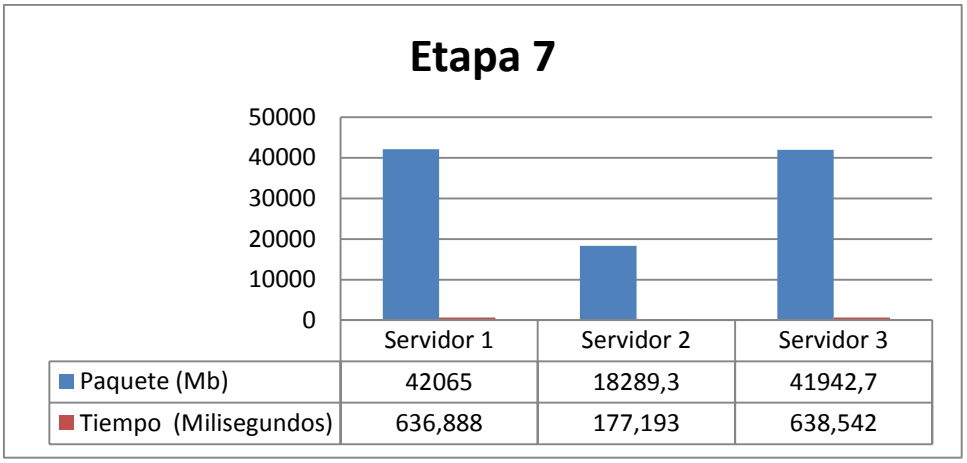


Ilustración 23 Etapa 7 Escenario Dinámico

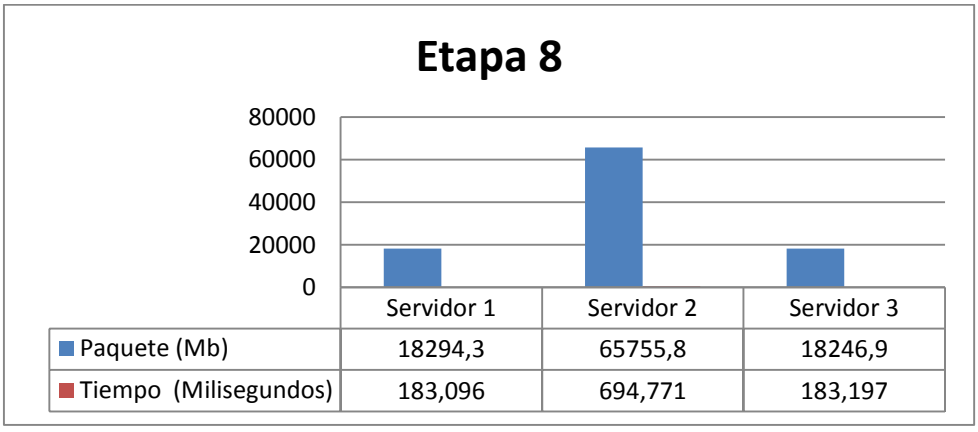


Ilustración 24 Etapa 8 Escenario Dinámico

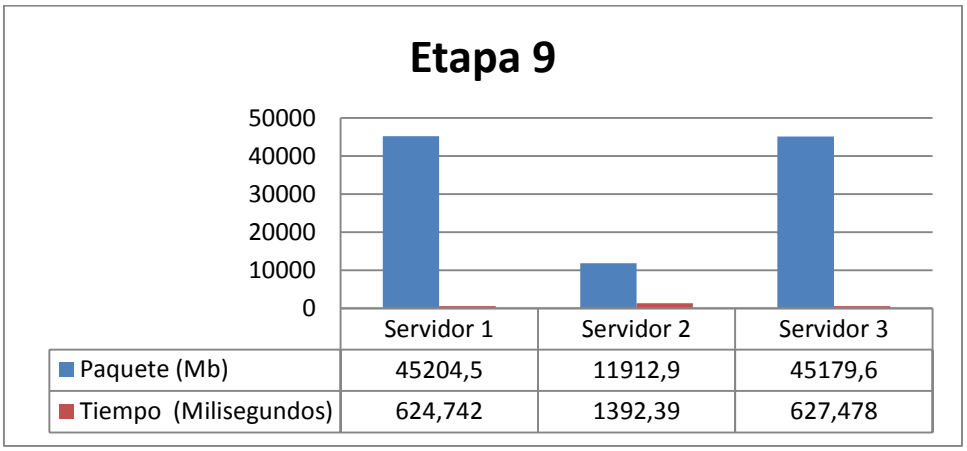


Ilustración 25 Etapa 9 Escenario Dinámico

En cada etapa, se puede apreciar la variación de los tamaños de los paquetes en factor al tiempo obtenido en la iteración anterior a la etapa.

11.3.3 Simulación de Red P2P con Paquetes Estáticos

Tomado en cuenta un escenario donde los paquetes están definidos del mismo tamaño en cada iteración, sin evaluar la capacidad de rendimiento de estos ni el tiempo que cada uno tardará en transmitir paquetes, se realizó una representación de esto en el mismo simulador. Se definió el tamaño de igual forma de transmisión por etapa de la simulación anterior (escenario Dinámico).

Como el tamaño de los paquetes a representar, es la misma en cada iteración, no es necesario graficar la variación de estos ya que no presenta ningún cambio.

Lo que si nos interesa, son los tiempo en terminar cada transmisión. La siguiente tabla, nos muestra la representación resumida de los tiempos por etapas del experimento.

PAQUETES ESTATICOS			
	Servidor 1	Servidor 2	Servidor 3
Etapa 0	1443,17	1598,28	1467,9
Etapa 1	206,633	202,588	208,155
Etapa 2	1414,19	5795,83	1417,56
Etapa 3	1232,35	2608,76	1234,8
Etapa 4	784,781	1217,45	787,556
Etapa 5	636,082	873,684	635,756
Etapa 6	617,588	701,286	618,511
Etapa 7	484,61	535,36	485,207
Etapa 8	473,513	530,737	474,438
Etapa 9	464,264	889,166	465,784
Total	7757,181	14953,141	7795,667

Tabla 4 Tabla representativa Tiempos Escenario Estático

11.4 Comparación tiempo Simulación Estática v/s Simulación Dinámica

11.4.1 Simulación Dinámica

A continuación, se presenta la tabla que contiene datos de cada etapa asociada a cada servidor. Esto permite representar de forma visual la variación existente entre cada uno de los tiempos de envíos por etapa.

PAQUETES DINÁMICOS			
	Servidor 1(ms)	Servidor 2(ms)	Servidor 3(ms)
Etapa 0	1443,17	1598,28	1467,9
Etapa 1	211,761	198,967	211,839
Etapa 2	1409,98	5596,26	1413,96
Etapa 3	1542,35	521,284	1545,84
Etapa 4	467,866	717,273	469,216
Etapa 5	777,808	191,808	778,44
Etapa 6	309,01	710,716	309,912
Etapa 7	636,888	177,193	638,542
Etapa 8	183,096	694,771	183,197
Etapa 9	624,742	1392,39	627,478

Tabla 5 Tabla representativa Tiempos Simulación Dinámica por etapas

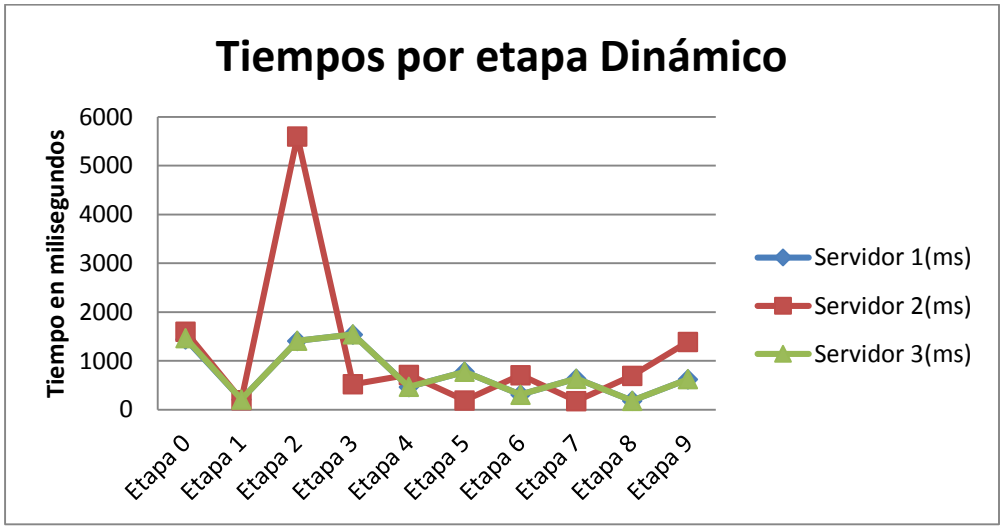


Ilustración 26 Gráfico representativo tiempo por etapa de cada servidor Dinámico

Por medio de la ilustración 13, se puede distinguir de forma visual las diferentes variaciones que experimentan los paquetes que a medida de cada evaluación, van cambiando, adaptándose a la cantidad necesaria según la fórmula que los redefinirá en cuanto a tamaño.

11.4.2 Simulación Estática

La simulación estática nos entrega los siguientes datos, resumidos en una tabla por etapa y servidor:

PAQUETES ESTÁTICOS			
	Servidor 1(ms)	Servidor 2(ms)	Servidor 3(ms)
Etapa 0	1443,17	1598,28	1467,9
Etapa 1	206,633	202,588	208,155
Etapa 2	1414,19	5795,83	1417,56
Etapa 3	1232,35	2608,76	1234,8
Etapa 4	784,781	1217,45	787,556
Etapa 5	636,082	873,684	635,756
Etapa 6	617,588	701,286	618,511
Etapa 7	484,61	535,36	485,207
Etapa 8	473,513	530,737	474,438
Etapa 9	464,264	889,166	465,784

Tabla 6 Tabla representativa Tiempos Simulación Estática por etapas

Al poner atención a los tiempos expuestos, podemos darnos cuenta en relación a sus variaciones de tiempo, estos presentan en balance más definido, donde no existen tantos cambios bruscos en relación al tiempo, a excepción del servidor 2 en la etapa 2, donde aumenta considerablemente en relación a los otros dos servidores

Luego de evaluar los datos anteriores, se presenta el gráfico que contiene la información antes señalada.

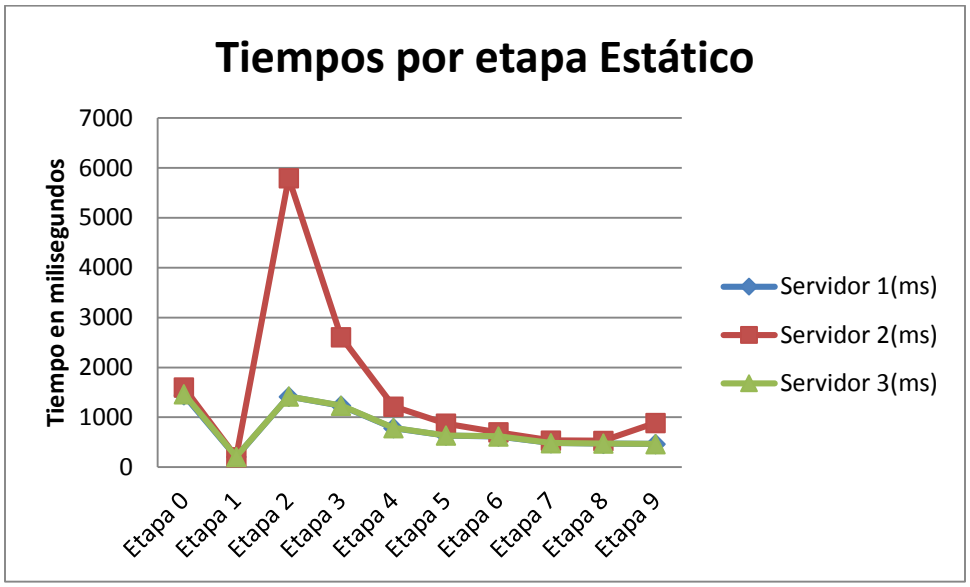


Ilustración 27 Gráfico representativo tiempo por etapa de cada servidor Estático

Como se aprecia en el gráfico, en comparación a la simulación Dinámica (Ilustración 13), presenta un desarrollo por etapa más estable, pero no significa que tenga mejores tiempos, esa es la diferencia más visible. Los puntos por algunas etapas presentan un mayor tiempo de descarga, generando una demora en la transmisión de los paquetes.

11.4.3 Comparación tiempos finales Estático vs Dinámico

La siguiente tabla resume los tiempos finales y la ganancia de segundos antes mencionada.

Comparación Tiempos Finales			
	Servidor 1(ms)	Servidor 2(ms)	Servidor 3(ms)
Simulación Dinámica	7606,671	11798,942	7646,324
Simulación Estática	7757,181	14953,141	7795,667

Diferencia Simulación Dinámica	150,51	3154,199	149,343

Tabla 7 Tiempos Finales Estático vs Dinámico

Graficando obtenemos la siguiente figura, la cual nos permite decidir que método genera mayor ganancia.

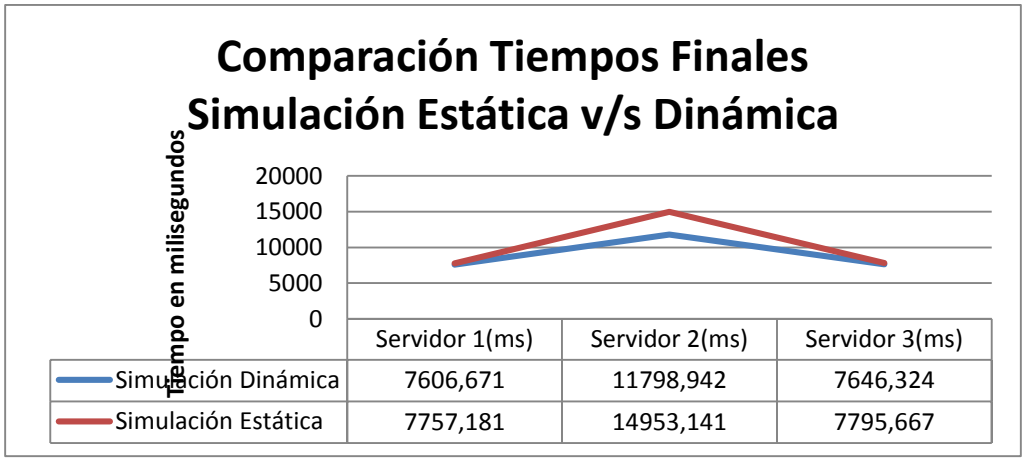


Ilustración 28 Gráfico Comparación Tiempos Finales Simulación Estática vs Dinámica

Al apreciar los gráficos y comparar los tiempos finales, podemos darnos cuenta de una mejora de una optimización de aproximadamente 3.1 segundos.

11.5 Experimento N°2

Ya sabemos cómo trabajan los métodos y las diferencias entre los envíos de paquetes dinámicos y estáticos, por lo que ahora, nos limitaremos a mostrar datos de tiempos finales de los dos tipos de simulaciones.

Se describe la topología y las tablas con los tiempos correspondientes.

Número de Etapas: 10

Total Transmitido por etapa: 102297 kilobytes

13.5.1 Escenario Estático

Topología		
Enlace S1 a R1	DataRate 15 Mbps	Delay 90ms
Enlace S2 a R1	DataRate 15 Mbps	Delay 90 ms
Enlace S3 a R1	DataRate 15 Mbps	Delay 90 ms
Enlace R1 a C1	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C2	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C3	DataRate 5Mbps	Delay 6 ms

Tabla 8 Descripción topología experimento N°2

Con la topología y enlaces definidos anteriormente, estos son los resultados de la simulación de envío de paquetes estáticos con NS3.

PAQUETES ESTÁTICOS			
	Servidor 1(ms)	Servidor 2(ms)	Servidor 3(ms)
Etapa 0	1758,28	1774	1789,72
Etapa 1	272,527	274,049	275,571
Etapa 2	3787,08	2033,21	2328,36
Etapa 3	2910,61	799,782	790,534
Etapa 4	1359,29	786,834	618,482
Etapa 5	983,747	615,379	612,932
Etapa 6	791,458	603,355	599,062
Etapa 7	622,995	590,736	587,038
Etapa 8	606,35	440,88	440,552
Etapa 9	598,137	439,629	435,331

Tabla 9 Tiempos de Paquetes estáticos Experimento N°2

A continuación, la representación gráfica de los valores obtenidos anteriormente:

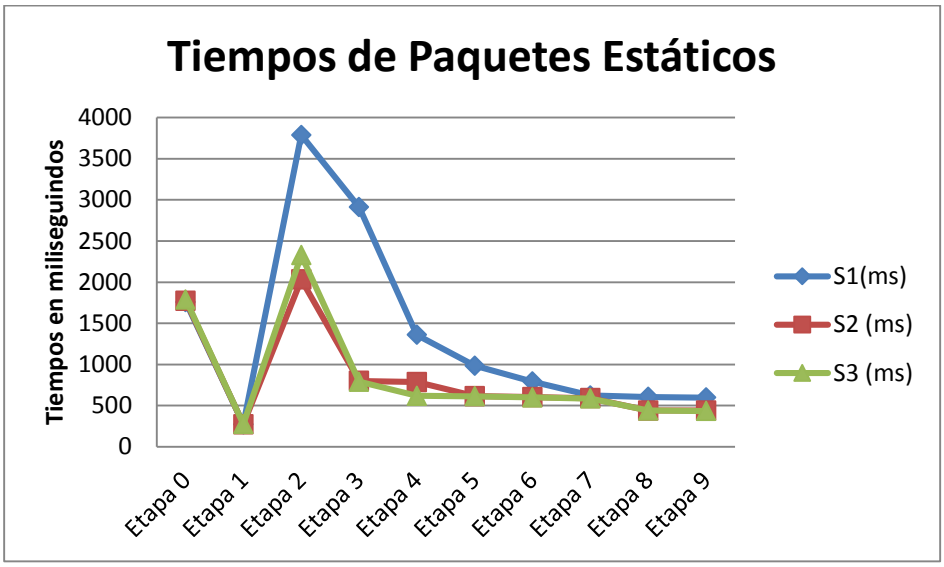


Ilustración 29 Gráfico tiempos Paquetes Estáticos Experimento N°2

11.5.2 Escenario Dinámico

PAQUETES DINÁMICOS			
	Servidor 1(ms)	Servidor 2(ms)	Servidor 3(ms)
Etapa 0	1758,28	1774	1789,72
Etapa 1	277,486	274,374	275,412
Etapa 2	3786,62	2033,48	2328,42
Etapa 3	2139,09	994,578	803,079
Etapa 4	791,635	793,289	987,395
Etapa 5	1359,02	615,2	427,963
Etapa 6	402,273	602,261	812,992
Etapa 7	1175,94	430,046	243,898
Etapa 8	208,45	429,182	811,615
Etapa 9	1171,55	402,603	203,674

Tabla 10 Tiempos de Paquetes Dinámicos Experimento N°2

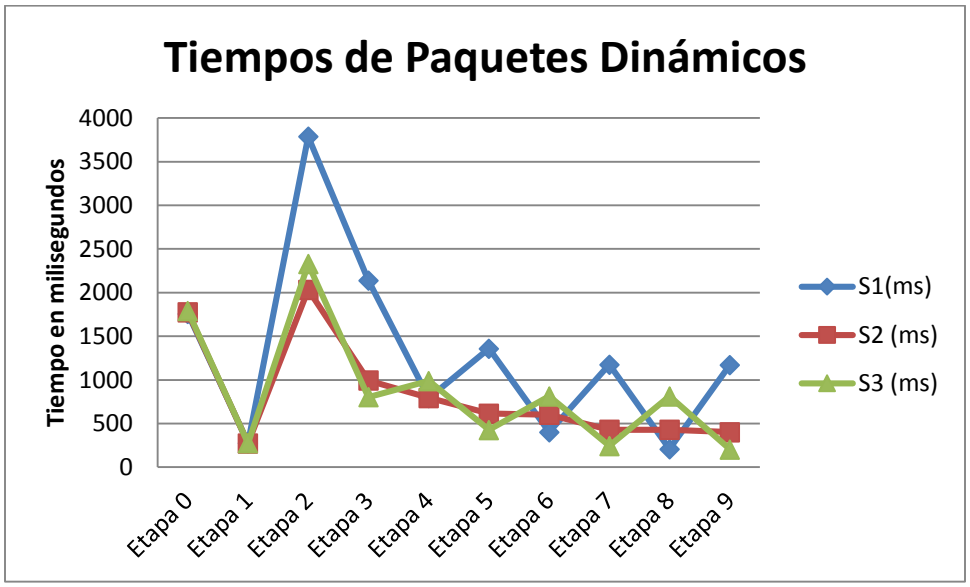


Ilustración 30 Gráfico Tiempos de Paquetes Dinámicos Experimento N°2

11.5.3 Desbalance de Carga

En este experimento, decidimos graficar de igual forma el desbalance de carga de cada prueba, es decir, la diferencia que existe entre el servidor con mejor y menor tiempo por etapa.

Estos son los resultados:

Desbalance de carga Estático		
	Mejor Servidor	Peor Servidor
Etapa 0	1758,28	1789,72
Etapa 1	272,527	275,571
Etapa 2	2033,21	3787,08
Etapa 3	790,534	2910,61
Etapa 4	618,482	1359,29
Etapa 5	612,932	983,747
Etapa 6	599,062	791,458
Etapa 7	587,038	622,995
Etapa 8	440,552	606,35
Etapa 9	435,331	598,137

Desbalance de carga Dinámico		
	Mejor Servidor	Peor Servidor
Etapa 0	1758,28	1789,72
Etapa 1	274,374	277,486
Etapa 2	2033,48	3786,62
Etapa 3	803,079	2139,09
Etapa 4	791,635	987,395
Etapa 5	615,2	1359,02
Etapa 6	402,273	812,992
Etapa 7	243,898	1175,94
Etapa 8	208,45	811,615
Etapa 9	203,674	1171,55

Tabla 11 Tiempos Desbalance Estáticos vs Dinámicos

Como se puede apreciar en las tablas, a partir de la etapa 4 aproximadamente comienza a ocurrir una variación en los tiempos.

Se representa en el siguiente gráfico:

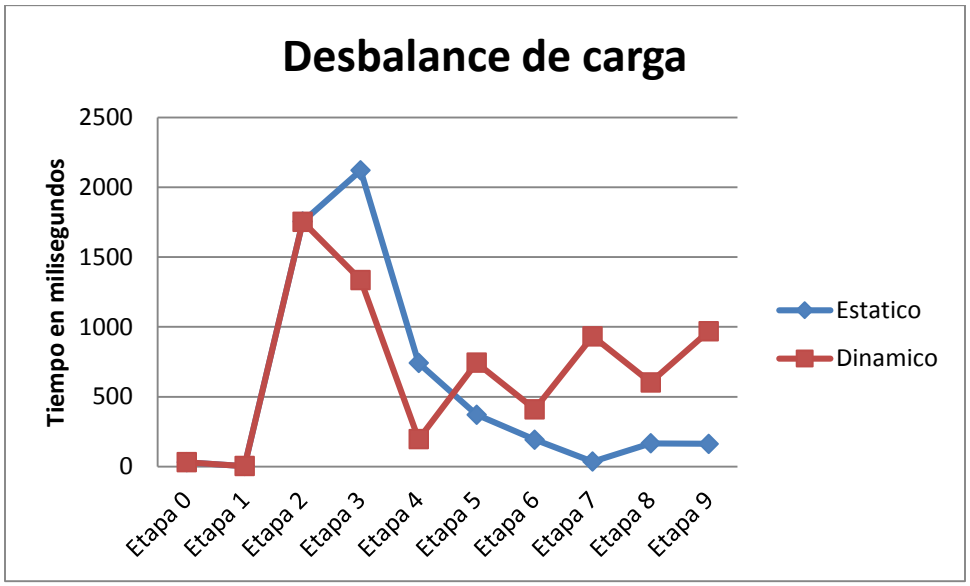


Ilustración 31 Gráfico Desbalance de carga Experimento N°2

11.6 Experimento N°3

Se describe la topología y las tablas con los tiempos correspondientes.

Número de Etapas: 10

Total Transmitido por etapa: 102297 kilobytes

Topología		
Enlace S1 a R1	DataRate 30 Mbps	Delay 150ms
Enlace S2 a R1	DataRate 30 Mbps	Delay 150 ms
Enlace S3 a R1	DataRate 30 Mbps	Delay 150 ms
Enlace R1 a C1	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C2	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C3	DataRate 5Mbps	Delay 6 ms

Tabla 12 Topología Experimento N°3

PAQUETES ESTATICOS			
	S1 (ms)	S2 (ms)	S3(ms)
Etapa 0	2709,55	2719,73	2729,9
Etapa 1	392,362	393,884	395,405
Etapa 2	7502,11	3271,73	3276,35
Etapa 3	4395,84	1261,55	1263,07
Etapa 4	1903,59	974,287	974,232
Etapa 5	1579,33	965,963	965,635
Etapa 6	1269,87	945,618	946,541
Etapa 7	978,91	678,701	679,297
Etapa 8	967,758	677,776	677,448
Etapa 9	949,263	671,303	671,247

PAQUETES DINAMICOS			
	S1(ms)	S2 (ms)	S3 (ms)
Etapa 0	2709,55	2719,73	2729,9
Etapa 1	392,566	394,085	395,402
Etapa 2	7502,35	3271,72	3276,16
Etapa 3	2829,74	1573,58	1574,31
Etapa 4	1584,8	1265,26	1266,14
Etapa 5	1580,12	966,545	967,428
Etapa 6	961,422	967,369	969,162
Etapa 7	1276,6	673,071	674,412
Etapa 8	647,453	950,146	950,922
Etapa 9	1283,89	637,524	638,521

Tabla 13 Tabla de Tiempos Estáticos vs Dinámicos

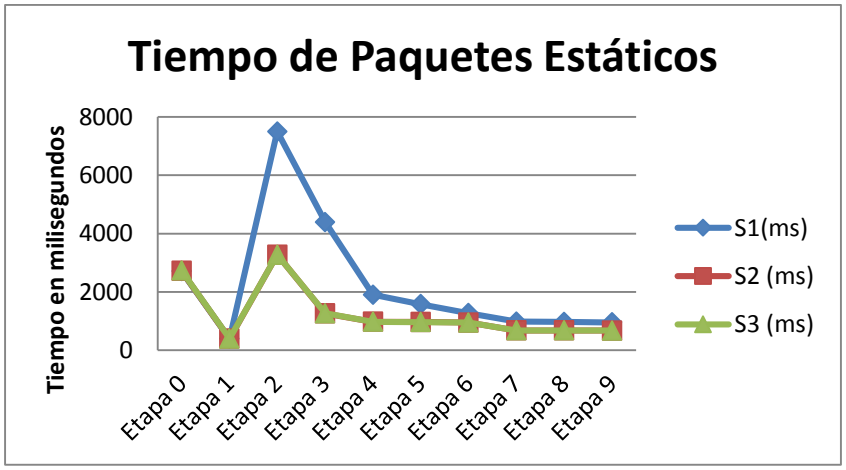


Ilustración 32 Tiempo de Paquetes Estáticos Experimento N°3

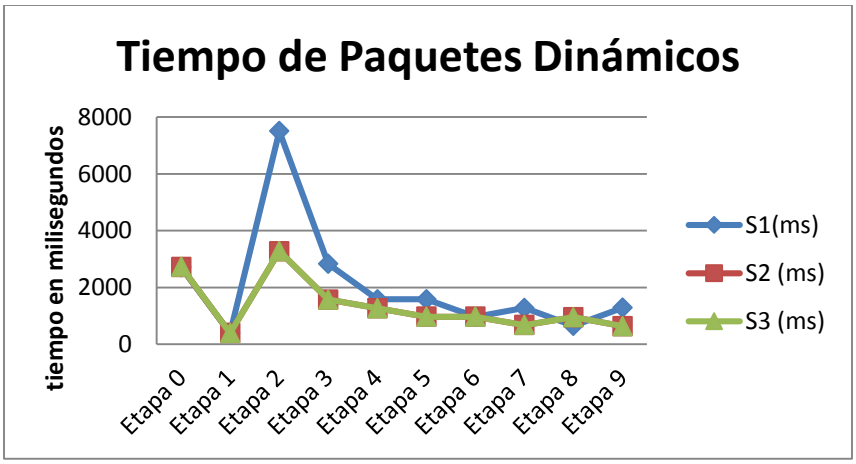


Ilustración 33 Tiempo de Paquetes Dinámicos Experimento N°3

Comparando los tiempos en ambos gráficos, se puede ver una diferencia de tiempo entre la etapa 2 y 4, factor que influye directamente en los paquetes dinámicos siguientes.

A continuación los gráficos de desbalance de carga:

Desbalance de carga Estático		
	Mejor Servidor	Peor Servidor
Etapa 0	2709,55	2729,9
Etapa 1	392,362	395,405
Etapa 2	3271,73	7502,11
Etapa 3	1261,55	4395,84
Etapa 4	974,232	1903,59
Etapa 5	965,635	1579,33
Etapa 6	945,618	1269,87
Etapa 7	678,701	978,91
Etapa 8	677,448	967,758
Etapa 9	671,247	949,263

Desbalance de carga Dinámico		
	Mejor Servidor	Peor Servidor
Etapa 0	2709,55	2729,9
Etapa 1	392,566	395,402
Etapa 2	3271,72	7502,35
Etapa 3	1573,58	2829,74
Etapa 4	1265,26	1584,8
Etapa 5	966,545	1580,12
Etapa 6	961,422	969,162
Etapa 7	673,071	1276,6
Etapa 8	647,453	950,922
Etapa 9	637,524	1283,89

Tabla 14 Desbalance de Carga Dinámico vs Estático Experimento N°3

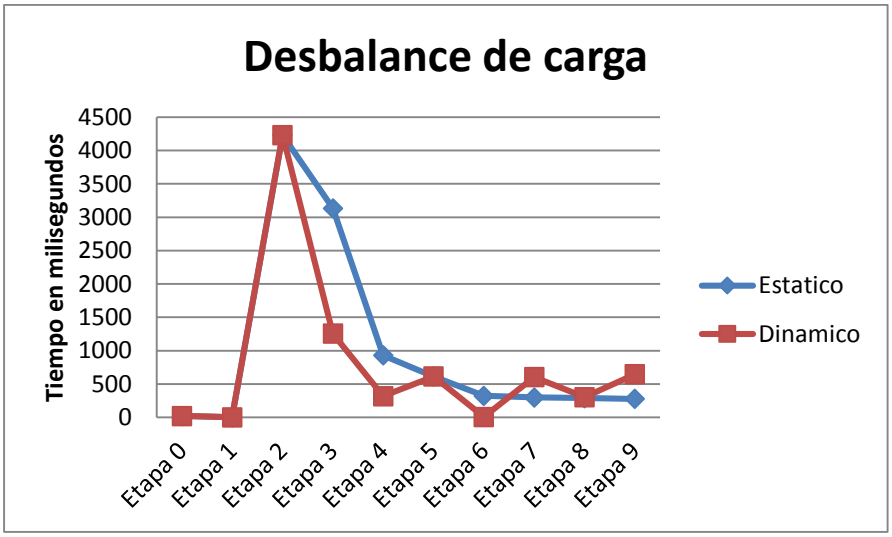


Ilustración 34 Gráfico desbalance de carga Experimento N°3

11.7 Experimento N° 4

Se describe la topología y las tablas con los tiempos correspondientes.

Número de Etapas: 20

Total Transmitido por etapa: 102297 kilobytes

Topología		
Enlace S1 a R1	DataRate 30 Mbps	Delay 150ms
Enlace S2 a R1	DataRate 30 Mbps	Delay 150 ms
Enlace S3 a R1	DataRate 30 Mbps	Delay 150 ms
Enlace R1 a C1	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C2	DataRate 5Mbps	Delay 6 ms
Enlace R1 a C3	DataRate 5Mbps	Delay 6 ms

Tabla 15 Topología de Experimento N°4

Desbalance de carga Estático		
	Mejor Servidor	Peor Servidor
Etapa 0	2709,55	2729,9
Etapa 1	392,362	395,405
Etapa 2	3271,73	7502,11
Etapa 3	1261,55	4395,84
Etapa 4	974,232	1903,59
Etapa 5	965,635	1579,33
Etapa 6	945,618	1269,87
Etapa 7	678,91	978,91
Etapa 8	677,448	967,758
Etapa 9	671,247	949,263
Etapa 10	659,279	882,347
Etapa 11	646,336	4401,45
Etapa 12	649,105	2197,27
Etapa 13	643,505	1583,03
Etapa 14	637,028	1268,95
Etapa 15	630,611	977,985
Etapa 16	388,336	964,985
Etapa 17	386,814	949,261
Etapa 18	386,814	880,497
Etapa 19	384,964	674,999

Desbalance de carga Dinámico		
	Mejor Servidor	Peor Servidor
Etapa 0	2709,55	2729,9
Etapa 1	392,566	395,402
Etapa 2	3271,72	7502,35
Etapa 3	1573,58	2829,74
Etapa 4	1265,26	1584,8
Etapa 5	966,545	1580,12
Etapa 6	961,422	969,162
Etapa 7	673,071	1276,6
Etapa 8	647,453	950,922
Etapa 9	637,524	1283,89
Etapa 10	563,08	688,663
Etapa 11	636,78	971,321
Etapa 12	635,746	664,963
Etapa 13	632,538	679,462
Etapa 14	632,866	658,398
Etapa 15	394,821	657,867
Etapa 16	361,408	648,67
Etapa 17	357,895	961,018
Etapa 18	316,002	654,919
Etapa 19	345,329	964,447

Tabla 16 Desbalance de carga Estático vs Dinámico Experimento N°4

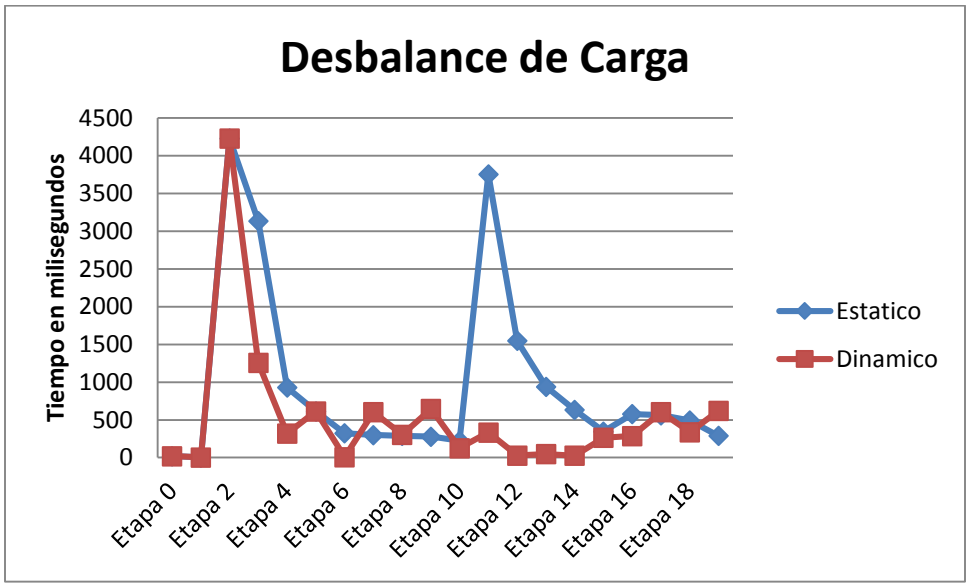


Ilustración 35 Gráfico Desbalance de Carga Experimento N°4

Se puede apreciar a través de gráfico, una variación importante al aumentar el número de etapas del proceso.

11.8 Experimento N°5

En este último experimento, se realizará una comparación para demostrar la diferencia de tiempo que existe entre descargar un archivo desde un solo servidor y las ventajas que existe de descargarlo en una red P2P desde 3 servidores diferentes.

PAQUETES ESTÁTICOS	
	Servidor 1(ms)
Etapa 0	1895,67
Etapa 1	240,128
Etapa 2	4316,95
Etapa 3	1848,07
Etapa 4	1389,03
Etapa 5	1095,33
Etapa 6	4316,95
Etapa 7	1848,07
Etapa 8	1389,02
Etapa 9	1095,33

PAQUETES DINÁMICOS			
	S1 (ms)	S2 (ms)	S3 (ms)
Etapa 0	1438,28	1454	1469,72
Etapa 1	237,419	234,372	235,238
Etapa 2	3065,19	1713,51	2011,46
Etapa 3	1849,4	795,003	637,775
Etapa 4	618,542	635,817	792,38
Etapa 5	1080,07	494,458	347,495
Etapa 6	322,716	482,445	652,432
Etapa 7	932,906	2311,94	187,225
Etapa 8	185,593	163,612	942,053
Etapa 9	642,725	1400,09	34,3952

Tabla 17 Tiempos por etapa Experimento

El servidor dinámico N°2 es el que demora más tiempo (10372.841), asumiendo esto lo contrastamos con el tiempo del servidor estático (19434.548).

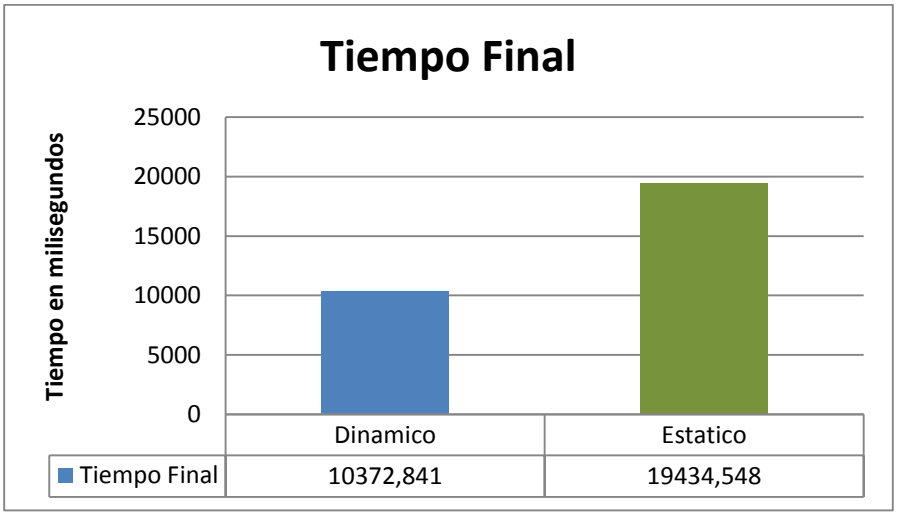


Ilustración 36 Comparación Servidor Estático vs Dinámico

11.9 Conclusiones de Pruebas

Por medio de las simulaciones antes expuestas, se puede decidir qué tipo de descarga presenta un mejor desempeño en escenarios de descarga en paralelo de un archivo alojado en varios servidores. La definición y la apreciación de la variable del tiempo, la cual es un factor muy importante relacionado con el tiempo de espera de los usuarios, y de cómo lograr optimizar esto.

Pero de cierta forma, se abren nuevas interrogantes relacionas con la asignación de paquetes dinámicos. Por ejemplo, si es asignado un paquete a un enlace del servidor que presento el menor tiempo en esa iteración, pero en la próxima etapa experimenta un cambio desfavorable en la velocidad de transferencia, será el servidor con más carga en relación a los otros servidores, los cuales finalizarán su etapa de forma anticipada, pero no se podrá calcular el paquete que se asignará en la próxima etapa hasta que finalice el servidor sobrecargado.

De cierta forma, presenta sus ventajas y desventajas. Puede ser que si poseemos un enlace estable, el cual no presenta variaciones muy grandes, se adapte de forma positiva a nuestros requerimientos, representando de forma óptima lo expuesto en nuestro trabajo.

El objetivo de simular una técnica que mejore este tipo de descargas en una red P2P, se ve cumplido en los escenarios establecidos.

12 Pruebas Aplicación

Al inicio de la simulación realizada en la aplicación Java, se define un paquete a transmitir el cual tendrá un valor definido, solo en la primera iteración. Esto permitirá analizar el tiempo de descarga de cada servidor, para luego realizar una estimación de la tasa de descarga.

Luego, cada ciclo de iteración, dependiendo del tiempo de descarga del paquete anterior, se evaluará la calidad de transmisión y asignará el tamaño del paquete próximo en la nueva iteración. Dependiendo de la estabilidad de la descarga, se logrará aprovechar el servidor que presenta un tiempo menor en relación a los otros servidores en el tiempo de transmisión del paquete.

Para representar las variaciones de tiempo y de tamaño de paquete, se definen a continuación las iteraciones de las pruebas. Estas pruebas, serán realizadas variando la cantidad de ciclos de iteración y el tamaño del archivo a descargar.

12.1 Información de pruebas

A continuación, se mostraran los datos obtenidos de cuatro pruebas, las cuales consisten en descargar 2 archivos de diferente tamaño en diferentes etapas, es decir, 2 pruebas por archivo. El tamaño de estos archivos será uno de tamaño pequeño (6 MB) y otro de gran tamaño (600 MB). Estos archivos se encontraran alojados en servidores diferentes, por ende, la prueba será de un cliente y dos servidores. Estas pruebas se realizaron con una conexión a internet de 2MB.

12.2 Experimentos

12.2.1 Experimento 1

En la primera prueba la haremos descargando el archivo “ccsetup518.exe” el cual está alojado en las siguientes direcciones

http://download.piriform.com/ccsetup518.exe http://www.trehuaco.com/decretos/2016/ccsetup518.exe

En esta prueba haremos tres ciclos de iteración.

Etapa 0:

Se asignan valores por defecto para transmitir en la red. Cada servidor transmitirá un paquete de tamaño 1148,928 kilobytes hacia el Cliente.

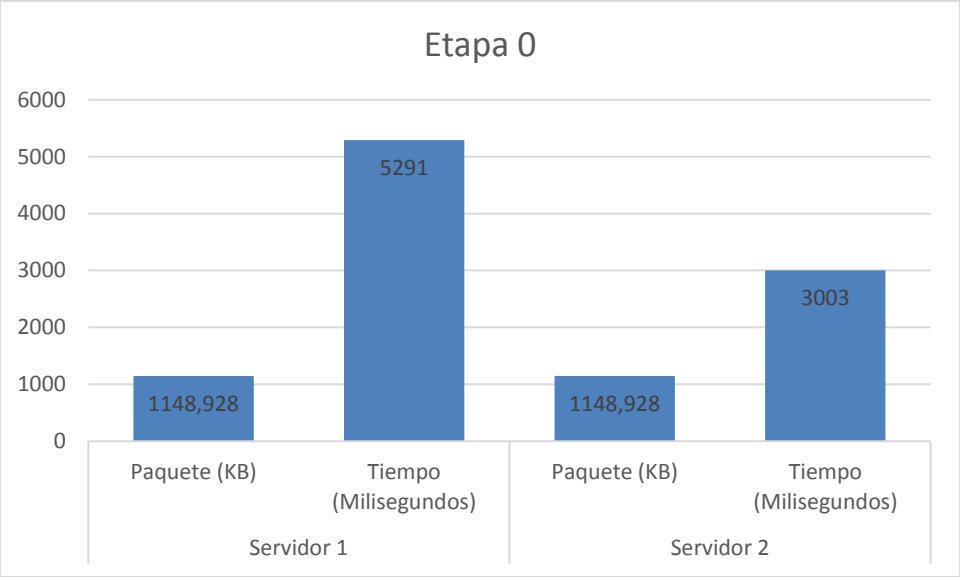


Ilustración 37 Etapa 0 experimento 1

Como se puede apreciar, en esta primera etapa, lo que se pretende es obtener los tiempos, para luego proceder a ejecutar un algoritmo de estimación para definir el tamaño del paquete para cada servidor en el próximo ciclo.

Se utilizan las fórmulas definidas en el Capítulo 6 “Definición del proyecto”.

Etapa 1:

Basándose en los resultados de la primera iteración, se definen los nuevos paquetes y se vuelve a definir el tamaño para la próxima iteración, este proceso se repetirá hasta llegar al límite de etapas definidos en la simulación.

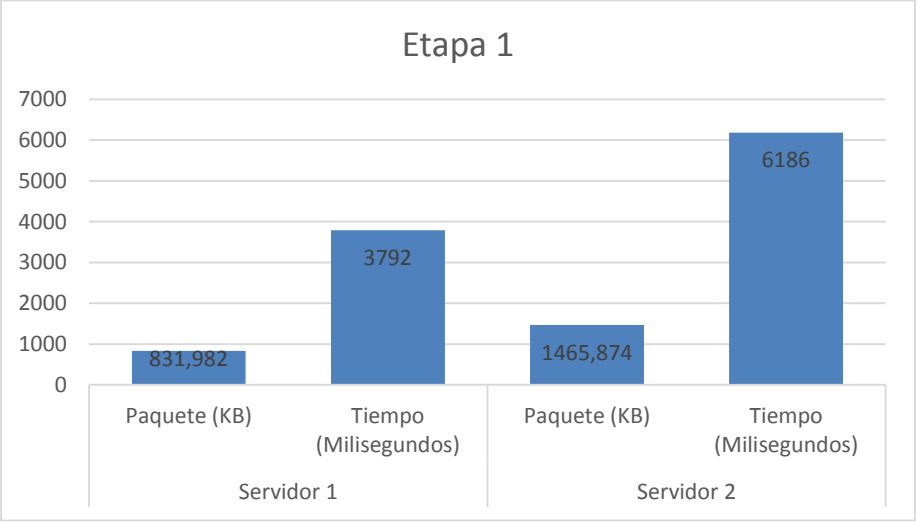


Ilustración 38 Etapa 1 experimento 1

Etapa 2

Como el servidor 1, obtuvo un tiempo de descarga menor en relación al otro servidor en la etapa anterior, el tamaño a descargar en esta iteración es mayor al anterior.

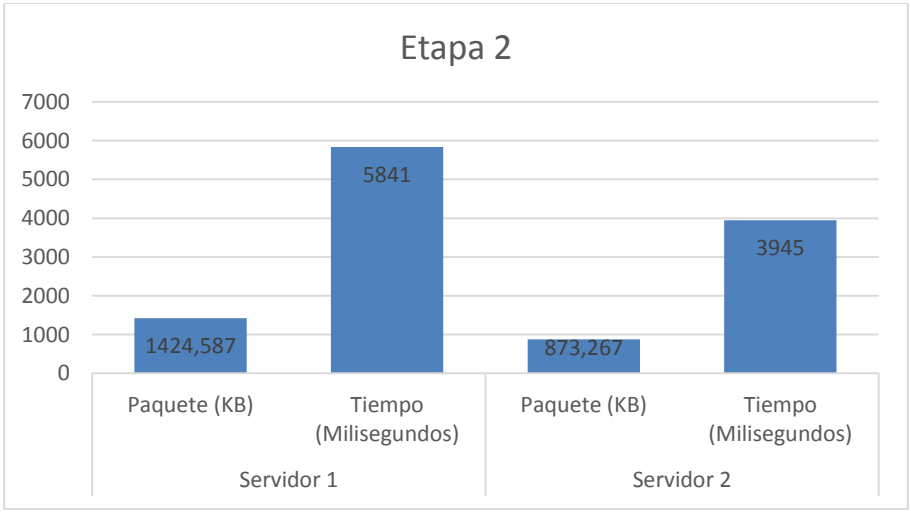


Ilustración 39 Etapa 2 experimento

Tiempo total de descarga: 17318 ms

En cada etapa, se puede apreciar la variación de los tamaños de los paquetes en factor al tiempo obtenido en la iteración anterior a la etapa. En esta etapa, como es la última de este experimento, se ha descargado completamente el archivo.

12.2.2 Experimento 2

En esta prueba haremos la descarga del mismo archivo que el experimento número uno, pero la diferencia será los ciclos de iteración, que en este caso serán cinco.

Etapa 0:

Al igual que en el experimento anterior, en la etapa 0 se hace una repartición de tamaño de paquete por defecto y comienza la descarga.

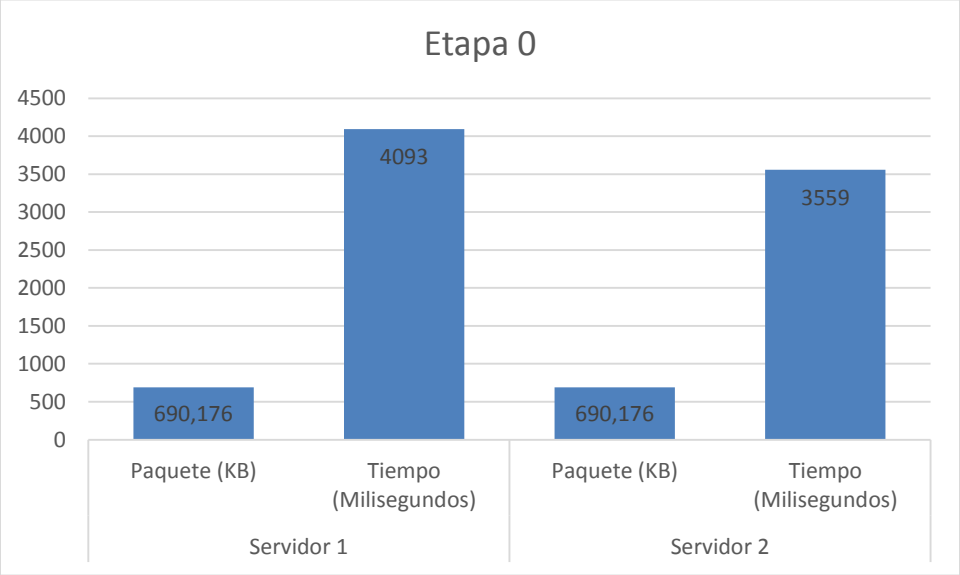


Ilustración 40 Etapa 0 experimento 2

Una vez terminado se hacen los cálculos correspondientes para efectuar la siguiente etapa.

Etapa 1:

Obtenido los tiempos de la etapa anterior se determina el tamaño de paquete para esta etapa. Como ya sabemos, a quien tenga un menor tiempo de descarga se le asignara un mayor tamaño de paquete a descargar y así veremos las distintas graficas hasta completar el número de etapas y descargar por completo el archivo.

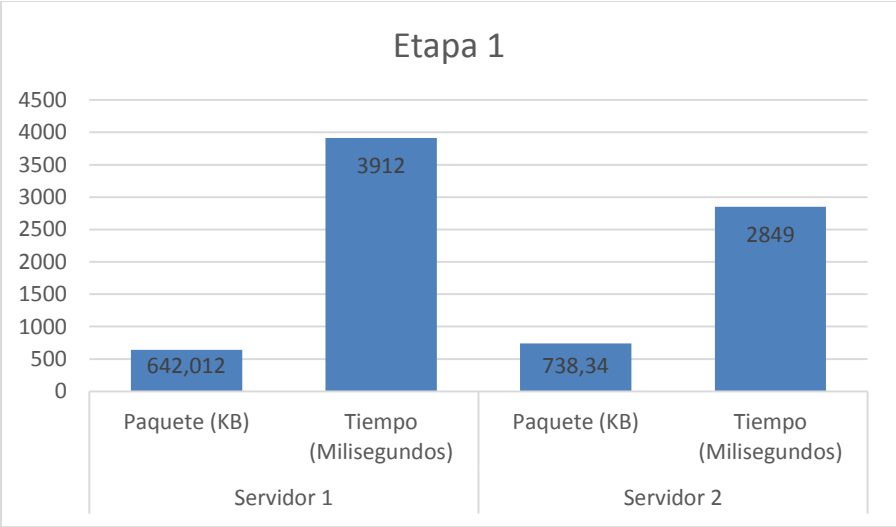


Ilustración 41 Etapa 1 experimento 2

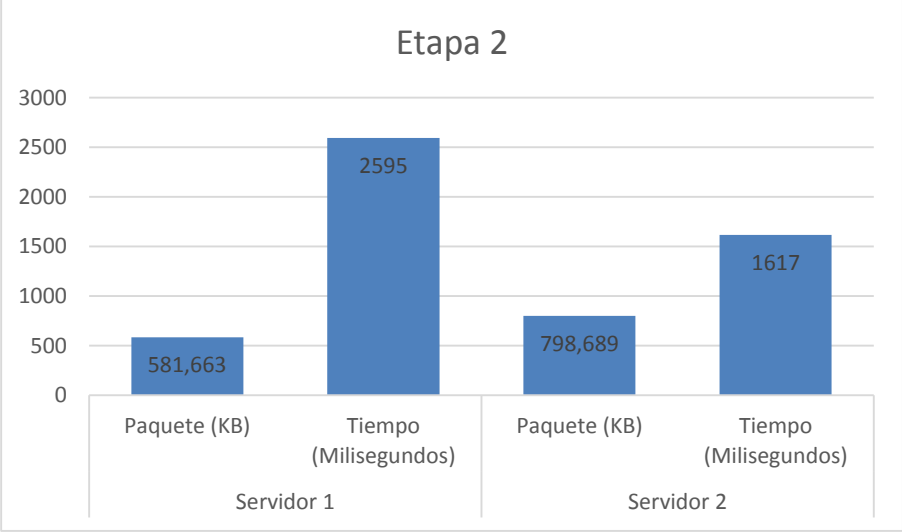


Ilustración 43: Etapa 2 experimento 2

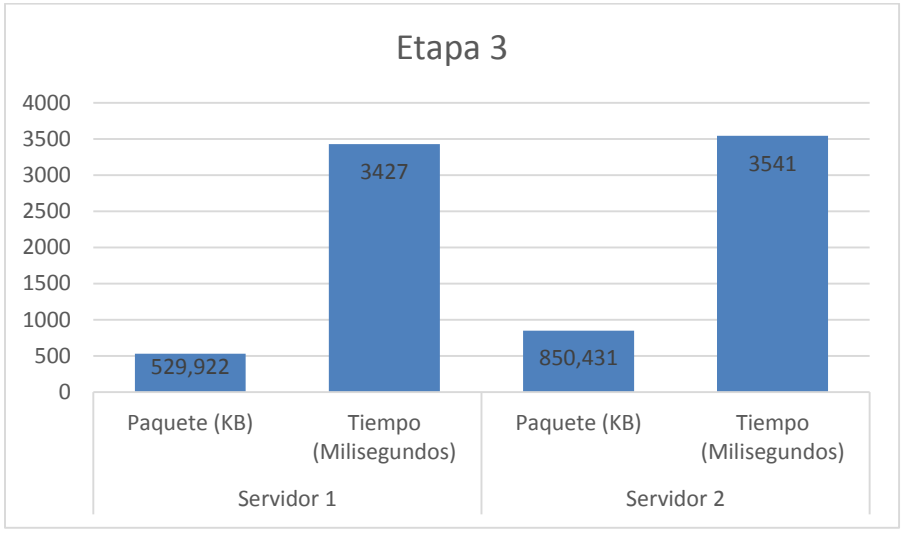


Ilustración 42 Etapa 3 experimento 2

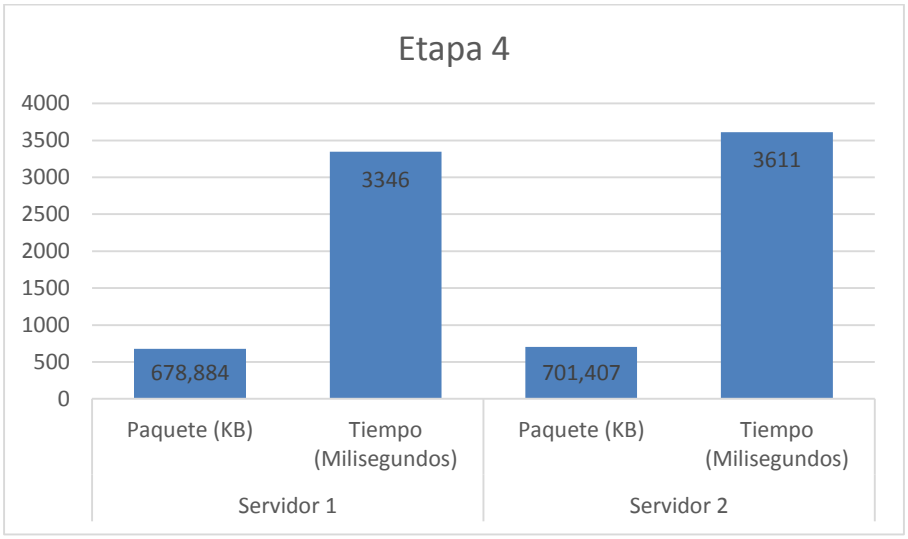


Ilustración 45: Etapa 4 experimento 2

Tiempo total de descarga: 17752 ms

12.2.3 Experimento 3

En la primera prueba la haremos descargando el archivo “debian-8.4.0-amd64-CD-1.iso” el cual está alojado en las siguientes direcciones:

http://debian.uni-duisburg-essen.de/debian-cd/8.4.0/amd64/iso-cd/debian-8.4.0-amd64-CD-1.iso
http://debian.inode.at/debian-cd/8.4.0/amd64/iso-cd/debian-8.4.0-amd64-CD-1.iso

En este experimento haremos cinco ciclos de iteración. Todo este proceso es realizado de la misma forma que los experimentos anteriores

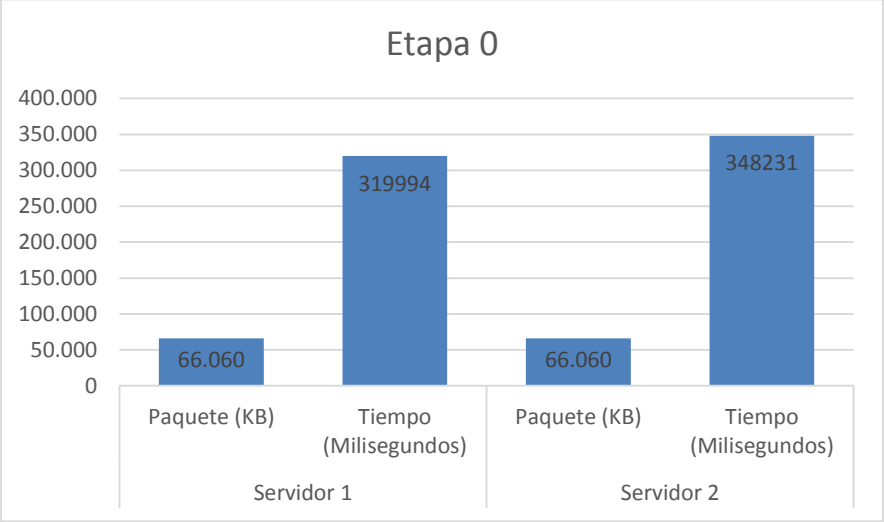


Ilustración 43 Etapa 0 experimento 3

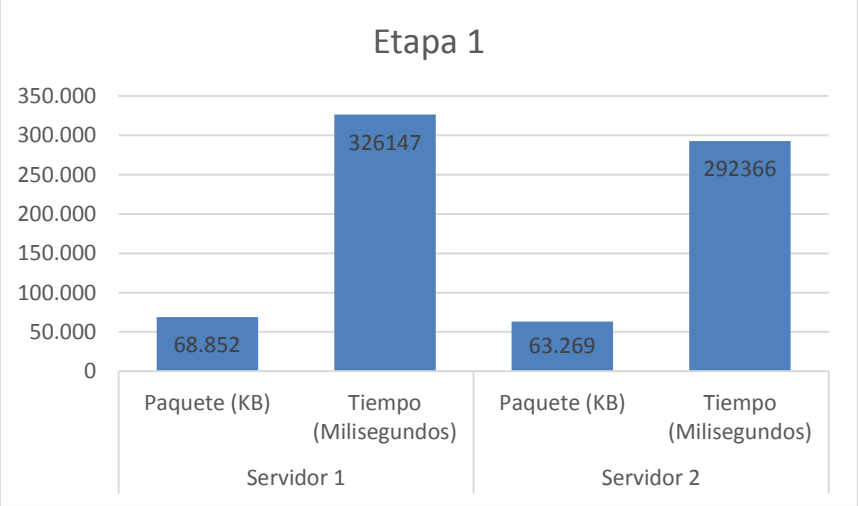


Ilustración 44 Etapa 1 experimento 3

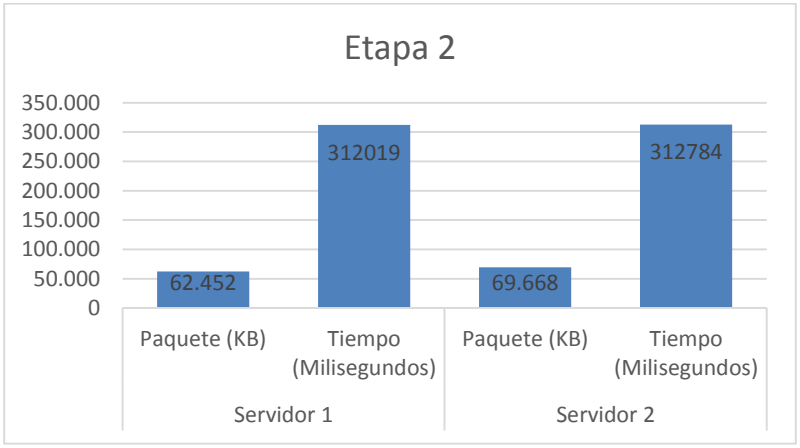


Ilustración 45 Etapa 2 experimento 3

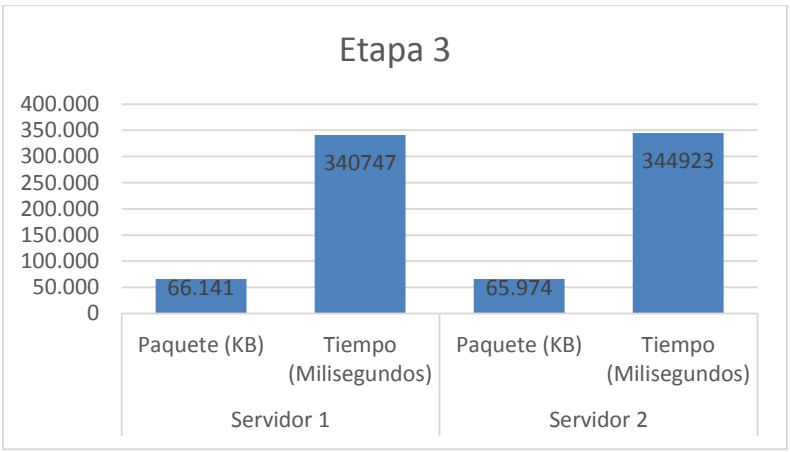


Ilustración 46 Etapa 3 experimento 3

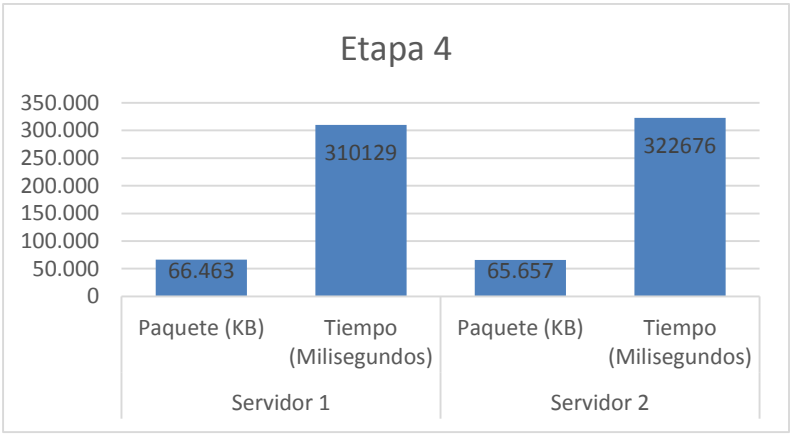


Ilustración 47 Etapa 4 experimento 3

Tiempo total de descarga: 1654761 ms

12.2.4 Experimento 4

En el siguiente experimento, descargaremos el mismo archivo que en el experimento 3, pero esta vez veremos el comportamiento cuando es descargado en diez ciclos de iteración.

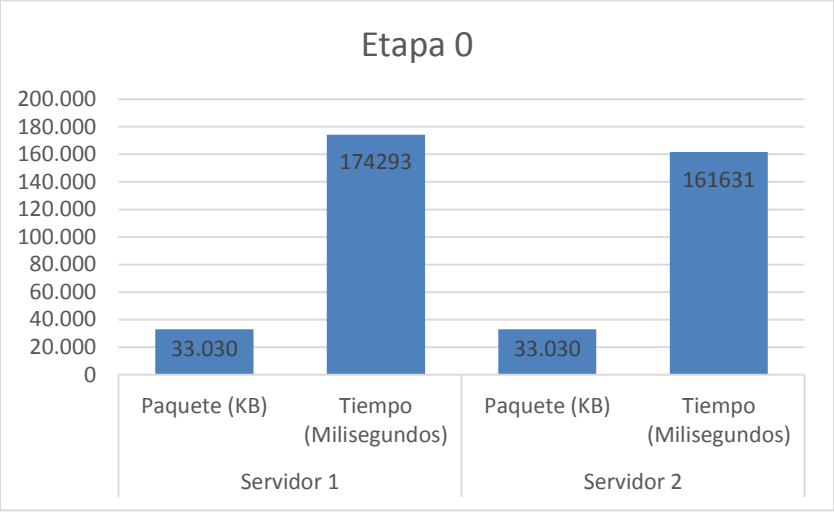


Ilustración 48 Etapa 0 experimento 4

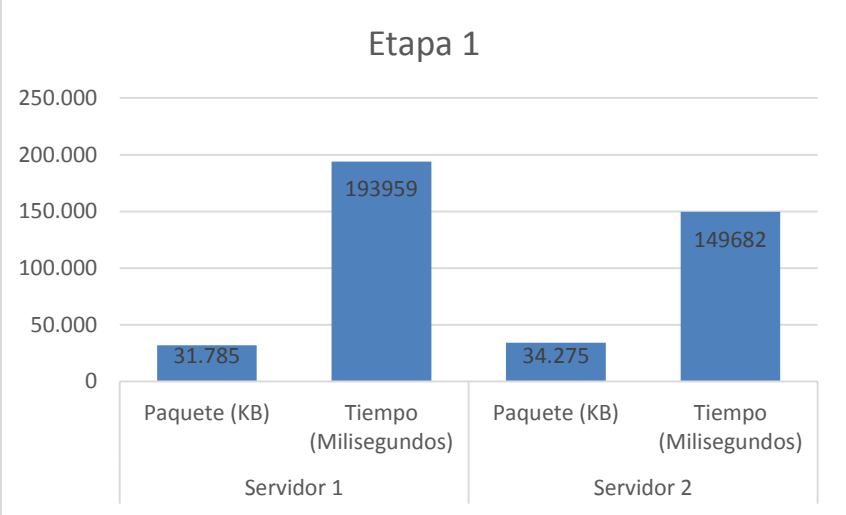


Ilustración 52: Etapa 1 experimento 4

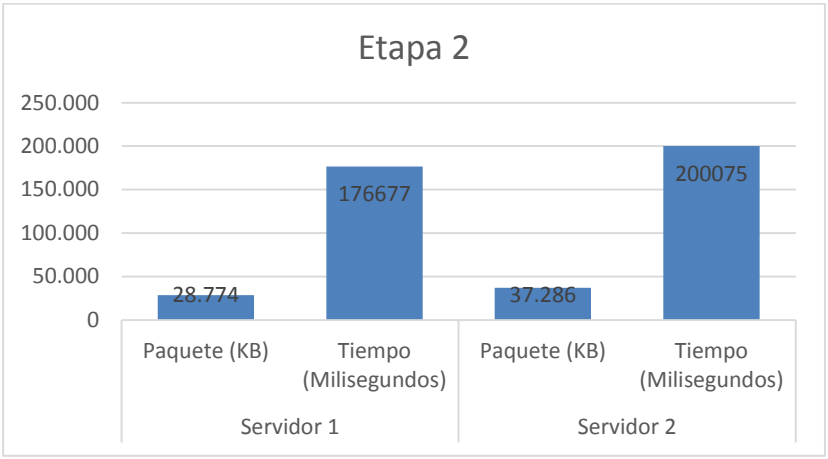


Ilustración 49 Etapa 2 experimento 4

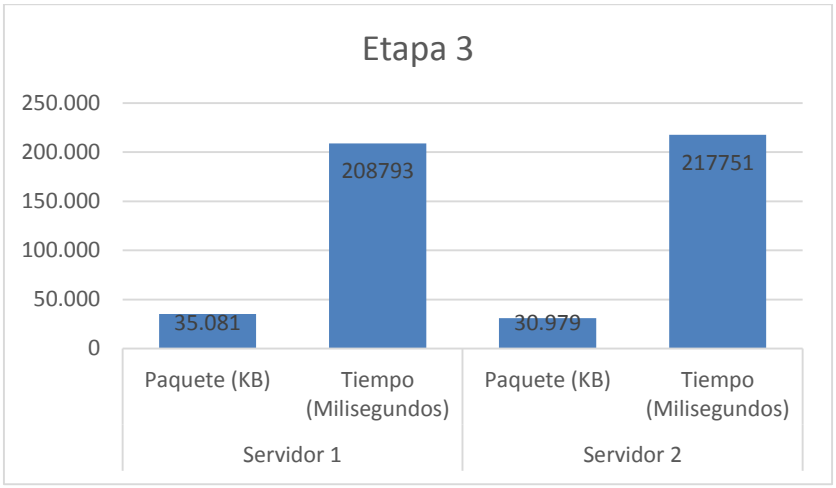


Ilustración 50 Etapa 3 experimento 4

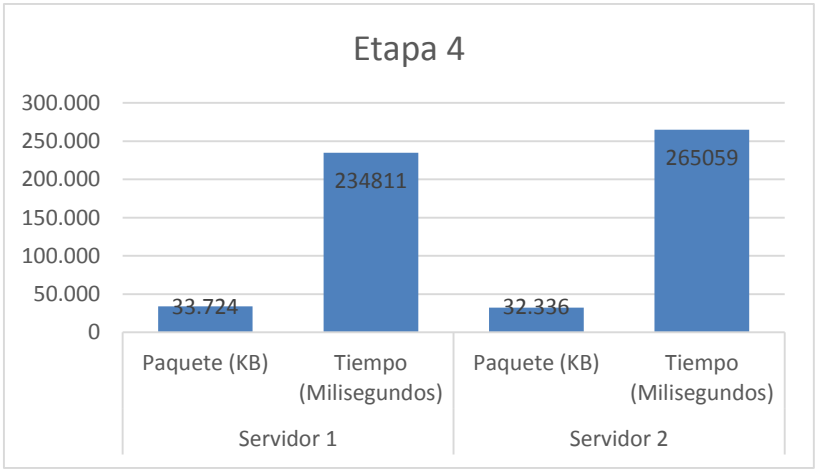


Ilustración 51 Etapa 4 experimento 4

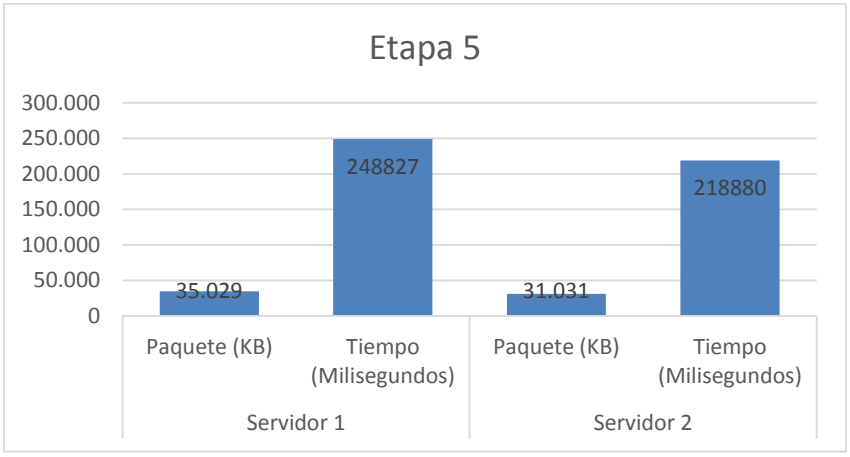


Ilustración 52 Etapa 5 experimento 4

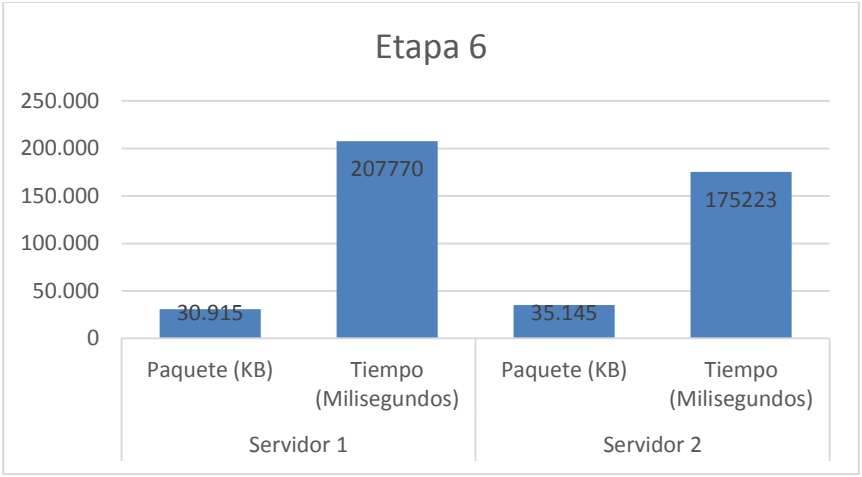


Ilustración 53 Etapa 6 experimento 4

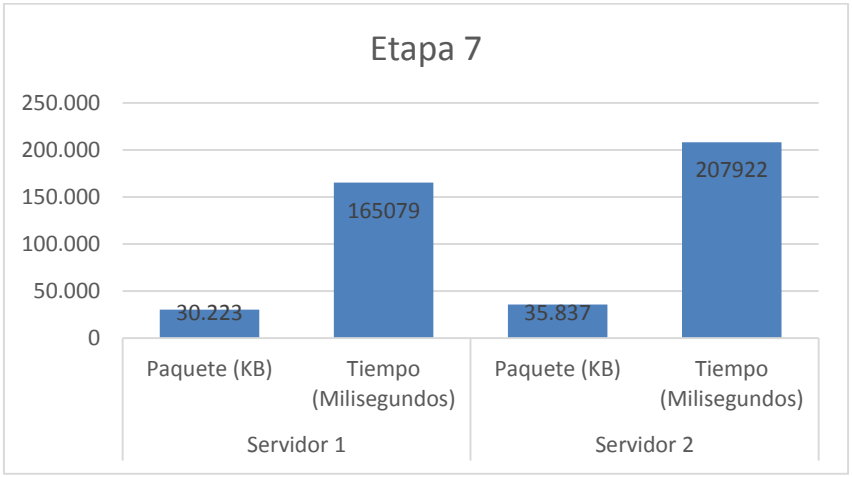


Ilustración 54 Etapa 7 experimento 4

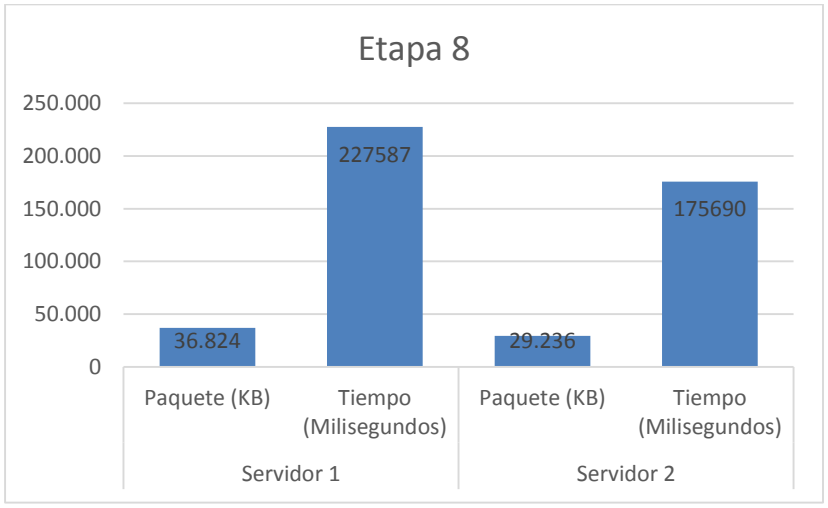


Ilustración 55 Etapa 8 experimento 4

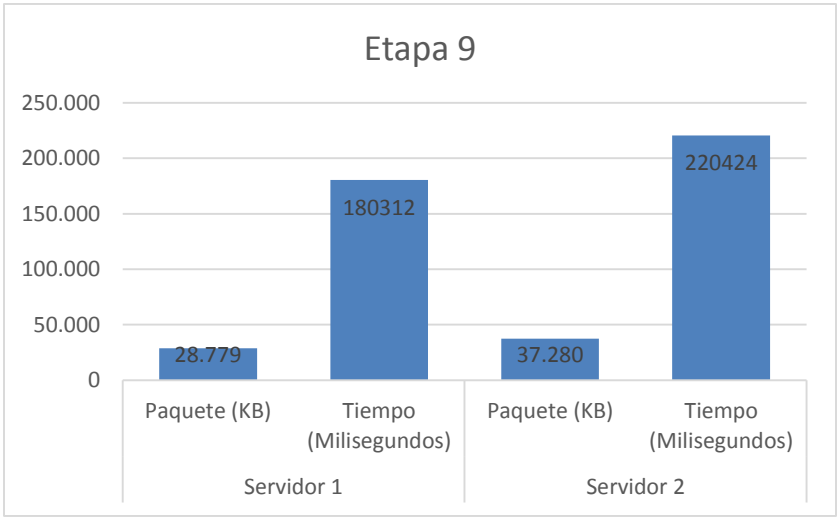


Ilustración 56 Etapa 9 experimento 4

Tiempo total de descarga: 2163667 ms

12.3 Conclusiones pruebas

Por medio de las simulaciones antes expuestas, se puede decidir qué tipo de descarga presenta un mejor desempeño en escenarios de descarga en paralelo de un archivo alojado en varios servidores. La definición y la apreciación de la variable del tiempo, la cual es un factor muy importante relacionado con el tiempo de espera de los usuarios, y de cómo lograr optimizar esto.

Pero de cierta forma, se abren nuevas interrogantes relacionas con la asignación de paquetes dinámicos. Por ejemplo, si es asignado un paquete a un enlace del servidor que presento el menor tiempo en esa iteración, pero en la próxima etapa experimenta un cambio desfavorable en la velocidad de transferencia, será el servidor con más carga en relación a los otros servidores, los cuales finalizarán su etapa de forma anticipada, pero no se podrá calcular el paquete que se asignará en la próxima etapa hasta que finalice el servidor sobrecargado.

De cierta forma, presenta sus ventajas y desventajas. Puede ser que si poseemos un enlace estable, el cual no presenta variaciones muy grandes, se adapte de forma positiva a nuestros requerimientos, representando de forma óptima lo expuesto en nuestro trabajo.

13 Bibliografía

- *[1] RFC 21616 "Protocolo de Transferencia de Hipertexto (HTTP/1.1)"*
- *[2] Rodriguez P. y Biersack W. "Dynamic Parallel Access to Replicated Content in the Internet". IEEE/ACM Transaction on Networking, 10(4), pp 455-465, Agosto 2002*
- *[3] Funasaka J., Nakawaki N., Ishida K. y Amano K. "A Parallel Downloading Method of Coping with Variable Bandwidth". ICDCS Workshops 2003, pp 14-19, Mayo 2003*
- *[4] J. Byers, M. Luby, and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads", In INFOCOM 99, April 1999.*
- *[5] S. Sohail, S.-K. Jha, S. Kanhere, C-T. Chou, "QoS Driven Parallelization of Resources to Reduce File Download Delay," IEEE Transactions on Parallel and Distributed Systems, vol. 17, no. 10, pp. 1204-1215, October, 2006.*
- *[6] D.Rafael Cano. "Entorno de simulación de redes TCP/IP usando servicios REST basados en la nube computacional" 11 Diciembre de 2012*
- *[7] Martín Zubeldía "Eficiencia y justicia en redes NS3" , 2012*
- *[8] Antonio Conejero "Redes de sensores inalámbricos y su simulación en el Network Simulator Version3" Junio 2014*
- *[9] Network Simulator 3. NS-3 http://www.nsnam.org/*
- *[10] Instalación del NS-3. http://www.nsnam.org/wiki/index.php/Installation*
- *[11] Cristopher Barrientos Matamala. "Descarga en paralelo de archivos sensible a variaciones del ancho de banda" 2014*
- *[13] P2P = Steinmetz, R.; Wehrle, K (2005). 2. What Is This "Peer-to-Peer" About? Springer Berlin Heidelberg. pp. 9–16.*
- *[14] UDP = https://es.wikipedia.org/wiki/User_Datagram_Protocol*
- *[15] TCP = Cerf, V.; Kahn, R. (1974). «A Protocol for Packet Network Intercommunication». IEEE Transactions on Communications. COM-22 (5): 637–648.*
- *[16] Helpers : https://www.nsnam.org/docs/manual/html/helpers.html*
- *[17] NS2 : http://www.isi.edu/nsnam/ns/*

- *[18] NS3 : https://www.nsnam.org/*
- *[19] Omnet++: https://omnetpp.org/*
- *[20] SimPy: https://simpy.readthedocs.org/en/latest/*
- *[21] QualNet: http://web.scalable-networks.com/content/qualnet*
- *[22] NIST. (10-7-2009, NIST Definition of Cloud Computing v15. Available: http://csrc.nist.gov/ groups/SNS/cloud-computing/*
- *[23] Computación en la nube: I. Cisco Syst., San Jose, CA. (2009, Cisco Cloud Computing - Data Center Strat-egy, Architecture, and Solutions. Avail-able: http://www.cisco.com/web/strategy/docs/gov/CiscoCloudCom-puting_ WP.pdf*
- *[24] Computación en la nube: Jaider Ospina, "An Approach to the state of art in Cloud Computing" abril 2013*
- *[25] Patricio Galdames Sepúlveda, Claudio Gutiérrez Soto, Cristopher Barrientos Matamala. "A cost-efficient subscription overlay network for large-scale cloud collaboration". 2015*

14 Conclusiones

Los objetivos planteados al inicio del proyecto se han ido cumpliendo de manera progresiva mediante el avance de la planificación establecida. El tipo de descarga en estudio representa una mejora en la manera de descargar archivos desde la web, acortando tiempos y mejorando el rendimiento del sistema. Razón por la cual resulta interesante el desarrollo de este tipo de descarga.

El estudio de la herramienta NS-3 ha resultado un poco difícil en un principio, aunque utiliza un lenguaje de programación relativamente conocido (C++), comprender el funcionamiento de sus clases y dispositivos internos, las diferentes formas de realizar las simulaciones y el poco material relacionado con nuestra investigación, nos han generado un retraso en el tiempo de la planificación planteada desde un principio.

Relacionado a la investigación de los protocolos utilizados, el proyecto nos permitió adquirir más conocimientos en el área de comunicación de datos y redes. La investigación del protocolo HTTP nos permitió conocer sus funcionalidades y facilidad de uso que presenta entre sus características, y lo fácil que puede ser realizar peticiones a servidores.

La implementación en lenguaje en Java de la aplicación, nos permitió ver todas las funcionalidades que se pueden desarrollar con este poderoso lenguaje de programación. La gran cantidad de material disponible de Java y la simplicidad del código, permite una buena comprensión de lo que se está desarrollando. Además, uno de los puntos fuertes del trabajo era el desarrollo de conexiones multi-hilos, característica que Java permite implementar sin problemas.

La investigación en trabajos similares nos entregó conocimiento y técnicas para poder realizar el estudio y la implementación de fórmulas para calcular los tiempos de descarga entre servidores y etapas descritas en el informe.

En relación al área académica, el desarrollo de este proyecto nos entregó capacidades y conocimientos con los cuales no contábamos, como utilizar el simulador de eventos discretos NS-3 y desarrollar varios tipos de simulaciones, conocimiento de los diferentes protocolos, las ventajas y desventajas de estos, el funcionamiento de las descargas en paralelo en la web y desarrollar e implementar un proyecto en Java.

Como equipo de trabajo, el informe nos entregó capacidades para trabajar en paralelo, herramientas y conocimientos personales en áreas que no presentábamos grandes conocimientos, enfrentar frustraciones y afrontar desafíos.

15 Anexos Códigos Simulación

Se adjunta Script de Simulación básica en NS3, a modo de ejemplo:

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/netanim-module.h"
#include "ns3/ipv4-nix-vector-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/event-id.h"

using namespace ns3;

long         tamano_total       = 1022976;
uint16_t   n_etapas             = 10;
long         tamano_etapa       = 102297;
uint16_t   n_servidores       = 3;
long         total_descargado = 0;
uint16_t sinkPort = 5000;
uint32_t count = 0;
long long suma_paquetes=0;
long long suma=0;
double valor;
uint32_t id_servidor;
std::vector<Ipv4Address> v_addrUsuarios;
std::vector<Ipv4Address> v_addrServidores;
std::map< Ptr<Socket>, Ipv4Address > m_socketAddressesRecv;
std::map< Ipv4Address, Ptr<Socket> > m_addressesSocketSend;
std::vector<float> v_bn;
std::vector<float> tiempo_servidor0;
std::vector<float> tiempo_servidor1;
std::vector<float> tiempo_servidor2;
std::vector<float> v_bn_basico;
```

```
std::vector<float> v_for_one;

struct Time_Servidor{
   uint32_t servidor_id;
   float tiempo_paquete;
   float tamano_paquete;
};

struct Array_Info{
   uint32_t id_etapa;
   uint32_t servidor;
   float time_paquete;
   float package_send;
};

std::vector<Time_Servidor> v_flujo;
std::vector<Array_Info> v_flujo_info;

void
SendPacket (Ptr<Socket> socket, Ptr<Packet> p)
{
   socket->Send (p);
}

void
ReceivePacket (Ptr<Socket> socket)
{
   valor=Simulator::Now ().GetSeconds ();

   Address sourceAddress;
   Ptr<Packet> packet = socket->RecvFrom (sourceAddress);
   InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);
   Ipv4Address sender = inetSourceAddr.GetIpv4 ();
   Ipv4Address receiver = m_socketAddressesRecv[socket];
```

```
  if(sender == "10.1.1.1")
  {
      id_servidor = 0;
  }else if(sender == "10.1.2.1"){
      id_servidor= 1;
  }else if(sender == "10.1.3.1"){
      id_servidor = 2;
  }

  suma_paquetes=suma_paquetes+packet->GetSize ();
  suma=suma+packet->GetSize();

  Time_Servidor flujotcp;
       flujotcp.servidor_id = id_servidor;
       flujotcp.tiempo_paquete = valor;
       flujotcp.tamano_paquete = packet->GetSize ();
       v_flujo.push_back(flujotcp);


  std::map< Ipv4Address, Ptr<Socket> >::iterator it;
  it = m_addressesSocketSend.find(receiver);
}
void
time_package(uint32_t num_etapa)
{
  suma=0;
  // Mostrar todos los flujos de la red
  for(uint32_t i = 0; i < v_flujo.size(); i++)
   {
      Array_Info flujo_info;
    flujo_info.id_etapa = num_etapa;
    flujo_info.servidor = v_flujo.at(i).servidor_id;
    flujo_info.time_paquete = v_flujo.at(i).tiempo_paquete;
    flujo_info.package_send = v_flujo.at(i).tamano_paquete;
    v_flujo_info.push_back(flujo_info);
   }
```

```
while (!v_flujo.empty())
    {
        v_flujo.back();
        v_flujo.pop_back();
    }

    while (!tiempo_servidor0.empty())
    {
        tiempo_servidor0.back();
        tiempo_servidor0.pop_back();
    }
    while (!tiempo_servidor1.empty())
    {
        tiempo_servidor1.back();
        tiempo_servidor1.pop_back();
    }
    while (!tiempo_servidor2.empty())
    {
        tiempo_servidor2.back();
        tiempo_servidor2.pop_back();
    }

    while (!v_bn_basico.empty())
    {
        v_bn_basico.back();
        v_bn_basico.pop_back();
    }

    while (!v_bn.empty())
    {
        v_bn.back();
        v_bn.pop_back();
    }

    while (!v_for_one.empty())
    {
```

```
            v_for_one.back();
            v_for_one.pop_back();
        }

}

void
calcula_paquete( float x1, float x2, float x3)
{
        v_bn_basico.push_back(x1*1000);
        v_bn_basico.push_back(x2*1000);
        v_bn_basico.push_back(x3*1000);
        float sumatoria_factor = 0;

          for(uint32_t i=0;i<v_bn_basico.size();i++)
          {
            std::cout << "Tiempo Servidor "<< i << " = " << v_bn_basico[i] <<" Milisegundos"<< std::endl;
            float factor = tamano_etapa/v_bn_basico[i];
            sumatoria_factor = sumatoria_factor+factor;
            v_for_one.push_back(factor);
          }

          //CONSTANTE RESULTANTE DE LA FORMULA 3
          float constante = tamano_etapa/sumatoria_factor;

          for(uint32_t j=0;j<v_for_one.size();j++)
          {
            v_bn.push_back(v_for_one[j] * constante);
            std::cout << "Tamaño Paquete Servidor "<< j << " = " << v_bn[j] <<" Mb "<< std::endl;
          }

}

void
calcula_tiempos(uint32_t iteracion)
{
```

```
  for(uint32_t i = 0; i < v_flujo_info.size(); i++)
   {

    if((v_flujo_info.at(i).id_etapa == iteracion) && (v_flujo_info.at(i).servidor == 0))
    {
         tiempo_servidor0.push_back(v_flujo_info.at(i).time_paquete);
    }
    if((v_flujo_info.at(i).id_etapa == iteracion) && (v_flujo_info.at(i).servidor == 1))
    {
         tiempo_servidor1.push_back(v_flujo_info.at(i).time_paquete);
    }
    if((v_flujo_info.at(i).id_etapa == iteracion) && (v_flujo_info.at(i).servidor == 2))
    {
         tiempo_servidor2.push_back(v_flujo_info.at(i).time_paquete);
    }

   }

   std::cout << "Numero de etapa : " << iteracion << std::endl;


   uint32_t final_serv0 = tiempo_servidor0.size();
   float ser0 = tiempo_servidor0[final_serv0-1]-tiempo_servidor0[0];

   uint32_t final_serv1 = tiempo_servidor1.size();
   float ser1 = tiempo_servidor1[final_serv1-1]-tiempo_servidor1[0];

   uint32_t final_serv2 = tiempo_servidor2.size();
   float ser2 = tiempo_servidor2[final_serv2-1]-tiempo_servidor2[0];

   calcula_paquete(ser0, ser1, ser2);

   std::cout << "*****************************************************" << std::endl;

}
```

```
void
HandleAccept (Ptr<Socket> s, const Address& from)
{

  s->ShutdownSend ();
  s->SetRecvCallback (MakeCallback (&ReceivePacket));
  m_socketAddressesRecv.insert (std::make_pair (s, v_addrUsuarios.at (0)));
  //count++;
}

int main(int argc, char *argv[])
{
  std::string animFile = "paquetes_dinamicos.xml";
  std::vector<NetDeviceContainer> vNdc;

  NodeContainer usuarios;
  NodeContainer servidores;
  NodeContainer router;
  NodeContainer usuarios_rev;
  NodeContainer servidores_env;

  usuarios.Create(3);
  router.Create(1);
  servidores.Create(3);

  NodeContainer s1r1 = NodeContainer(servidores.Get(0), router.Get(0));
  NodeContainer s2r1 = NodeContainer(servidores.Get(1), router.Get(0));
  NodeContainer s3r1 = NodeContainer(servidores.Get(2), router.Get(0));
  NodeContainer u1r1 = NodeContainer(usuarios.Get(0), router.Get(0));
  NodeContainer u2r1 = NodeContainer(usuarios.Get(1), router.Get(0));
  NodeContainer u3r1 = NodeContainer(usuarios.Get(2), router.Get(0));


  servidores_env.Add(s1r1.Get(0));
  servidores_env.Add(s2r1.Get(0));
  servidores_env.Add(s3r1.Get(0));
```

```
usuarios_rev.Add(u1r1.Get(0));
usuarios_rev.Add(u2r1.Get(0));
usuarios_rev.Add(u3r1.Get(0));

MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

  PointToPointHelper pointToPoint;

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("15Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("70ms"));
  NetDeviceContainer ds1r1 = pointToPoint.Install(s1r1);
  vNdc.push_back(ds1r1);
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("15Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("80ms"));
  NetDeviceContainer ds2r1 = pointToPoint.Install(s2r1);
  vNdc.push_back(ds2r1);
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("15Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("70ms"));
  NetDeviceContainer ds3r1 = pointToPoint.Install(s3r1);
  vNdc.push_back(ds3r1);
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("6ms"));
  NetDeviceContainer du1r1 = pointToPoint.Install(u1r1);
  vNdc.push_back(du1r1);
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("6ms"));
  NetDeviceContainer du2r1 = pointToPoint.Install(u2r1);
  vNdc.push_back(du2r1);
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("6ms"));
  NetDeviceContainer du3r1 = pointToPoint.Install(u3r1);
  vNdc.push_back(du3r1);
```

```
    Ipv4NixVectorHelper nixRouting;
    Ipv4StaticRoutingHelper staticRouting;

    Ipv4ListRoutingHelper list;
    list.Add (staticRouting, 0);
    list.Add (nixRouting, 10);

    InternetStackHelper ipStack;
    ipStack.SetRoutingHelper (list);
    ipStack.Install (usuarios);
    ipStack.Install (servidores);
    ipStack.Install (router);
    mobility.Install (usuarios);
    mobility.Install (servidores);
    mobility.Install (router);


    // Posiciones de los nodos de la red 1
  Ptr<ConstantPositionMobilityModel> mms1 = servidores.Get (0)->GetObject<ConstantPositionMobilityModel>
(); // Posición servidor 1
  mms1->SetPosition (Vector (2, 5, 0));

  Ptr<ConstantPositionMobilityModel> mms2 = servidores.Get (1)->GetObject<ConstantPositionMobilityModel>
(); // Posición servidor 2
  mms2->SetPosition (Vector (4, 5, 0));

  Ptr<ConstantPositionMobilityModel> mms3 = servidores.Get (2)->GetObject<ConstantPositionMobilityModel>
(); // Posición servidor 3
  mms3->SetPosition (Vector (6, 5, 0));

  Ptr<ConstantPositionMobilityModel> mmr1 = router.Get (0)->GetObject<ConstantPositionMobilityModel> (); //
Posición router 1
  mmr1->SetPosition (Vector (4, 8, 0));

  Ptr<ConstantPositionMobilityModel> mmu1 = usuarios.Get (0)->GetObject<ConstantPositionMobilityModel> ();
// Posición usuarios 1
  mmu1->SetPosition (Vector (2, 12, 0));
```

```
   Ptr<ConstantPositionMobilityModel> mmu2 = usuarios.Get (1)->GetObject<ConstantPositionMobilityModel> ();
// Posición usuarios 2
   mmu2->SetPosition (Vector (4, 12, 0));


   Ptr<ConstantPositionMobilityModel> mmu3 = usuarios.Get (2)->GetObject<ConstantPositionMobilityModel> ();
// Posición usuarios 3
   mmu3->SetPosition (Vector (6, 12, 0));


   Ipv4AddressHelper ipv4;
     std::vector<Ipv4InterfaceContainer> interfaceNodes;


     for(uint32_t i = 0; i < vNdc.size (); ++i)
     {
        std::ostringstream subnet;
        subnet << "10.1." << i+1 << ".0";
        ipv4.SetBase (subnet.str ().c_str (), "255.255.255.0");
        interfaceNodes.push_back (ipv4.Assign (vNdc[i]));
     }
    v_addrServidores.push_back("10.1.1.1");
     v_addrServidores.push_back("10.1.2.1");
     v_addrServidores.push_back("10.1.3.2");

     v_addrUsuarios.push_back("10.1.4.1");
     v_addrUsuarios.push_back("10.1.5.1");
     v_addrUsuarios.push_back("10.1.6.1");

     for(uint32_t i = 0; i < 3; i++)
     {

        for(uint32_t j= 0; j< 3 ; j++)
        {

           Ptr<Socket> sourceSocket = Socket::CreateSocket (servidores_env.Get (i),
TcpSocketFactory::GetTypeId ()); //ringUsers es el nodo y TcpSocketFactory el tipo
           InetSocketAddress remote = InetSocketAddress (v_addrUsuarios.at (j), sinkPort); //v_addrRemote es el
argumento ipv4(aca vendria siendo donde va dirigido) y skinport el puerto
           std::cout << "id nodes envia: " << servidores_env.Get (i)->GetId ()  << " //  id nodes recibe: " <<
usuarios.Get (j)->GetId () << std::endl;
```

```
            sourceSocket->Bind ();
            sourceSocket->Connect (remote);
            sourceSocket->ShutdownRecv ();
            m_addressesSocketSend.insert (std::make_pair (v_addrServidores.at (i), sourceSocket));


            Ptr<Socket> recvSocket = Socket::CreateSocket (usuarios_rev.Get (j), TcpSocketFactory::GetTypeId
());

            InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), sinkPort);    //Aca se pone el
local y el puerto

            recvSocket->Bind (local);
            recvSocket->Listen ();
            recvSocket->ShutdownSend ();
            //std::cout << "v_addrUsers.at (i): " << v_addrUsers.at (i) << std::endl;
            std::cout << "local: " << local.GetIpv4 () << std::endl;

            recvSocket->SetAcceptCallback (MakeNullCallback<bool, Ptr<Socket>, const Address &> (),
                                           MakeCallback (&HandleAccept));

        }

      }

   AnimationInterface anim (animFile);
for(uint32_t i=0;i<n_servidores;i++)
{
   v_bn.push_back(34099);
}

   for(uint32_t i = 0; i<n_etapas ;i++)
   {

        for(uint32_t j=0;j<n_servidores;j++)
        {
                uint32_t bn =v_bn[j];
```

```
            Ptr<Packet> packet = Create<Packet> (bn);
            Ptr<Socket> srcSocket = m_addressesSocketSend[v_addrServidores.at (j)];
            Simulator::Schedule (Seconds(0.0),   &SendPacket, srcSocket, packet);
        }

         Simulator::Stop ();
         Simulator::Run ();
         std::cout << "TAMAÑO TRANSMITIDO   EN ETAPA "<< i << " = " <<suma << std::endl;
         time_package(i);
         calcula_tiempos(i);
         std::cout << "TAMAÑO TRANSMITIDO ACUMULADO EN ETAPA "<< i << " = " <<suma_paquetes <<
std::endl;

    }

    Simulator::Destroy ();
    return 0;

}
```

16 Anexos NS3

16.1 Instalación NS3

16.1.1 Requisitos previos

Primero que todo, para poder instalar NS-3 simulador debemos cumplir ciertos prerrequisitos de sistema e instalarlos. Para instalar lo que el sistema necesita, daremos la lista de comandos que necesitaremos para completar la operación a continuación:

Instalación de C++:

- $ sudo apt-getinstallgcc g++ Python

Instalación de Python:

- $ sudo apt-getinstallgcc g++ pythonpython-dev

Instalación de Mercurial:

- $ sudo apt-getinstall mercurial

Instalación de Bazaar:

- - $ sudo apt-getinstallbzr

Instalación de CSV:

- apt-getinstallcvs

Instalación de GIT:

- sudo apt-getinstallgit

Instalación de UnzipTool:

- sudo apt-getinstallunzip

Instalación de UnrarTool:

- sudo apt-getinstallunrar-free

Instalación de 7z data compressionutility:

- sudo apt-getinstall p7zip-full

Instalación de XZ data compressionutility:

- $ sudo apt-getinstallxz-utils

Instalación de make:

- sudo apt-getinstallbuild-essential

Instalación de make:

- sudo apt-getinstallcmake

Instalación de patchtool:

- sudo apt-getinstallpatch

Instalación de autoreconftool:

- sudo apt-getinstalldh-autoreconf

16.1.2 Instalación

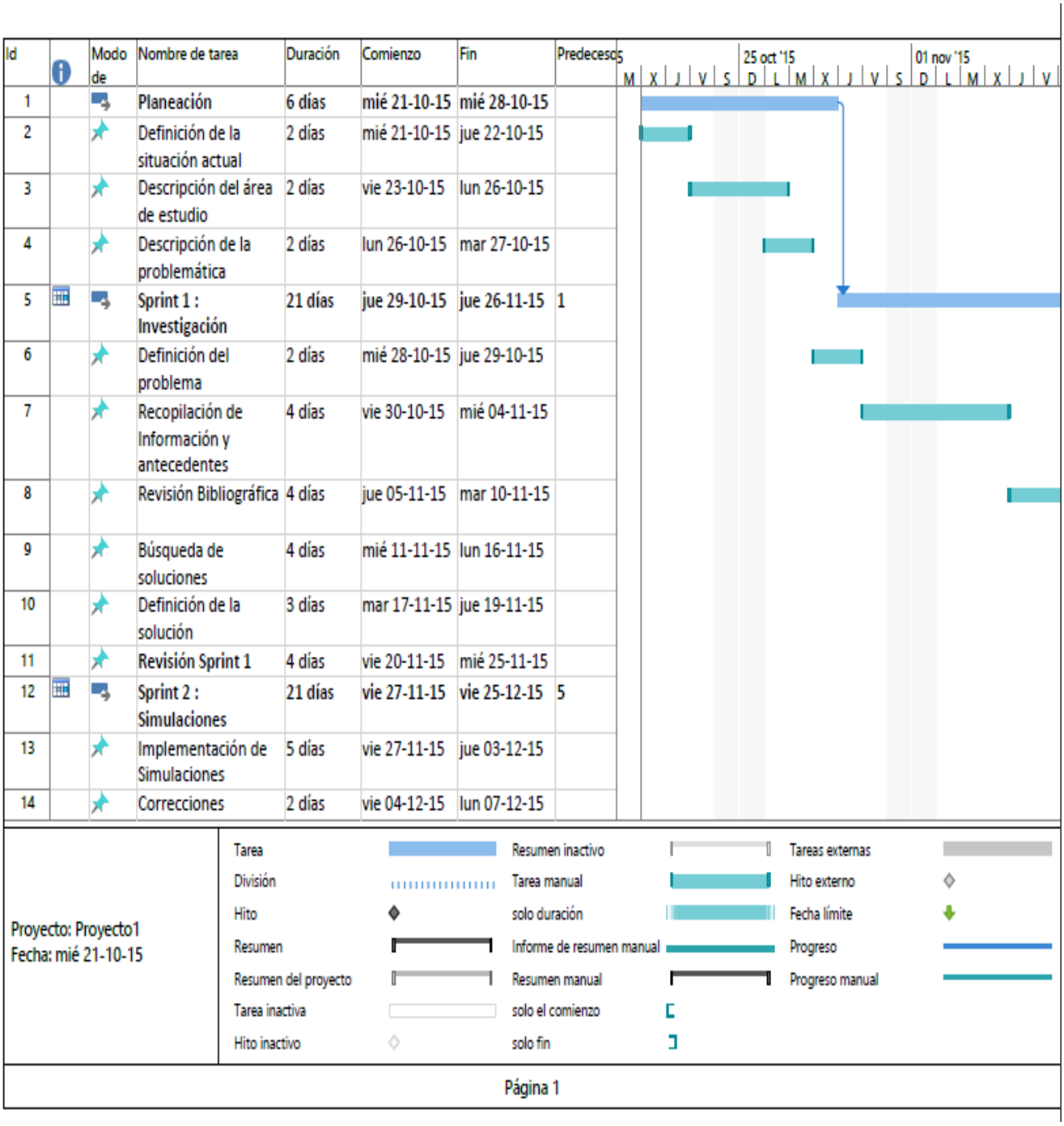
Luego de instalar todo lo necesario para empezar la instalación, debemos hacer los siguientes pasos:

- **Paso 1:** Dirigirse a la página oficial de ns-3, que es https://www.nsnam.org para revisar la documentación oficial.
- **Paso 2:** Abrir la terminal y ejecutar los siguientes comandos:
  mkdir ns3 //Crea un directorio
- **Paso 3:** Luego ocuparemos la forma de instalación tipo Bake (mediante comandos). También existe el modo Tarball (instalar desde un archivo comprimido)
- **Paso 4:** Ingresaremos el siguiente comando:
  hg clone http://code.nsam.org/bake //descarga los archivos necesarios en la carpeta llamada bake, ubicada en el directorio que creamos en el paso 2

- **Paso 5:** Ingreso de comando para la exportación de paquetes esenciales:
  export BAKE_HOME='pwd'/bake //exportar primer paquete
  export PATH-$PATH:$BAKE_HOME //exportar segundo paquete
  export PATH-$PYTHONPATH=$PYTHONPATH:$BAKE_HOME //exportar tercer paquete
- **Paso 6:** cd bake //Ingresar al nuevo directorio que es ns3/bake
- **Paso 7:** Configurar el archivo bake.py con el siguiente comando:
  ./bake.py configure .e ns-3.20
- **Paso 8:** Chequear todo lo necesario para comenzar la instalación de ns-3 con el comando
  ./bake.py check
- **Paso 9:** Descargar la aplicación y paquetes extras necesarios con el comando
  ./bake.py download //Descarga ns-3.20 y netanim-3.105 (visor de simulación)
- **Paso 10:** Compilar .bake con el comando:
  ./bake.py build
- **Paso 11:** Compilar ns3, ingresando a la carpeta ns3 ubicada en la carpeta source/ns-3.20 con el comando:
  cd/bake/source/ns.3.205 ./wafbuild
- **Paso 12:** Testear archivos de ns-3
  cd/bake/source/ns.3.205 ./test.py –c core
- **Paso 13:** Ejecutar archivo de ejemplo de ns-3 con el comando:
  cd/bake/source/ns.3.205 ./waf –run hello-simulator
- **Paso 14:** ns-3 ha quedado operativo en su sistema.

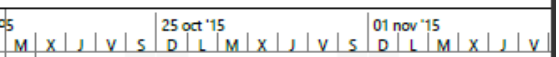
17 Anexo Planificación Inicial

Se adjunta Carta Gantt con la planificación inicial del proyecto.

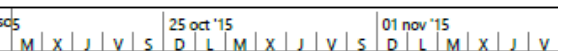


Id	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Planeación	6 días	mié 21-10-15	mié 28-10-15	
2		Definición de la situación actual	2 días	mié 21-10-15	jue 22-10-15	
3		Descripción del área de estudio	2 días	vie 23-10-15	lun 26-10-15	
4		Descripción de la problemática	2 días	lun 26-10-15	mar 27-10-15	
5		Sprint 1 : Investigación	21 días	jue 29-10-15	jue 26-11-15	1
6		Definición del problema	2 días	mié 28-10-15	jue 29-10-15	
7		Recopilación de Información y antecedentes	4 días	vie 30-10-15	mié 04-11-15	
8		Revisión Bibliográfica	4 días	jue 05-11-15	mar 10-11-15	
9		Búsqueda de soluciones	4 días	mié 11-11-15	lun 16-11-15	
10		Definición de la solución	3 días	mar 17-11-15	jue 19-11-15	
11		Revisión Sprint 1	4 días	vie 20-11-15	mié 25-11-15	
12		Sprint 2 : Simulaciones	21 días	vie 27-11-15	vie 25-12-15	5
13		Implementación de Simulaciones	5 días	vie 27-11-15	jue 03-12-15	
14		Correcciones	2 días	vie 04-12-15	lun 07-12-15	

Id	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
15		Implantación de casos	5 días	mar 08-12-15	lun 14-12-15	
16		Correcciones	2 días	mar 15-12-15	mié 16-12-15	
17		Revisión Profesor Guía	1 día	jue 17-12-15	jue 17-12-15	
18		Correcciones	2 días	vie 18-12-15	lun 21-12-15	
19		Interpretación y Conclusión de Resultados	3 días	mar 22-12-15	jue 24-12-15	
20		**Revisión Sprint 2**	1 día	vie 25-12-15	vie 25-12-15	
21		**Sprint 3: Aplicación**	**5 días**	**lun 28-12-15**	**vie 01-01-16**	12
22		Especificación de requerimientos de SW	1 día	lun 28-12-15	lun 28-12-15	
23		Alcances	1 día	mar 29-12-15	mar 29-12-15	
24		Objetivos del SW	1 día	mié 30-12-15	mié 30-12-15	
25		Descripción del producto	1 día	jue 31-12-15	jue 31-12-15	
26		Requerimientos Específicos	1 día	vie 01-01-16	vie 01-01-16	
27		**Factibilidad**	**5 días**	**lun 04-01-16**	**vie 08-01-16**	21
28		Factibilidad Técnica	1 día	lun 04-01-16	lun 04-01-16	
29		Factibilidad Operativa	1 día	mar 05-01-16	mar 05-01-16	



Id	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
30		Factibilidad Económica	1 día	mié 06-01-16	mié 06-01-16	
31		Casos de Uso	1 día	jue 07-01-16	jue 07-01-16	
32		**Revisión Sprint 3 : Analisis y Factibilidad**	1 día	vie 08-01-16	vie 08-01-16	
33		**Diseño**	**7 días**	**lun 11-01-16**	**mar 19-01-16**	27
34		Diseño funcional	2 días	lun 11-01-16	mar 12-01-16	
35		Diseño interfaz usuario	2 días	mié 13-01-16	jue 14-01-16	
36		**Revisión Sprint 3: Diseño**	3 días	vie 15-01-16	mar 19-01-16	
37		**Pruebas**	**7 días**	**mié 20-01-16**	**jue 28-01-16**	33
38		Generar pruebas de aplicación	2 días	mié 20-01-16	jue 21-01-16	
39		Documentar resultados	2 días	vie 22-01-16	lun 25-01-16	
40		**Revisión Sprint 3: Pruebas**	3 días	mar 26-01-16	jue 28-01-16	
41		**Entrega de Proyecto Simulación en NS3 de una red hibrida P2P para soportar Cloud Colaboration**	**1 día**	**vie 29-01-16**	**vie 29-01-16**	37



Proyecto: Proyecto1
Fecha: mié 21-10-15

Tarea	Resumen inactivo	Tareas externas
División	Tarea manual	Hito externo
Hito	solo duración	Fecha límite
Resumen	Informe de resumen manual	Progreso
Resumen del proyecto	Resumen manual	Progreso manual
Tarea inactiva	solo el comienzo	
Hito inactivo	solo fin	

Página 3