PROTOCOLO DE TRANSFERENCIA MASIVA DE DATOS DESDE DISPOSITIVOS HARDWARE HACIA REPOSITORIOS DE DATOS

Luis Alejandro Torres Niño

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingeniería de sistemas e Informática

Bucaramanga

2016

PROTOCOLO DE TRANSFERENCIA MASIVA DE DATOS DESDE DISPOSITIVOS HARDWARE HACIA REPOSITORIOS DE DATOS

Luis Alejandro Torres Niño

Trabajo de Investigación para optar por el título de:

Magíster en Ingeniería de Sistemas e Informática

Director:

Ph.D. Luis Alberto Núñez

Codirector

Ph. D. Carlos Jaime Barrios

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingeniería de sistemas e Informática

Bucaramanga

2016

DEDICATORIA

A Heidy, quien siempre ha estado a mi lado apoyando locuras como ésta y que sin ella, su paciencia y su amor, jamás hubiera salido adelante.

Charlotte gracias por acompañarme y tener tanta paciencia con este loco. Siempre has sido un gran apoyo.

Ustedes dos son los pilares de mi vida.

AGRADECIMIENTOS

En particular quiero dar gracias al Profesor Carlos Jaime Barrios que sin su ayuda éste proyecto jamás hubiera llegado hasta el final, sin su guía, ayuda y sobre todo compresión en muchas situaciones difíciles las cosas hubieran sido más difíciles de lo que ya pueden ser con una maestría. Al profesor Luis Núñez por toda su ayuda en este proceso. No quiero olvidar a todos esos locos agrupados en el SC3 que con ideas, comentarios y todo tipo de ayuda hicieron más llevadero todo este proceso.

A mis compañeros de maestría, siempre aprendo cosas nuevas de todos ellos.

Los resultados de los experimentos presentados en ésta publicación, fueron obtenidos usando la plataforma GridUIS-2 soportada por el Centro de Supercomputación y Cálculo Científico de la Universidad Industrial de Santander (SC3-UIS www.sc3.uis.edu.co) y está acción es soportada por la Vicerrectoría de Investigación y Extensión de la UIS (VIE-UIS).

CONTENIDO

TABLA DE FIGURAS

RESUMEN

TITULO: PROTOCOLO DE TRANSFERENCIA MASIVA DE DATOS DESDE DISPOSITIVOS HARDWARE HACIA REPOSITORIOS DE DATOS[1].

AUTOR: LUIS ALEJANDRO TORRES NIÑO[2].

PALABRAS CLAVE: UDT, SWORD, TRANSFERENCIA DE DATOS, REPOSITORIOS DE DATOS.

DESCRIPCIÓN:

La producción de datos científicos por parte de los grupos de investigación ha crecido de forma exponencial en los últimos años, haciendo que el protocolo TCP sea insuficiente para lograr realizar la transferencia de estos grandes volúmenes de datos. Por otra parte, una vez transferidos los datos, es necesario catalogarlos y almacenarlos en bases de datos llamadas repositorios. Ahora bien, estas tareas de transferencia, catalogación y almacenamiento de datos son realizadas de forma manual por la gran mayoría de las comunidades científicas modernas, por lo cual se hace necesario proporcionar "inteligencia" a los instrumentos generadores de datos.

El estándar de interoperabilidad SWORD permite realizar ingestiones remotas desde equipos hardware hacia repositorios. Esta herramienta fue tomada para desarrollar el primer modelo cliente/servidor para dar solución al problema. Se observa que SWORD se encuentra limitado en el contexto del presente trabajo y por tanto, se replantea un nuevo desarrollo. UDT es un protocolo que permite la transferencia de datos a través de redes WAN y que permite la utilización de casi todo el ancho de banda disponible. En base a esto, se construye un nuevo modelo cliente/servidor en donde se utiliza UDT para transferir datos y metadatos desde dispositivos hardware (instrumentos generadores de datos) hacia los repositorios de datos. El modelo cliente/servidor basado en UDT es integrado al repositorio de datos utilizando las herramientas de administrativas dadas por el software de repositorio de datos DSpace.

[1] Trabajo de Investigación.

[2] Facultad de Ingenierías Físico-mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Luis A. Núñez, Doctor en Física. Codirector: Carlos Jaime Barrios, Doctor en Informática.

ABSTRACT

TITLE: DATA MASSIVE TRANSFER PROTOCOL FROM HARDWARE DEVICE TO DATA REPOSITORIES[3].

AUTHOR: LUIS ALEJANDRO TORRES NIÑO[4].

KEYWORDS: UDT, SWORD, DATA TRANSFER, DATA REPOSITORIES

DESCRIPCTION:

Producing scientific data by research groups has grown exponentially in recent years, making the TCP protocol is insufficient to achieve the transfer of these large volumes of data. On the other hand, once transferred the data, it is necessary to label them and store them in databases called repositories. However, these tasks transfer, cataloging and storing data are performed manually by the vast majority of modern scientific communities, so it is necessary to provide "intelligence" to the data generators instruments.

The standard of interoperability SWORD allows remote intakes of data from hardware equipment to repositories. This tool was used to develop the first client/server model to solve the problem. Note that SWORD is limited in the context of this study and therefore, we develop a new model. UDT is a protocol that allows the transfer of data through networks WAN and allowing the use of almost all the bandwidth available. On this basis, builds a new client/server model where the UDT Protocol is used to transfer data and metadata from the hardware device (data generating instruments) to data repositories. The UDT-based client/server model is integrated to the data repository using the administrative tools given by the data repository software, DSpace.

[3] Master's degree research work.

[4] Department of Physical-mechanical Engineering. School of Systems Engineering and Computer Science. Advisor: Luis A. Núñez, Ph.D. in Physics. Co-Advisor: Carlos Jaime Barrios, Ph.D in Computer Science.

INTRODUCCIÓN

La transferencia masiva de datos así como su almacenamiento se han convertido en un tema de gran interés para las ciencias computacionales modernas. Las bajas tasas de transferencia obtenidas por el protocolo TCP han hecho que surjan nuevos protocolos o al menos variaciones de éste. Se han formulado nuevos algoritmos para el control de transmisión de TCP logrando obtener tasas de 10 Gbps en redes de alta velocidad.

Por otra parte, se ha utilizado el protocolo UDP como alternativa para incrementar las velocidades de transferencia mediante la asignación de algoritmos encargados de la verificación y recepción de todos los paquetes transmitidos. Este es el caso del protocolo UDP (UDP based Data Transfer Protocol), el cual ha demostrado su eficiencia con tasas de transferencia de 940 Mbps en una red Gigabit.

Ahora en cuanto al almacenamiento, catalogación y diseminación de los datos es realizado mediante la implementación de repositorios digitales y sus herramientas estandarizadas. Tal es el caso del estándar de interoperabilidad SWORD que permite no solo realizar estas tareas sino que además realiza la transferencia de los datos a través de la red mediante la utilización del protocolo TCP.

Estas dos problemáticas, transferencia y almacenamiento de datos, se entrelazan al hablar de comunidades de investigación, en particular aquellas de e-Ciencia. En éstas, los investigadores pueden pertenecer a diferentes instituciones y países y por ello, los datos deben estar alojados en un repositorio que les permitan acceder a los datos sin importa su ubicación. Ahora por otra parte, estas comunidades en su mayoría cuentan con instrumentación que puede llegar a generar enormes cantidades de datos y muchas veces, el tamaño de los archivos generados puede llegar a los GB, incrementando aún más la complejidad de la problemática.

La transferencia, catalogación almacenamiento y diseminación de los datos generados por los instrumentos de las comunidades de e-Ciencia son realizadas de forma manual. Esto produce que los datos lleven más tiempo en estar en manos de los investigadores, además de correrse el riesgo de la generación incorrecta de metadatos causando datos mal catalogados. Por esto, se hace necesario proporcionarle a los instrumentos la capacidad de realizar estas tareas de forma autónoma y siendo este el objetivo del presenta trabajo de investigación.

En el primer capítulo se presenta el contexto de la investigación mostrando la problemática abordada. En el segundo capítulo se presentan los antecedentes y trabajos relacionado con respecto a la transferencia masiva de datos, repositorios

digitales y los dispositivos que son usados para la generación de ellos. También se discute sobre el estándar de interoperabilidad SWORD y el protocolo de transferencia UDT.

En el tercer capítulo se describe detalladamente los modelos de transmisión de datos planteados tanto para SWORD como para UDT. Igualmente se discute ciertas restricciones presentes en las librerías de desarrollo para nuevos clientes SWORD así como de las ventajas de usar el modelo UDT.

En el cuarto capítulo se muestra como se realizó la implementación de los modelos sobre diferentes arquitecturas hardware y de presentan los requisitos y configuraciones necesarias para la implementación tanto de los clientes como de los servidores,

En el quinto y sexto capítulo se presentan los resultados obtenidos al evaluar ambos modelos y se hace un contraste entre ellos. Se muestra la implementación de los modelos en la

Finalmente en el séptimo capítulo se enumeran las conclusiones obtenidas en este proyecto y se postulan los trabajos futuros en base al estudio realizado durante este trabajo.

1. CONTEXTO DEL TRABAJO DE INVESTIGACIÓN

1.1. PLANTEAMIENTO DEL PROBLEMA

En los últimos años, la transferencia masiva de datos sobre las redes se ha convertido es un parte integral de las ciencias computacionales modernas. Ahora bien, esta transferencia es realizada con el fin de aprovechar un sin fin de recursos computacionales que generan nuevos conjuntos de datos (datos tratados, datos simulados y/o datos de visualización) [1].

Por otra parte y no menos importante, está el almacenamiento de los datos y su correcta catalogación y diseminación para su uso por parte de las comunidades científicas o los miembros de cada uno de los proyectos de investigación. Estas tareas son llevadas a cabo mediante almacenes de datos denominados repositorios de datos de investigación y cabe aclarar que por lo general, estos repositorios no se encuentran en la misma ubicación de los detectores o de los instrumentos utilizados para la generación u obtención de los datos crudos [2].

Las comunidades de ciencia modernas han evolucionado hacia lo que se conoce como e-Ciencia[5] [3] y en un contexto más general, la ciencia actual se mueve en un entorno cada vez más colaborativo donde grupos de diferentes países no solo comparten bases de datos (repositorios) sino también el uso de instrumentos que son controlados de forma remota. Igualmente, se comparten recursos de computación de alto rendimiento (*High Perfomance Computing*) para la utilización de herramientas de análisis y simulación [4]. Este nuevo escenario es lo que se conoce como e-infraestructura, en donde la infraestructura de investigación combina elementos bien conocidos tales como la red de comunicación y los recursos computacionales [5].

Es importante resaltar que los proyectos científicos desarrollados por estas comunidades de e-Ciencia han abordado objetivos cada vez más ambiciosos. Estos proyectos exigen mejores y mayores recursos computacionales tanto para la resolución de problemas de gran complejidad como para la transferencia, manejo y análisis de grandes volúmenes de datos que son generados por sus instrumentos [6][7].

[5] e-Ciencia: Definida como mejora (*enhancement*) del potencial de la Ciencia mediante la explotación de diferentes recursos distribuidos geográficamente e interconectados en red

Con referencia a la transferencia de datos, ésta se ha caracterizado por presentar bajos rendimientos que son provocados por una gran variedad de razones, entre las cuales destacan las configuraciones inadecuadas entre los host de envío y recepción, perdida de paquetes causados por redes de baja calidad, buffers pequeños, firewall, entre otros [8]. Ahora bien, los datos transferidos por la gran mayoría de los instrumentos no son almacenados en repositorios de forma automática, por lo cual deber ser transferidos, catalogados y almacenados de forma manual [2].

Las comunidades de investigación actuales en particular aquellas de e-Ciencia, han requerido tener conectados a la red la mayor cantidad de recursos posibles y que permitan proporcionar una colaboración mucho más efectiva. Llama la atención el hecho que muchas tareas se realicen de forma manual, en particular aquellas concernientes a transferencia, catalogación y almacenamiento de los datos haciendo que la interconexión de recursos no produzca los resultados esperados para hacer estas comunidades mucho más eficientes [2][9]. Por tanto, se hace necesario brindarle "inteligencia" a esta infraestructura, en particular a los instrumentos encargados de la generación de datos, siendo estos capaces de realizar transferencias de grandes volúmenes de datos de forma intensiva sin presentar los bajos rendimientos mencionados anteriormente. De igual forma, se hace necesario brindar la capacidad de tomar la configuración del instrumento para generar los metadatos que hacen único a cada archivo de datos obtenido. Esta "inteligencia" permitiría a los instrumentos interoperar con los repositorios de datos de tal forma que puedan realizar las tareas de transferencia, catalogación y diseminación de forma automática.

Un buen ejemplo práctico de este tipo de comunidades de e-Ciencia es la comunidad LAGO (*Latin American Giant Observatory*) en donde dentro de su e-infraestructura se cuenta con una gran cantidad de detectores y un repositorio para los datos generados y simulados. De manera general, LAGO cuenta con detectores en 8 países de Latinoamérica (México, Guatemala, Colombia, Venezuela, Ecuador, Perú, Bolivia y Argentina) y con lo cual puede llegar a producir enormes cantidades de datos, lo cual depende del fenómeno registrado. Estos deben ser transferidos, catalogados y almacenados en el repositorio y de esta manera permitirles a los investigadores llevar a cabo sus estudios mediante la utilización de varios recursos computacionales enlazados en una e-infraestructura [9]. Cabe agregar que dentro de la comunidad LAGO y otras semejantes, estas tareas son realizadas por personal que puede incurrir en errores de factor humano y la "inteligencia" propuesta para los instrumentos proveería una manera rápida y viable de automatizar las tareas de

transferencia, catalogación y almacenamiento de los datos producidos evitando caer en algún fallo durante la realización de estas tareas.

1.2. OBJETIVOS

1.2.1. OBJETIVO GENERAL

- Proponer un protocolo de transferencia de datos para el binomio detector-repositorio que tenga la capacidad de soportar datos de gran tamaño y que realice las tareas de catalogación de forma automática.

1.2.2. OBJETIVOS ESPECIFICOS

- Proponer un modelo de interacción cliente-servidor para la transferencia de datos y metadatos desde detectores hacia repositorios, en donde se permita la transferencia masiva e intensiva apoyándose en protocolos de trasferencia tipo SWORD[6].

- Proponer una técnica que permita integrar el servidor del modelo anteriormente propuesto al repositorio de datos y otra para integrar el cliente en los instrumentos generadores de datos.

- Establecer y validar un método de implementación para el protocolo de transferencia de datos propuesto.

[6] (*Simple Web-service Offering Repository Deposit*) es un estándar de interoperabilidad que permite a los repositorios digitales aceptar el depósito de contenido desde múltiples fuentes en diferentes formatos.

2. ANTECEDENTES Y TRABAJOS RELACIONADOS

En este capítulo se explican los fundamentos teóricos, antecedentes y trabajos relacionados en cuanto a la transferencia masiva de datos de gran tamaño, los repositorios digitales y su uso en el almacenamiento de los mismos. Como parte final del capítulo, se explican los diferentes elementos hardware utilizados para el registro y transferencia de datos involucrados en este proyecto.

2.1. TRANSFERENCIA MASIVA DE DATOS

La ciencia moderna está conformada por diferentes colaboraciones científicas con instrumentos que pueden llegar a producir Petabytes de datos que son almacenados junto a los datos simulados para posteriores análisis. De manera general, los científicos pertenecientes a estas comunidades tienen prioridad en el uso de los datos, pero después de un tiempo determinado por las colaboraciones, estos son liberados para poder ser usados por diferentes investigadores a lo largo y ancho del planeta [10].

Puede parecer lógico y eficiente ubicar las herramientas de análisis de datos tales como recursos de cómputo (software especializado para análisis de datos y recursos de HPC) junto con la fuente que los ha producido, pero actualmente este no es un escenario típico, en particular cuando se habla de aquellas organizaciones multidisciplinarias y colaboraciones científicas que tienen miembros en diferentes instituciones y países, por ello, las soluciones distribuidas son mucho más comunes en esta escala científica [11]. Basados en esta necesidad, se ha hecho necesario la implementación de herramientas y protocolos eficientes para mover grandes cantidades de datos científicos sobre redes de alta velocidad, las cuales enlazan tales colaboraciones [12].

Uno de los ejemplos más representativos de este tipo de colaboraciones es el CERN y su *Large Hadron Collider (LHC)*[7], una colaboración en el área de la Física de Altas Energías, la cual ha sido la base y fuerza impulsora para el despliegue de las conexiones de banda ancha en el mundo de la investigación y la educación, planteando retos importantes en diferentes campos, en particular en la generación, distribución y análisis de datos [10].

Muchas otras disciplinas de investigación enfrentan los mismos desafíos planteados anteriormente. En el campo de la genómica, los costos de la secuenciación han

[7] The Large Hadron Collider: http://lhc.web.cern.ch/lhc/

disminuido drásticamente permitiendo que el volumen de datos producidos crezca de forma exponencial. Otro campo importante, es el área de la climatología, en donde los investigadores deben analizar conjuntos de datos simulados y observados en instalaciones ubicadas en diferentes lugares del planeta; se espera que los datos superen los 100 Exabytes para el 2020 [10][13].

La generación actual de datos por parte de los instrumentos utilizados por estas colaboraciones puede llegar a producir 300 o más Megabytes/segundo y se espera que próximamente pueda llegar a volúmenes aún más altos [10]. Entre otros proyectos de gran escala que pueden llegar a generar esta gran cantidad de datos, se encuentra el *International Thermonuclear Experiment Reactor (ITER)*[8], el *Square Kilometre Array (SKA)*[9], un radiotelescopio de gran tamaño que puede llegar a generar una cantidad similar de datos que el LHC.

Con el fin de superar las limitaciones con las que cuentan las tecnologías de redes existentes y para apoyar la creciente demanda en la transferencia de datos, se ha hecho necesario implementar nuevos enfoques y alternativas al omnipresente Protocolo de Control de Transmisión (TCP), el cual es bien conocido por tener problemas de rendimiento en grandes distancias y en redes de gran ancho de banda [10][14][15].

Una de las características más importantes de TCP es asegurar que los datos que emite el cliente sean recibidos por el servidor sin errores y en el mismo orden en el que fueron emitidos, incluso si el servicio prestado por el protocolo IP no es confiable. De manera más detallada, el algoritmo de congestión del protocolo TCP es el que le permite realizar estas transmisiones de forma segura es el que provoca la pérdida de rendimiento en la transferencia [14].

Ahora bien, TCP permite ser ajustado y con un algoritmo de congestión más apropiado puede realizar transferencias a más de 10 Gbps. Cabe señalar que TCP a esta tasa de transferencia puede llegar a sobrecargar el sistema utilizando un núcleo completo de procesador y si la velocidad de la red sigue creciendo como hasta el momento, se pone en duda la viabilidad de éste en un futuro. Por otra parte, también es un gran reto la carga administrativa que representa las constantes adaptaciones de TCP a diferentes escenarios de red para lograr realizar las trasferencias de grandes volúmenes de datos [10].

Actualmente, para la transferencia de grandes volúmenes de datos científicos se utilizan herramientas tales como Globus Online/GridFTP[10] que puede mover datos mediante *streams* TCP en paralelo. Pero a pesar de estas herramientas y de la

[8] ITER: http://www.iter.org/

[9] SKA: http://www.skatelescope.org/

[10] Globus GridFTP: http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/

utilización de *sockets* TCP para aumentar el rendimiento, solo se ha conseguido transferencias de alrededor de 13 Gbps. Cabe señalar que es un futuro cercano estas limitaciones no permitirán a sitios de *Big Data* realizar las transferencia de los conjuntos de datos en el tiempo deseado [10].

Un ejemplo de ese futuro, es el procesamiento de datos para el proyecto del telescopio SKA. El diseño actual poseerá un total de 2.900 sensores en operación e iniciará la recolección de datos en 2020, con una tasa de datos total de 15.000 Tb. Los datos de los receptores probablemente serán transmitidos directamente a un correlacionador (procesador de datos) para su respectivo filtrado. Una vez realizada esta tarea, los datos se reducirán a 400 Tb y enviados a centros de supercomputación para su respectivo análisis, los cuales pueden encontrarse a una distancia superior a los 1.000 Km. Un ejemplo a más corto plazo es el experimento *Bell II High Energy Physics* que se encuentra actualmente en construcción en Japón[11]. Este experimento podría llegar a generar 1.8 GB/s cuando se inaugure en 2016. Los datos producidos por éste se analizarán por científicos en 65 instituciones de 17 países diferentes y una copia completa del conjunto de datos será enviada a los estados unidos de forma recurrente.

Tanto para el proyecto SKA, como el Bell II, así como para otros de menor envergadura, se requiere de protocolos más eficientes para la transferencia de datos que los que se encuentran disponibles actualmente. Por esto, alternativas como UDT (*UDP-based Data Transfer Protocol*)[12] surgieron y proporcionaron un uso más eficiente del ancho de banda, superando los niveles normales de TCP, logrando llegar a niveles del 90% del total del ancho de banda disponible [16].

UDP-based Data Transfer Protocol (UDT)

UDP-based Data Transfer Protocol es un protocolo de alto rendimiento para la transferencia de grandes volúmenes de datos en redes WAN. Esencialmente, este protocolo realiza el envió confiable de datos agregando mecanismos de control de congestión a UDP (*User Datagram Protocol*) junto con algoritmos de verificación de recepción de paquetes. Por otra parte, UDT cuenta con una librería que implementa en la capa de usuario el protocolo y permite, como valor agregado, modificar el control de congestión preestablecido, con lo cual puede adaptarse a diferentes condiciones. Actualmente UDT se encuentra en su versión 4 añadiendo nuevas

[11] Belle II: http://belle2.kek.jp/
[12] UDT: http://udt.sourceforge.net/

características como el soporte a alta concurrencia y cruce de cortafuegos, así como permitir múltiples conexiones al mismo puerto UDP.

En cuanto al diseño del protocolo, se definen dos tipos de paquetes; los paquetes de control y los de datos, los cuales se diferencian en el valor del primer bit de la cabecera.

A continuación se especifica y detalla la estructura de los campos de la cabecera de los paquetes para cada tipo de mensaje:

- **Paquete de datos** (Ver Figura 1.a)**:**
    - **Flag bit:** El primer bit de la cabecera y se encarga de hacer la distinción entre un paquete de datos o uno de control. Para el caso de un paquete de datos es cero.
    - **Número de secuencia:** Realiza la misma función que el número de secuencia en TCP, evitando duplicados y permitiendo ordenar los mensajes cuando llegan al receptor.
    - **Número de mensaje:** Usado por la aplicación para el envío de mensajes y éste puede ocupar más de un paquete, por tanto será tratado como un bloque de información pero con un sentido particular.
    - **Campo FF:** Indica que tipo de paquete es el que se ha transferido dentro de un mensaje. Para el primer paquete se usa el valor 10, para un paquete intermedio se usa 00, para el último paquete del mensaje se usa 01 y en caso de que el mensaje ocupe solo un paquete se usa 11.
    - **Timestamp:** Marca de tiempo relativa al paquete la cual se inicia con el inicio de la conexión y se toma en microsegundos.
    - **Campo O (orden):** Es un indicador utilizado para definir si los paquetes deben ser entregados en orden y por tanto un paquete deba retrasarse hasta que los anteriores sean entregados.

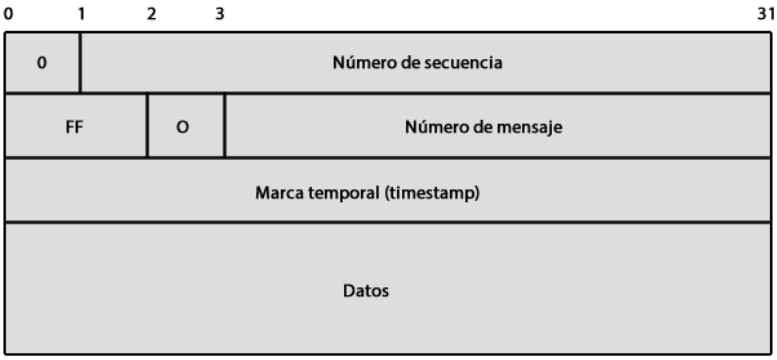
- **Paquete de control** (Ver Figura 1.b)**:**
    - **Flag bit:** Como se mencionó antes, distingue entre un paquete de control y uno de datos, para el caso de un paquete de control el valor es 1.
    - **Tipo:** Existen siete tipos diferentes para la gestión del protocolo y un octavo para paquetes personalizados por la aplicación o definidos por el usuario:
        - **Handshake:** Es usado para establecer una nueva conexión y para la negociación de las características que definirán esta.

        - **ACK (*Acknowledgement*):** Mensaje de reconocimiento de paquete recibido.
        - **ACK2 (*Negative Acknowledgement*):** mensaje de reconocimiento de un mensaje de reconocimiento.
        - **Informe de pérdida:** Utilizado para informarle al emisor de que un paquete que ha enviado se ha perdido.
        - **Keep-alive:** Son paquetes que se envían periódicamente entre el emisor y el receptor para informar que ambos siguen “vivos” y que la conexión debe permanecer abierta.
        - **Shutdown:** Utilizado para cerrar una conexión.
        - **Paquete descartado:** Utilizado para pedirle al receptor que elimine un paquete que ha sido enviado.
        - **Paquete definido por el usuario:** Es utilizado para definir los paquetes de control definidos por la aplicación y desarrollados con la librería. Cabe aclarar que este tipo de paquetes deben ser tratados por un código definido por el desarrollador.
        - **Advertencia de congestión:** Es utilizado para informar al emisor que se ha presentado una congestión en la red y por tanto debe reducir la tasa de envío.
    - **Tipo extendido:** Es usado en los paquetes de control definidos por el usuario.
    - **Número de secuencia del ACK:** Utilizado para enumerar cada ACK que se produzca siendo este independiente del número de secuencia del paquete que reconocen.
    - **Timestamp:** Es una marca de tiempo relativa al paquete la cual se inicia con el inicio de la conexión y se toma en microsegundos.
    - **Información del paquete de control:** Información perteneciente al paquete de control.

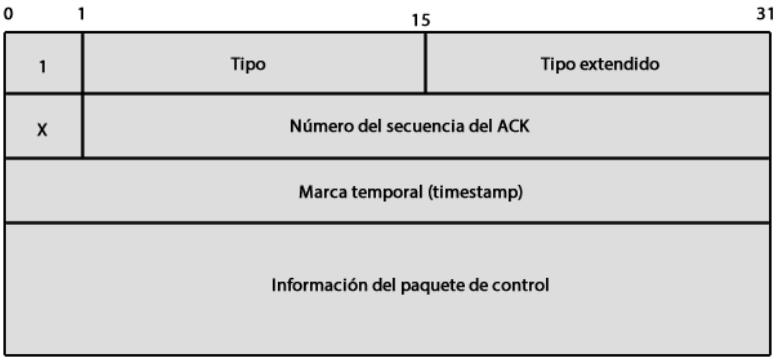
UDT proporciona dos tipos de inicio de conexión: el modo cliente/servidor y el modo *rendezvous*. Para el primero de ellos, el cliente envía un paquete *handshake* al servidor, en el cual se indica la versión de la librería UDT, el tipo de socket (SOCK_STREAM o SOCK_DGRAM), un número aleatorio para el inicio de la secuencia, el tamaño máximo de paquete y el tamaño máximo de la ventana de flujo. Una vez recibido el paquete por parte del servidor, se comprueba la versión de la librería y el tipo de socket, se compara el tamaño de paquete recibido con el del servidor, tomando como propio el valor más pequeño y el valor resultante se devuelve al cliente en forma de respuesta con un paquete *handshake*, junto con el número de secuencia inicial y el tamaño máximo de la ventana de flujo del servidor.

Al completar el intercambio de mensajes, el cliente y el servidor se encuentran listos para enviar/recibir datos.

Figura 1. ***Estructura de los campos de la cabecera UDT.*** *El primer bit del paquete de la cabecera es una bandera para indicar si se trata de un paquete de datos (0) o un paquete de control (1). Los paquetes de datos contienen un número de secuencia de 31 bits, 29 bits para el número de mensaje y 32 bits de marca temporal. En la cabecera del paquete de control, los bits del 1 al 15 son información del tipo de paquete y del bit 16 al 31 se puede usar para tipos definidos por el usuario. La información detallada de paquete de control depende del tipo de paquete. [16]*



a. Paquete de datos



b. Paquete de control

En una configuración tradicional cliente/servidor la conexión requiere que el servidor se inicie primero y posteriormente el cliente se conecte a éste, haciendo que sea interrumpida la solicitud del cliente al servidor cuando ambas máquinas se encuentran tras un cortafuego. Como se mencionó anteriormente, ésta es una de las nuevas características añadidas a la cuarta versión del protocolo y es el segundo tipo de inicio de conexión, denominado ***rendezvous***. Para este tipo de conexión, ambos nodos tratan de establecer la conexión de forma simultánea y el que reciba el mensaje en primer lugar, es quien iniciará la fase de configuración.

En cuanto al control de fiabilidad/reconocimiento de los mensajes usado por UDT para el control de congestión y la confiabilidad de los datos, el protocolo implementa dos tipos: **reconocimiento selectivo en base a tiempo** que genera un acuse de recibo en un intervalo fijo, si hay nuevos paquetes de datos recibidos continuamente. Esto quiere decir que cuando se cuenta con transferencias a alta velocidad, el control de tráfico consume una muy pequeña porción del ancho de banda, mientras que en anchos de banda muy pequeños, UDT actúa como un protocolo que reconoce cada paquete de datos como lo realiza TCP.

El otro tipo de control de fiabilidad/reconocimiento es el **modo de mensajes de fiabilidad parcial**, en donde el emisor asigna a cada mensaje saliente una marca de tiempo (*timestamp*). Si el tiempo de vida de un mensaje expira en el momento en que un paquete de ese mensaje debe ser enviado o retransmitido, el emisor enviará un mensaje de paquete caído al receptor. Al recibirse este tipo de mensaje, el receptor considerará como recibidos todos los paquetes del mensaje y marcara este como mensaje caído.

Finalmente y para tener una alta fiabilidad, UDT alberga una lista de paquetes perdidos para recordar el número de secuencia del paquete y la cual será usada por el receptor para enviar paquetes ACK y NACK de forma periódica informando al emisor los paquetes que han sido recibidos y cuales han sido perdidos.

Como se mencionaba, el control de fiabilidad/reconocimiento es usado por el de congestión y éste a su vez se basa en un control de la tasa de transferencia y un control de flujo basado en ventanas. El control de la tasa es un mecanismo para sincronizar el periodo de envío entre paquetes mientras que el control de flujo es para especificar un umbral dinámico que trata de impedir la generación de paquetes NACK.

El algoritmo de control de congestión en UDT se denomina DAIMD (*Decreasing Aditive Increase, Multiplicative Decrease*). Este algoritmo incrementa la tasa de envío de paquetes x en $\alpha(x)$, cuando para cada intervalo, no se presentan perdidas de paquetes, demoras o algún tipo de NACK. De esta forma, la tasa queda definida como:

$$x = x + \alpha(x)$$

En donde $\alpha(x)$ es decreciente y que tiende a cero cuando x tiende a infinito.

Para cuando se presentan NACK la tasa de envío decrece en un factor constante β $(0 < \beta < 1)$:

$$x = (1 - \beta) * x$$

De manera matemática el control de congestión se establece como:

$$a(x) = 10^{|\log(L-C(x)|-\tau} * \frac{1500}{S} * \frac{1}{SYN}$$

Donde:

x: Tasa de envió [paquetes/segundo].

L: Capacidad del canal o ancho de banda [bits/segundo]

S: Tamaño del paquete UDT [bytes]

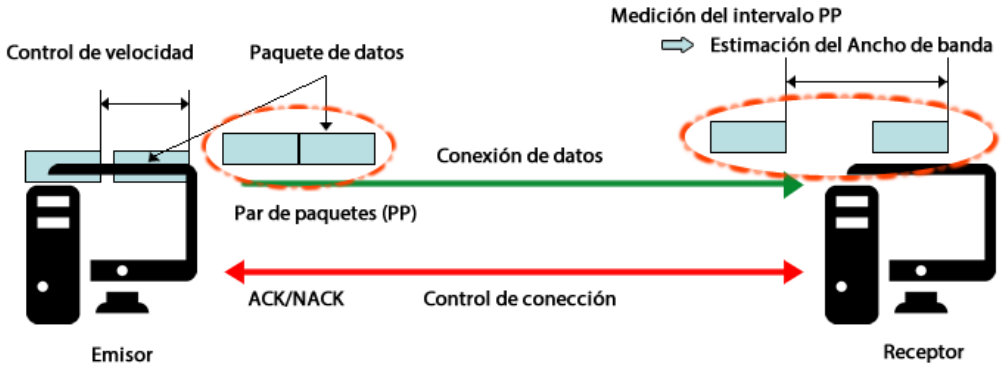
$C(x)$: Función de conversión de unidades [paquetes/segundo bits/segundo], $C(x) = x * S * 8$

τ: Parámetro del protocolo con valor de 9 para la especificación actual del protocolo.

SYN: Intervalo de sincronización (0.01)

Para estimar el ancho de banda L, UDT envía un par de paquetes (omitiendo el tiempo de espera entre ellos) cada 16 paquetes de datos, en donde el receptor almacena el intervalo de tiempo de llegada de éstos y estima la capacidad del canal por S/T, en donde T es el tiempo medio entre llegadas de los paquetes y S el tamaño del paquete. Una vez realizada la estimación, se enviará al emisor mediante un ACK (Ver Figura 2).

Figura 2. ***Estimación del Ancho de Banda por parte de UDT.*** *UDT envía un par de paquetes (PP) cada 16 paquetes de datos desde el emisor al receptor y en donde éste con el tiempo de llegada entre ambos estima el ancho de banda y se lo informa al Emisor mediante un ACK [16].*

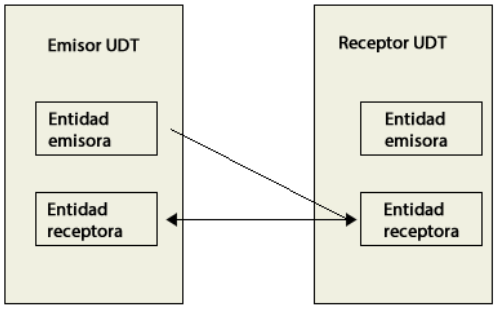


El algoritmo de control de congestión de UDT que se acaba de describir no se habilita hasta que el primer NACK es recibido o la ventana de flujo ha alcanzado su tamaño máximo. Este es un pequeño periodo de tiempo en donde se inicia el control de congestión y el periodo de tiempo entre paquetes se establece en cero, el tamaño

inicial de la ventana de flujo en 2 y éste se modifica según el número de paquetes reconocidos cada vez que un ACK es recibido.

Anteriormente se ha descrito el protocolo UDT y sus mecanismos internos, pero no se ha mencionado cómo trabajar con éstos. Para ello UDT ha implementado una librería en la que cada nodo cuenta con una entidad emisora y una receptora. Cuando se establece una conexión, el nodo emisor usa su entidad emisora para enviar paquetes de datos a la receptora del otro nodo y, ésta a su vez realiza un envío de paquetes de control a la entidad receptora del emisor (Ver Figura 3)

Figura 3. ***Relación entre el Emisor y Receptor UDT.*** *Todas las entidades UDT poseen la misma arquitectura, cada una tiene tanto una entidad emisora y una receptora. Esta figura demuestra la situación en donde un Emisor UDT envía datos a un Receptor UDT. Los datos son transferidos desde la Entidad emisora hasta la Entidad Receptora, mientras que el control de la información se intercambia entre las dos Entidades receptoras [16].*

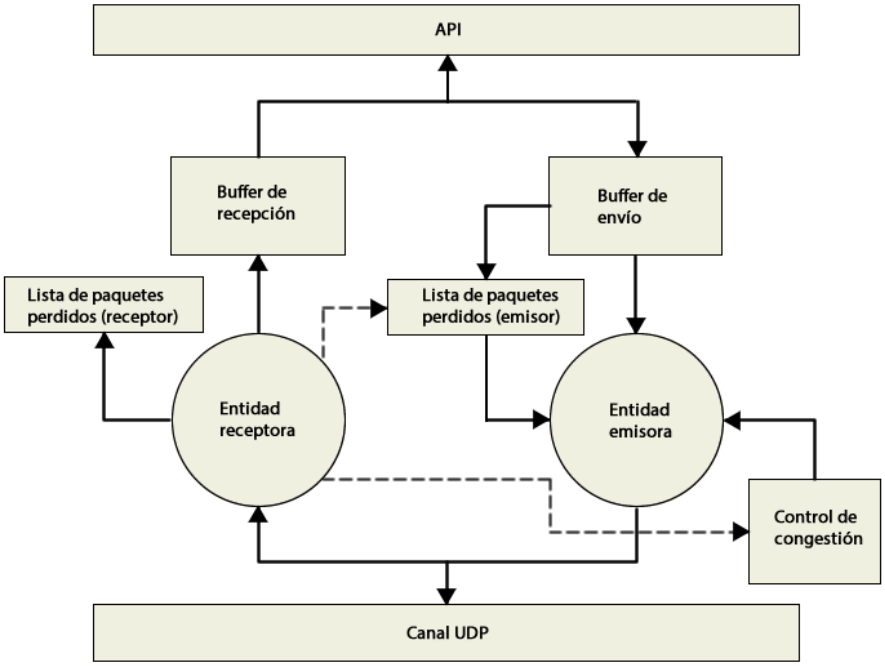


La librearía de UDT puede ser representada con una simple estructura (Ver Figura 4) en donde se cuenta con una primera capa encargada de interactuar con la aplicación (API), de la cual dependen los dos buffers: uno utilizado para la recepción y el otro para el envío de paquetes, los cuales permiten la comunicación con las entidades anteriormente descritas. La entidad emisora accede al buffer de envío para recolectar los datos y los transmite al canal de envío, sin antes consultar la información de envío proporcionada por el módulo de control de congestión. Por otra parte, la entidad emisora cuenta con una lista de paquetes perdidos que usará para saber cuáles debe retransmitir. La entidad receptora recoge los paquetes del canal de mensajes y los ubica en el buffer de recepción, si detecta la pérdida de un paquete los coloca en la lista de paquetes perdidos del receptor para después informar al emisor.

La característica más importante de la librería UDT es proporcionar una base para diseñar y probar nuevos algoritmos de control de congestión sin tener que modificar la implementación completa del protocolo. Esta característica se debe esencialmente a una configuración dinámica que proporciona una clase extensible de la librería denominada CCC, la cual permite definir el comportamiento de algunos

eventos que pueden producirse durante la transmisión. Por otra parte, la librería también permite monitorear el rendimiento de la transferencia de datos entre el cliente y el servidor, información valiosa a la hora de conocer la eficiencia de la transmisión o para definir el comportamiento de algoritmos de control que cambian de comportamiento a lo largo del tiempo.

Figura 4. ***Arquitectura software de la implementación de UDT.*** *La línea solida representa el flujo de datos y la línea punteada el flujo de control. Los buffers y lista de paquetes son los cuatro componentes de datos, mientras que los restantes son componentes de control [16].*



Por último, es conveniente anotar que la librería de UDT proporciona un excelente entorno de desarrollo de nuevos protocolos de comunicación ofreciendo una estructura orientada a la conexión, evitando tareas innecesarias como la gestión de las conexiones y sobre todo permitiendo trabajar a un nivel tan bajo como el protocolo UDP.

2.2. REPOSITORIOS DIGITALES Y SU USO PARA EL ALMACENAMIENTO DE GRANDES VOLUMENES DE DATOS

Un repositorio o repositorio digital es un conjunto de servicios de almacenamiento, gestión y diseminación de materiales digitales disponibles a los miembros de una determinada comunidad académica[17]. Desde una perspectiva más general, los

repositorios digitales fueron clasificados para incluir diferentes tipos de contenido y agrupados según éste:

- Repositorios Institucionales: Destinados a recompilar todo tipo de datos en una gran gama de formatos, los cuales pertenecen a una universidad o institución.
- Repositorios basados en una temática: Destinados a una única disciplina que generalmente abarca más de una institución y a menudo es de carácter internacional.
- Repositorios de formato: Destinados a recolectar un formato determinado y su alcance se encuentra bastante limitado. Entre éstos se encuentran repositorios de e-tesis, datos de investigación, imágenes digitales, etc.

Ahora bien, cuando se habla de repositorios institucionales, estos consisten en una aplicación web interoperable dedicada a preservar y difundir los recursos científicos y académicos de las instituciones o comunidades de investigación, todo esto a partir de un conjunto de datos específicos a cada recurso (metadatos). Según datos obtenidos de OpenDOAR[13], hay 2.973 repositorios en todo el mundo, en donde el 84.2% son institucionales y el restante son aplicados a diferentes contextos.

Por otro lado, se encuentran los repositorios basados en temáticas, que al igual a los institucionales consisten en una aplicación web y como ejemplo puede observarse iRODS[14], el cual archiva y preserva archivos multimedia relativos a diferentes áreas. En un entorno más científico, se han desarrollado proyectos como BXGrid[15], siendo éste un repositorio para la investigación biométrica y al mismo tiempo un sistema de gestión de datos que automatiza las copias de seguridad y la obtención de metadatos. Por último, hay sistemas como SDSS (*Sloan Digital Sky Survey*)[16], que al igual que *GRAND Data Lab*[17], están diseñados para exponer datos reales en un formato fácil de entender, tanto para investigadores aficionados como profesionales [2][18].

De forma particular, al diseñar repositorios de datos científicos para grandes experimentos, hay dos cuestiones a tenerse en cuenta. En la primera de ellas, el repositorio debe ser escalable; cualquier sistema que almacene Gigabytes o más datos día y que opere a largo plazo, llegará de forma rápida a un punto en donde la capacidad de almacenamiento de una única máquina no sea suficiente. Por lo tanto, un repositorio de datos científicos debe poder ser escalable a varias máquinas y además de ello, ser tolerante a fallos; al perderse un componente puede

[13] OpenDOAR: http://www.opendoar.org
[14] iRODS: https://www.irods.org/
[15] BXGrid: http://www3.nd.edu/~ccl/operations/bxgrid/
[16] SDSS: http://www.sdss.org/
[17] GRAND Data Lab: https://grandbackup.phys.nd.edu/

conservarse la integridad de los datos y preferiblemente sin interrumpir el acceso del usuario [2].

En segundo lugar, el sistema debe ser fácilmente accesible. En muchos casos, las colaboraciones científicas entre diferentes universidades e instituciones analizan datos de forma cooperativa y por tanto, los repositorios de datos deben contar con herramientas especializadas que permitan el acceso, la recuperación y el acceso a los datos de forma sencilla [2].

Es importante subrayar que los repositorios digitales y en particular aquellos de acceso abierto, cuentan con ciertas herramientas o protocolos utilizados para la diseminación de sus contenidos, entre los cuales está el protocolo de recolección de metadatos OAI-PMH (*Open Archives Initiative-Protocol for Metadata Harvesting*). Este protocolo es el estándar fundamental para asegurar la exposición, agregación, acceso e interoperabilidad de los contenidos depositados en los repositorios [19].

Otra iniciativa relevante para mejorar la calidad e interoperabilidad de los repositorios, son las directrices DRIVER (*Digital Repository Infrastructure Visión for European Research*). Éstas están dirigidas a los gestores de repositorios y sirven de guía de como exponer los recursos digitales usando el protocolo OAI-PMH junto con el formato de metadatos DublinCore[18]. DRIVER es además un repositorio que crea un punto único de acceso a la literatura científica europea y que se encarga de recolectar todo el contenido digital de los repositorios europeos, siempre y cuando se cumplan con las directrices. Actualmente DRIVER ha sido combinado con el proyecto OpenAire[19] que cumple las mismas funciones de éste.

En la siguiente sección se describe de forma detallada una de las iniciativas más importantes en los últimos años en cuanto a interoperabilidad de repositorios digitales; el proyecto SWORD.

Simple Web-service Offering Repository Deposit – SWORD

SWORD es un estándar de interoperabilidad que permite a los repositorios digitales aceptar depósitos de contenido en diferentes formatos desde múltiples fuentes vía un protocolo estandarizado. De la misma forma como el protocolo HTTP permite a los navegadores “hablar” con algún servidor web, SWORD permite a los clientes hacerlo con los repositorios. También tiene como finalidad reducir las barreras que

[18] DublinCore: El conjunto de elementos de metadatos Dublin Core es un vocabulario de quince propiedades para su uso en la descripción de recursos. El nombre "Dublín" es debido a su origen en un taller por invitación de 1995 en Dublin, Ohio; "Core" porque sus elementos son amplio y genérico, utilizable para describir una amplia gama de recursos.
[19] OpenAire: https://www.openaire.eu

existen para el depósito de documentos permitiendo la ingestión de archivos desde lugares remotos y por otra parte también permite obtenerlos. Entre otras características de SWORD, éste permite el depósito desde múltiples puntos y desde diferentes herramientas tales como Microsoft Office® e incluso permite el depósito múltiple a diferentes repositorios de forma paralela [20].

Originalmente fue una iniciativa financiada por JISC[20] y desarrollado inicialmente en el 2007 para hacer frente a la necesidad de una interfaz estándar de depósito en repositorios digitales [20]. Básicamente, es un perfil especializado del protocolo *Atom Publising*, pero se limita exclusivamente al ámbito de depositar archivos digitales en los repositorios [21].

La idea de tomar a *Atom Publising Protocol* como base, se debió a que en vez de desarrollar un nuevo estándar a partir de cero, se optó por aprovechar un protocolo de nivel de aplicación para la publicación y edición de recursos web [22]. SWORD se centró en dos aspectos claves: el depósito de archivos en lugar de documentos y el mecanismo de extensión para especificar los parámetros de depósito adicionales. El resto de la especificación del protocolo se deja libre para posteriores implementaciones que se deseen hacer como agregado a SWORD pero que no son de competencia para el proyecto de JISC [21].

Posteriormente se desarrolló e implementó la segunda versión de SWORD en el 2011 y su objetivo fue ampliar el soporte dado en todo el ciclo de vida del proceso de depósito. Esto se logra proporcionando mecanismos para no sólo depositar recursos, sino también para actualizarlos, reemplazarlos y eliminarlos [23]. Sin estas características adicionales, muchos de los siguientes casos de uso no serían posibles:

- Caso de uso 1: Editor al repositorio.
- Caso de uso 2: Sistema de información de la investigación al repositorio.
- Caso de uso 3: Escritorio al repositorio.
- Caso de uso 4: Repositorio a repositorio.
- Caso de uso 5: Interface de usuario especializada en depósito al repositorio.
- Caso de uso 6: Sistema de envíos a Conferencia al repositorio.
- Caso de uso 7: Equipos de laboratorio al repositorio.
- Caso de uso 8: Ingestión a granel en el repositorio.
- Caso de uso 9: Autoría colaborativa.

Entre los casos de uso, uno de los más prometedores es la ingestión de datos desde detectores/sensores. Muchos instrumentos actuales cuentan con interfaces que permiten capturar de forma automática los resultados en un sistema de información

[20] JICS: https://www.jisc.ac.uk/

y esta característica puede ser utilizada por SWORD para cargar directamente los datos en un repositorio sin la intervención humana. Una de las grandes ventajas de recolectar los datos mediante este método, es poder registrar las condiciones en que se han generado los datos en archivos de metadatos específicos a cada uno [24].

Entre los software que son compatibles con este estándar se encuentran principalmente los repositorios arXiv, DSpace, Eprints, Fedora y Microsoft Zentity. Además de esto, SWORD proporciona librerías para desarrollar nuevos clientes basados en PHP, Java y Python.

2.3. DISPOSITIVOS IMPLEMENTADOS EN LA GENERACIÓN Y TRANSFERENCIA DE DATOS

Los nuevos desarrollos tecnológicos se han incrementado, en particular a lo referente a los circuitos integrados, gracias a los avances de la miniaturización del hardware y en las herramientas de diseño. Dispositivos como las ASIC (*Application-Specific Integrated Circuits*), proveen recursos altamente optimizados para la realización de tareas críticas a alta velocidad como lo son la adquisición de datos (DAQ – *Data Adquisition*[21]) al igual que para poder realizar tareas de forma paralela.

En lo referente a aplicaciones paralelas, se han desarrollado máquinas que implementan ASIC, como se ha hecho en MDGRAPE-3; una supercomputadora desarrollada por RIKEN[22] y destinada a realizar simulaciones de dinámica molecular y de predicción de la estructura de las proteínas con la capacidad de alcanzar velocidades del orden del Petaflop. A pesar de este ejemplo, la utilización de las ASIC se ha tornado inviable para la mayoría de las aplicaciones paralelas por el gran costo inicial y la tendencia a ser superadas por las computadoras de propósito general en una o dos décadas de desarrollo de chips, tal y como se describe en la Ley de Moore.

Las ASIC modernas a menudo incluyen procesadores de 32 bits, bloques de memoria RAM, ROM, EEPROM y Flash, así como diferentes tipos de módulos y frecuentemente a éstas se le son llamados “Sistemas en Chip” o SoC (*System on a chip*) por sus siglas en inglés. Estas ASIC son las implementadas para el desarrollo de lo que actualmente se conoce como SBC (*Single-board Computer*) como por

[21] DAQ: http://www.ni.com/data-acquisition/what-is/esa/
[22] RIKEN: http://www.riken.jp/en/

ejemplo la Raspberry Pi[23], Cubieboard[24] o la tarjeta desarrollada por NVIDA, la Jetson TK1[25].

Por otra parte, las ASIC involucran ingeniería No-Recurrente (NRE por sus siglas en inglés); costo de investigación, desarrollo, diseño y pruebas. Ésta puede llegar a tener valores muy altos y que junto a los ya altos costos de producción hacen que estas sean inviables para diseños pequeños o con volúmenes de producción bajos. Estas desventajas han hecho que las FPGAs (*Field Programmable Gate Array*) tomen mayor fuerza en aplicaciones en donde sea necesaria una alta flexibilidad en el diseño de circuitos. Sin embargo, las FPGAs son más lentas, poseen un mayor uso de potencia y no pueden abarcar sistemas muy complejos, pero pueden utilizarse como DAQ.

Como se mencionaba anteriormente, las ASIC y las FPGA son implementadas para la adquisición de datos y, por lo tanto, gran parte de los grandes instrumentos a nivel mundial basan sus detectores y sistemas DAQ en estos. Al mismo tiempo han sido usados para la transferencia de datos mediante la generación de paquetes TCP, los cuales son enviados a través de la red hasta almacenes de datos o repositorios digitales de datos científicos [25]. Por otra parte, también han sido usados para la interconexión de los detectores con computadoras de propósito general o a SBC [26][27].

Unos ejemplos de estos experimentos son el KLOE-2 [7], LAGO [28] y el SuperKamiokonde[26]; un detector Cherenkov[27] de agua de 50 Kilotoneladas y conocido en el mundo de la física de partículas como el detector de neutrinos más sensible que se ha construido hasta la fecha. El sistema actual de adquisición de datos (DAQ) lee todas las salidas de los fotomultiplicadores[28] que superen el umbral del trigger y las almacena en el archivo de datos. Este sistema DAQ está diseñado e implementado en FPGAs, incluso aquellos diseñados para la detección de eventos como supernovas [6].

A continuación se define de forma más detallada el concepto de FPGA y SBC, así como los diferentes dispositivos implementados durante la realización del proyecto.

[23] Raspberry Pi: https://www.raspberrypi.org/
[24] Cubieboard: http://cubieboard.org/
[25] NVIDIA Jetson TK1: http://www.nvidia.com/object/jetson-embedded-systems.html
[26] Super-Kamiokonde: http://www-sk.icrr.u-tokyo.ac.jp/sk/
[27] Detector Cherenkov: Tiene como finalidad registrar la caída de rayos cósmicos y se basa en el efecto Cherenkov. Este efecto se produce cuando una partícula cargada se mueve en un medio con una velocidad mayor que la de la luz en dicho medio. Esto producirá una perturbación electromagnética que origina una emisión de luz que puede ser detectada por un fotomultiplicador.
[28] Fotomultiplicador: Es un dispositivo que permite detectar luz con alta sensibilidad o en otras palabras transforma impulsos luminosos débiles en una corriente eléctrica amplificada.

2.3.1. Field Programmable Gate Array – FPGA

Las FPGA son dispositivos hardware que poseen dos características que les dan una gran ventaja sobre otros: una gran velocidad y eficiencia energética así como el hecho de poder ser reprogramadas. Con respecto a este último, los algoritmos pueden ser programados en el chip en cualquier momento, a diferencia de las ASIC que son grabados de forma permanente durante el proceso de fabricación [29].

La reprogramación proporciona un medio para añadir nuevas funcionalidades a diseños previos o para programar diferentes sectores de la FPGA, con lo cual se logran la realización de tareas independientes y de forma simultánea [30]. Por lo general, las FPGAs son útiles sólo para operaciones que procesan grandes flujos de datos, tales como el procesamiento de señales, redes y tareas similares.

2.3.2. Single-board Computer – SBC

Una SBC es una computadora construida en una sola placa de circuito, con microprocesadores, memoria, puertos de entrada/salida (E/S) y otras características requeridas en un equipo funcional. Originalmente fueron diseñadas y construidas como computadoras embebidas para el control de sistemas, pero posteriormente fueron modificadas e implementadas en diferentes sistemas para la enseñanza de electrónica y programación de sistemas [31]. Se caracterizan por ofrecer dimensiones muy reducidas y con una gran variedad de conectores externos necesarios para implementar un sistema embebido completo.

Las SBC son utilizadas comúnmente en situaciones industriales, en los procesos de control o embebidos dentro de otros dispositivos para proveer una interfaz con el usuario. Debido a los altos niveles de integración, las SBC son más livianas, pequeñas y energéticamente eficientes que las placas de los actuales equipos informáticos [31].

Entre las SBC más destacadas se encuentra la Raspberry Pi[29] y aunque fue diseñada principalmente para ser un recurso de enseñanza en las aulas de clase e interesar a los estudiantes por las ciencias de la computación, ésta SBC posee una sencillez de conexión para una gran diversidad de sensores, lo cual le permite su uso en otro tipo de proyectos como lo son centros multimedia o robótica aplicada.

Otra SBC es la Cubieboard[30] que, al igual a la Raspberry Pi, ejecuta un sistema Linux, pero a diferencia de ésta, cuenta con muchos más puertos y conectores. Al

[29] Raspberry Pi: https://www.raspberrypi.org/
[30] Cubieboard: http://cubieboard.org/

comparar el hardware de estas dos placas, la Cubieboard presenta mejores características, lo cual permite ser implementada en proyectos de BigData mediante el uso de Hadoop[31] o su uso en clusters computacionales.

NVIDIA[32] por su parte ha desarrollado su propia SBC a la cual se le ha agregado la posibilidad de desarrollar aplicaciones integradas que usen GPUs. La Jetson TK1[33] ha sido diseñada sobre la base del SoC NVIDIA® Tegra® K1 proporcionándole la misma arquitectura de cálculo que se emplea en numerosos supercomputadores. Esta SBC permite desarrollar proyectos en campos como la visión artificial, la robótica, entre otros.

[31] Hadoop: https://hadoop.apache.org/
[32] NVIDIA: http://www.nvidia.com
[33] JTK1: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

3. MODELOS DE TRANSFERENCIA DE DATOS

En este capítulo se presenta los dos modelos cliente/servidor desarrollados. En la primera parte se muestra el modelo basado en el estándar de interoperabilidad SWORD y sus diferentes características. En la segunda parte se hablará del modelo cliente/servidor construido utilizando el protocolo UDT y su respectiva librería de desarrollo.

3.1. MODELO CLIENTE/SERVIDOR BASADO EN SWORD

Antes de iniciar a describir el modelo desarrollado, se mostrará el funcionamiento del estándar de interoperabilidad SWORD con el fin de aclarar el por qué no es funcional para ciertos alcances del proyecto y el por qué se construye otro modelo cliente/servidor basado en el protocolo UDT.

3.1.1. Funcionamiento del estándar SWORD

El estándar de interoperabilidad SWORD es por sí mismo solo una definición de una serie de procesos o pasos a seguir de cómo realizar desde la comunicación hasta la trasferencia de datos y metadatos. Por consiguiente, se han creado varias librerías para simplificar la programación de nuevos clientes, entre las cuales se encuentran las librerías para Java, Python, PHP y Ruby. Además, para desarrollos más avanzados, se proporcionan librerías para el desarrollo de servidores SWORD en Python y Java.

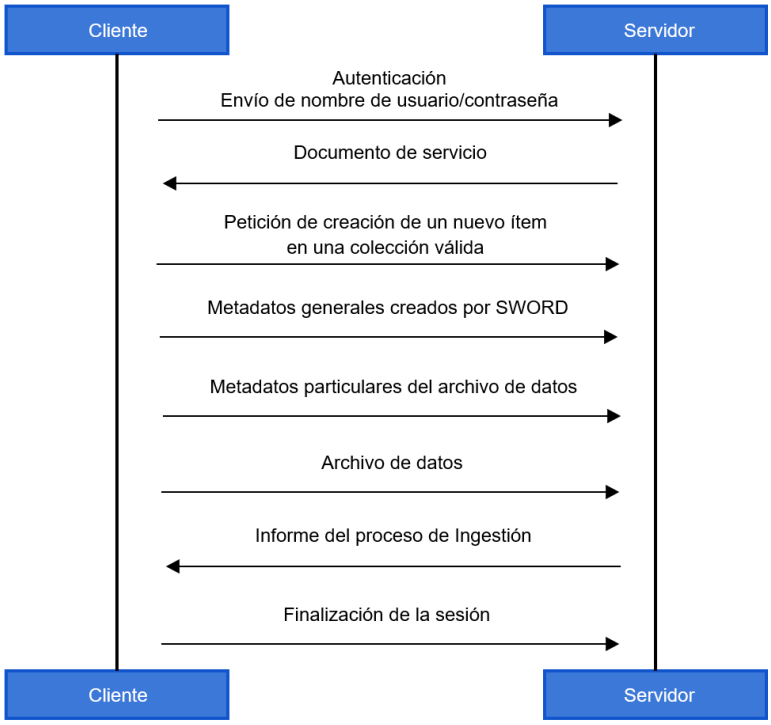
SWORD presenta dos interacciones principales. Durante la primera interacción con el servidor, se validan los datos de conexión (nombre de usuario y contraseña) y como respuesta, el servidor devuelve el documento de servicio, en donde se contiene información detallada del repositorio, de las comunidades y colecciones existentes. En la segunda interacción, se realizan los procesos de transferencia de datos desde el cliente hacia el repositorio. En relación con esta interacción, se presentan una serie de tareas que pueden ejecutarse, tales como el envío de metadatos o la consulta de las colecciones disponibles o válidas para el usuario ya identificado.

En cuanto a la transferencia de datos, el repositorio inicia el proceso de ingestión con la creación de una serie de metadatos generales por parte de SWORD: la fecha de envío, un nombre provisional y el nombre del autor. Cabe hacer notar que

durante esta etapa pueden enviarse nuevos metadatos o modificar los ya existentes. Una vez terminada esta tarea, se procede con la transferencia desde el cliente y el servidor culmina con el proceso de ingestión. Finalmente, el servidor envía un mensaje al cliente indicándole que todo el proceso se ha llevado a cabo de forma correcta y la ingestión ha sido exitosa. Si se llegará a presentar algún problema, el servidor retorna un mensaje de error al cliente indicando de forma detallada en qué punto del proceso se produjo el fallo.

En la Figura 5 puede observarse una adaptación del proceso descrito anteriormente para la transferencia de archivos que se ha propuesto en este trabajo.

Figura 5. ***Modelo Cliente/Servidor basado en SWORD.*** *Después de la autenticación en el repositorio, se envía el documento de servicio desde el Servidor al Cliente, con el cual puede realizarse envío al repositorio mediante la creación de un nuevo ítem, la generación de metadatos y envío final de los datos. Al finalizar la tarea de ingestión el Servidor informa al Cliente que el proceso se ha llevado con éxito y termina la sesión.*



Una de las principales características de un servidor SWORD es el hecho de permitir conexiones múltiples, con lo cual una gran cantidad de clientes pueden realizar ingestiones de forma concurrente. Por otra parte, los clientes SWORD, incluso el desarrollado durante este trabajo, no cuentan con la capacidad de transferir múltiples archivos de forma simultánea al repositorio. Ahora bien, para lograr agrupar y almacenarlos, éstos deben ser empaquetados en un único archivo antes de la realización del proceso de transferencia e ingestión realizado por el cliente.

Por otro lado, la descripción del estándar de interoperabilidad SWORD menciona la posibilidad de implementar otros esquemas de metadatos diferentes a DublinCore, pero las librerías desarrolladas basadas en este estándar no admiten el uso de otros esquemas. En otras palabras, archivos como por ejemplo datos de investigación, no pueden ser descritos por completo debido al hecho de que el estándar DublinCore fue diseñado únicamente para documentos. Al mismo tiempo, muchas otras capacidades de SWORD descritas en la definición del estándar no se encuentran presentes en estas librerías, obstaculizando la creación de nuevos clientes con una funcionalidad completa.

3.1.2. Descripción del modelo cliente/servidor

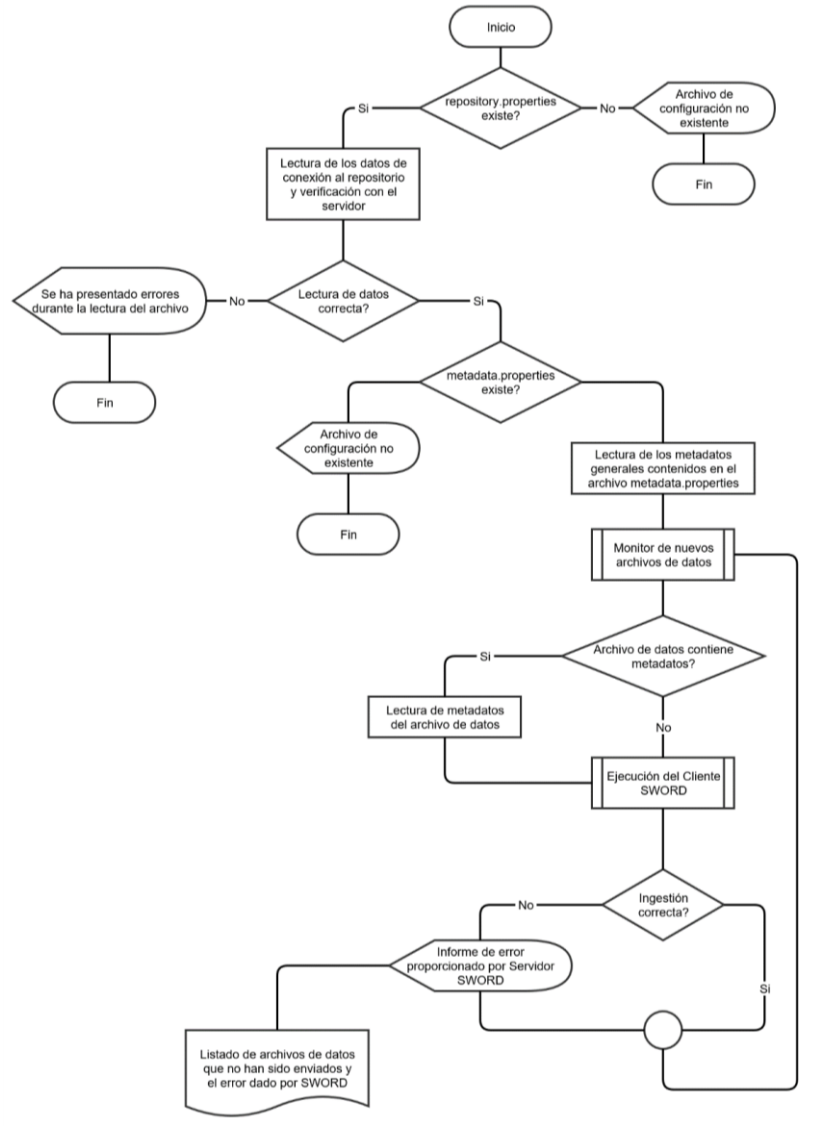
En el capítulo anterior se han descrito el uso de repositorios para el almacenamiento de datos de investigación y la utilización de dispositivos hardware tales como FPGA, SBC basados en SoC, tanto para ser usados como DAQ como para la transferencia de datos a través de la red. Del mismo modo, se ha hablado sobre las características del estándar de interoperabilidad SWORD y sus casos de uso. En esta sección se abordará el caso de uso en donde SWORD es implementado en equipos de laboratorio, más exactamente, en detectores soportados por una SBC.

Antes de entrar en detalle sobre el modelo cliente/servidor, se enumeran una serie de consideraciones y requerimientos importantes en las cuales se basa el modelo y que hacen énfasis en la utilización de las herramientas software y hardware descritas a los largo del presente trabajo:

- Los detectores cuentan con un sistema de adquisición de datos y además de una unidad que tenga un procesador que sea capaz de ejecutar un sistema operativo.
- Los detectores poseen un puerto Ethernet de al menos 10/100 Mbit/s y se encuentra conectado a una red.
- Los detectores deben tener la capacidad de realizar de forma automática la ingestión de los datos producidos por su sistema de adquisición.
- Cada detector tiene un archivo que lo describe, siendo éste utilizado para la generación de los metadatos generales.
- Cada archivo de datos generado por el sistema de adquisición contiene una serie de metadatos en su cabecera, los cuales serán tomados y transmitidos al repositorio en el momento de realizar la ingestión.
- Los datos a transferir son almacenados temporalmente en el detector.
- El servidor SWORD está enlazado a un repositorio de datos tal como DSpace o Eprints.

Sobre la base de las ideas anteriormente expuestas se construyó el modelo considerando las limitaciones dadas por las librerías de desarrollo de SWORD, tales como el uso exclusivo del esquema de metadatos DublinCore y la restricción de sólo poder transferir un archivo por ítem a ingerir en el repositorio. En la Figura 6 se presenta el esquema general del cliente SWORD desarrollado para este trabajo.

Figura 6. ***Esquema general del cliente SWORD.*** *Inicialmente el cliente comprueba la existencia del archivo repository.properties de donde tomará la información de conexión al repositorio. Seguido se comprueba la existencia del archivo metadata.properties y se leen los metadatos generales que caracterizan al dispositivo hardware que funciona como detector. Una vez comprobado la existencia de los archivos iniciales se activa el monitor de nuevos datos y al encontrarse uno nuevo, se lee la cabecera de metadatos, si ésta existe, activando acto seguido el cliente SWORD encargado de la ingestión al repositorio. Finalmente, el servidor transmite un mensaje de éxito o fracaso de la ingestión, en caso de este último se almacena en una lista el nombre y ruta del archivo que no pudo ser ingerido y posteriormente se retoma nuevamente el monitoreo de nuevos archivos de datos.*



Esencialmente, el cliente se inicia con la lectura de dos archivos de configuración que le permitirán establecer la parte inicial del proceso de conexión con el servidor SWORD y el monitoreo de nuevos datos. El primero de ellos, *repository.properties*, contiene la información necesaria para ejecutar el sistema de monitoreo, la adquisición del documento de servicio desde el servidor SWORD, así como el tamaño del encabezado que contiene los metadatos del archivo de datos. El segundo archivo, *metadata.properties*, es un archivo que contiene los ya antes mencionados metadatos generales que serán transmitidos para la creación del ítem en el repositorio; si estos no son proporcionados, SWORD los creará de forma automática. En cuanto al archivo de metadatos, éste contiene una segunda sección en la cual se encuentran los nombres de los metadatos ubicados en la cabecera del archivo de datos; esto es usado para relacionarlos con el esquema DublinCore;

Retomando el esquema del cliente, una vez leído el archivo *repository.properties*, se realiza la primera interacción con el servidor SWORD y se obtiene el documento de servicio, tal como se muestra en la Figura 5. Puede llegar a producirse un error de autenticación de tal manera que el servidor retornará un error de conexión y el cliente no podrá ser iniciado. Ahora bien, una vez llevado a cabo las tareas anteriores y establecida de forma correcta la conexión, el servicio de monitoreo es inicializado y queda a la espera de la generación de nuevos archivos de datos por parte del sistema de adquisición.

Finalmente y una vez se detecte un nuevo archivo de datos, se recorre la cabecera del archivo en búsqueda de sus metadatos y éstos son relacionados con el esquema DubliCore. Tanto si existe como no esta cabecera, se procede a realizar el llamado al servicio de ingestión y con ello a las demás tareas expuestas en la Figura 6. Cabe aclarar que los metadatos ya leídos de los archivos de configuración iniciales son mantenidos en memoria durante toda la ejecución y no podrán ser modificados hasta que se reescriban los archivos y se reinicie el cliente, mientras que los metadatos contenidos en la cabecera de cada archivo de datos son escritos cada vez que sea generado uno nuevo.

Otra tarea prioritaria dentro del cliente es mantener un informe de aquellos archivos que no pudieron ser transmitidos y el motivo por el cual la ingestión fue rechazada. Esta situación en muchos casos se debe a problemas de conexión entre el cliente y el servidor o, por la caída del servicio durante el proceso de ingestión.

Una vez finalizado el proceso con o sin éxito, el cliente retorna nuevamente a monitorear la generación de nuevos archivos de datos por parte del sistema de adquisición. Como resultado exitoso del proceso de ingestión, los datos estarán accesibles en el repositorio permitiendo realizar búsquedas indexadas de los datos a través de sus metadatos.

En relación al servidor SWORD, no es necesario realizar ningún otro procedimiento más que el proceso de instalación recomendado por la empresa desarrolladora del software elegido para el repositorio digital. Esto gracias a que SWORD se ha convertido en un estándar en este tipo de herramientas. Para el caso de este trabajo se realizó la configuración utilizando DSpace y se validó que el puerto 8080 sea accesible y de manera más particular que pueda solicitarse el documento de servicio al repositorio.

3.2. MODELO CLIENTE/SERVIDOR BASADO EN UDT

En la sección anterior se presentó el modelo cliente/servidor basado en SWORD y se mostró la problemática presente en cuanto al uso de esquemas de metadatos diferentes a DublinCore durante el desarrollo del cliente. En secciones posteriores se discutirá otras fallas presentes en las librerías del estándar de interoperabilidad, tales como la restricción en el tamaño del archivo a transferir, pero por el momento se toma la problemática presente en el uso de otros esquemas de metadatos como punto de partida para iniciar la búsqueda de alternativas de transferencia e ingestiones masivas hacia repositorios digitales.

En esta sección se hablará del diseño de un nuevo modelo de transmisión de datos basado en UDT, en donde se ha creado tanto el cliente como el servidor para la realización de la transferencia y la ingestión de los datos a un repositorio digital. Antes de describir este modelo se mostrará la herramienta software utilizada para la puesta en marcha del repositorio de datos y se hablará de las características más importantes de éste.

3.2.1. Software para repositorios DSpace

DSpace es un software de código abierto para la puesta en marcha de repositorios digitales que tiene la capacidad de capturar, almacenar, ordenar, preservar y distribuir material digital con el fin de garantizar la preservación y distribución de toda la producción científica en las instituciones que hacen uso de éste. Fue creado por el conjunto de bibliotecas del MIT y tenía como objetivo inicial acoger las más de cien mil unidades de contenido digital producidas cada año por sus profesores e investigadores. Fue desarrollado utilizando las normas y estándares existentes, lo que le permite integrarse a otros repositorios o sistemas de información.

Una de las características más importantes de DSpace es su sistema de indexación y búsqueda sobre metadatos y texto completo. Además de esto, DSpace crea url´s

permanentes para cada envío realizado y permite la realización de copias de seguridad de forma automática.

A continuación se presentan las características generales de DSpace:

- Los autores utilizan una interfaz web para el depósito de archivos.
- Los metadatos son almacenados junto a los archivos para apoyar la preservación.
- Los documentos se organizan en:
    - Comunidades
    - Sub-comunidades

  Éstas corresponden a partes de la organización tales como departamentos, laboratorios y escuelas.
- Dentro de las comunidades o sub-comunidades se crean las colecciones que pueden corresponder a las diferentes investigaciones realizadas por un departamento o laboratorio.
- Las colecciones están compuestas por ítems los cuales pueden contener uno o varios archivos digitales que dentro del marco de DSpace se denominan bitstreams.
- Proporciona herramientas administrativas a través de la consola de comandos que permiten importar, exportar, actualizar y realizar una gran cantidad de tareas de mantenimiento sobre el repositorio.

En relación con esta última característica, DSpace cuenta con el comando *import* que permite realizar ingestiones de datos siempre y cuando se cumpla con la estructura *Simple Archive Format (SAF)*[34]. Como puede observarse en la Figura 7, SAF es una simple estructura de directorios en donde por cada ítem a ingerir se crean dos archivos esenciales (dublin_core.xml y contents), uno opcional (metadata_[prefix].xml) que es usado para esquemas de metadatos diferentes al DublinCore y el resto de archivos que se muestran son los archivos a depositar.

[34] SAF: https://wiki.duraspace.org/display/DSDOC18/Importing+and+Exporting+Items+via+Simple+Archive+Format

*Figura 7. **Simple Archive Format**. Estructura de archivos necesaria para llevar acabo la ingestión de datos mediante el comando import. Se compone esencialmente de una carpeta por cada ítem a ingerir, en donde cada uno contiene los archivos a ingerir, el esquema de metadatos DublinCore en formato xml y un archivo llamado contents que lista los nombres de los archivos a ingerir. Existe otro archivo xml para describir el ítem usando otros esquemas de metadatos denominado metadata_[prefix].xml en donde [prefix] es el nombre del esquema de metadatos registrado en el repositorio.*

```
archive_directory/
    item_000/
        dublin_core.xml
        metadata_[prefix].xml
        contents
        file_1.doc
        file_2.pdf
    item_001/
        dublin_core.xml
        contents
        file_1.png
        ...
```

En resumen, la herramienta *import* junto con la estructura SAF son esenciales para la creación del nuevo modelo cliente/servidor y que junto al protocolo UDT permitirá la transferencia masiva de datos y la creación del sistema de ingestión automático.

3.2.2. Descripción del modelo cliente/servidor

En esta sección se describe el modelo cliente/servidor dado como alternativa al modelo presentado para la ingestión de datos mediante el estándar de interoperabilidad SWORD. Para este modelo se han diseñado y programado cada uno de los métodos necesarios para el establecimiento de la conexión, la interacción del cliente con el servidor y viceversa, así como los métodos para la obtención y creación de metadatos, de la estructura SAF y la ejecución de las ingestiones en el repositorio implementando la herramienta *import*

El nuevo modelo cliente/servidor se basó en SWORD asimilando su forma de interactuar con el repositorio durante todo el proceso de conexión, transmisión y cierre de la conexión. Ahora bien, SWORD usa canales de transmisión basados en el protocolo TCP mientras que el modelo planteado en esta sección se basa esencialmente en la utilización del protocolo UDP mediante la implementación de la librería UDT y de la utilización de su algoritmo predefinido de control de congestión.

En la Figura 8 se presenta el modelo cliente/servidor construido como resultado a las consideraciones anteriores. Como puede apreciarse se ha diseñado y construido tanto el cliente como el servidor y todos los métodos encargados de la interacción

entre ellos. Cabe recordar que en el modelo para SWORD, sólo se creó el cliente y éste está restringido a las interacciones dadas por el servidor que viene por defecto en las herramientas software para repositorios.

Conviene destacar que el nodo que inicia la conexión con el repositorio tiene funciones de cliente y de servidor dependiendo de la tarea que se esté realizando, de la misma forma el nodo que contiene al repositorio cambia su función de servidor a cliente cuando se realiza la transferencia del archivo. Con esta flexibilidad se permite que el encargado de la recolección de datos sea el nodo que contiene el repositorio, otorgándole la facultad de requerir cuantas veces sea necesario el archivo de datos a los clientes hasta que la transferencia sea completada. Esto se hace con el fin de que en caso de una caída del nodo cliente o perdida de conexión, el nodo del repositorio puede seguir requiriendo el archivo de datos hasta que pueda ser transferido e ingerido en el repositorio.

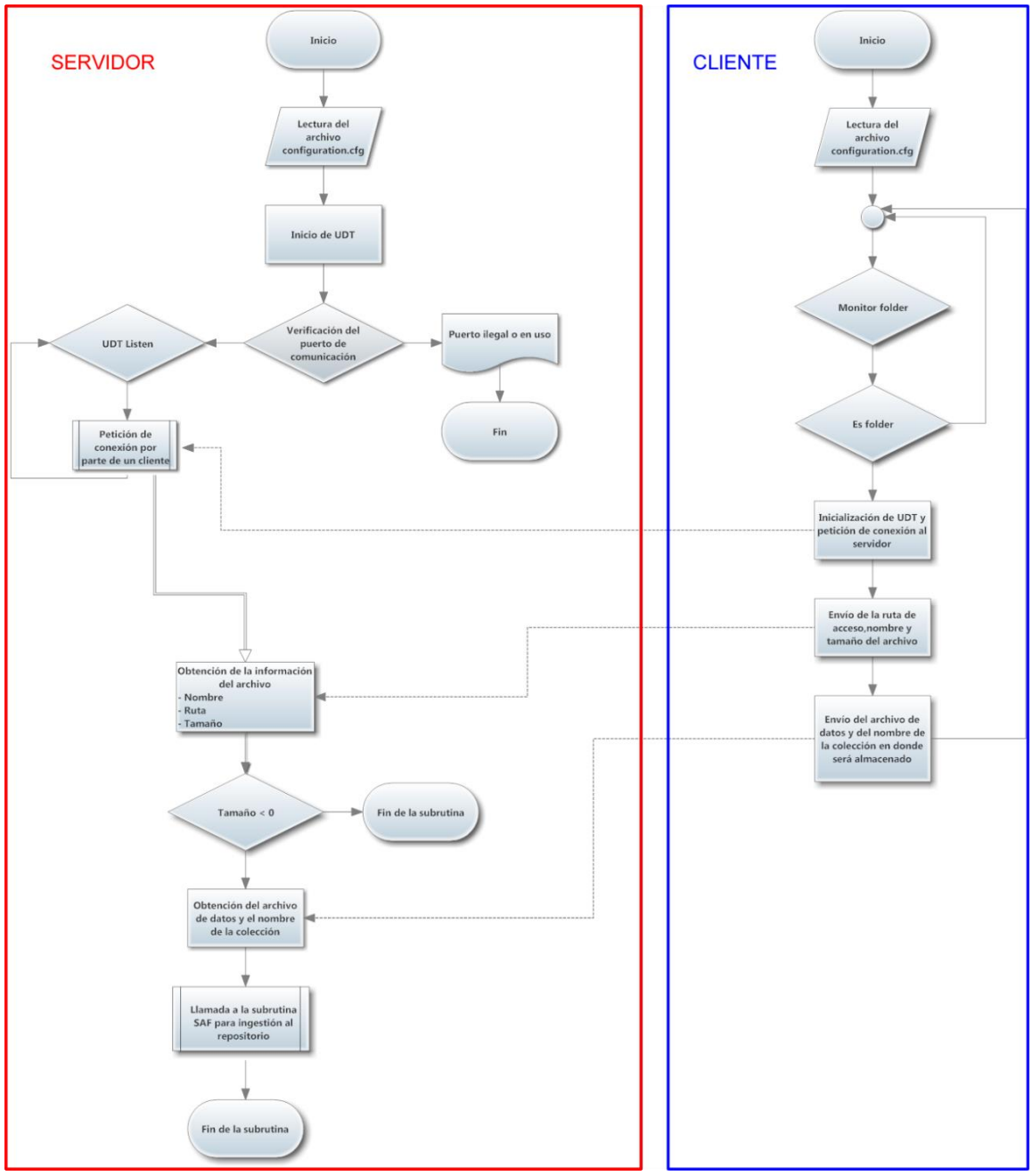
En cuanto al esquema presente en la Figura 8, al lado izquierdo se observa el modelo del servidor, ubicado en el nodo donde está alojado el repositorio y al lado derecho el modelo del cliente que se ejecutará en el hardware del detector. Nos permitimos poner en relieve que los detectores que implementan una SBC no poseen muchos recursos computacionales y a diferencia del cliente SWORD, este cliente es mucho más simplificado y tareas como la obtención y creación de la estructura de metadatos, ingestión y las tareas de verificación de transferencia fueron trasladadas al servidor, quitándole de esta forma carga computacional a los detectores dejando el trabajo “pesado” al servidor. Motivo por el cual se asume que el nodo en donde está alojado el repositorio es una máquina con los recursos suficientes para soportar todos los procesos de ingestión (creación de la estructura SAF y ejecución del proceso de ingestión mediante la herramienta *import*), los cuales son requeridos cada vez que un detector solicite transferir un nuevo archivo de datos.

Esencialmente, el cliente inicia su función con la lectura del archivo de configuración *configuratión.cfg*, el cual contiene la información suficiente para establecer la conexión con el repositorio y además contiene la ubicación de la carpeta en donde se generan los datos por parte del sistema de adquisición; se realiza la verificación de su existencia para acto seguido activar la función de monitoreo de nuevos datos.

Al ser detectado un nuevo archivo de datos, el cliente inicializará UDT y realizará una petición de conexión al servidor. Posterior al establecimiento del enlace, el cliente enviará la ubicación del archivo, su nombre y el tamaño en bytes. Como se mencionó anteriormente, el cliente cambia su función a servidor de datos, esperando a que el nodo del repositorio, ya con la información de la existencia de un nuevo archivo de datos, solicite el archivo. Como se menciona anteriormente este cambio de rol se realiza para asignarle la tarea de recolector de datos al nodo

del repositorio y como método para asegurar la transferencia e ingestión de datos al repositorio.

Figura 8. ***Esquema Cliente/servidor basado en UDT.*** *Del lado izquierdo se observa la estructura Servidor y del derecho la del cliente. Ambos algoritmos verifican la existencia de sus archivos de configuración para posteriormente iniciar sus respectivos servicios. Del lado del cliente se inicia el monitoreo de nuevos archivos de datos y por parte del servidor éste queda a la espera de alguna petición de conexión. Una vez confirmado por parte del cliente, la creación de un nuevo archivo de datos, se inicia la interacción entre los dos nodos. El cliente realiza una petición de conexión y una vez establecida, el flujo de datos y metadatos entre el cliente y el servidor se lleva a cabo. Una vez finalizada, el servidor ejecuta la rutina de ingestión creando de esta manera la estructura SAF y ejecutando el comando import.*



Ahora bien, cuando se describió el modelo del protocolo basado en SWORD se mencionaba un archivo de metadatos, el cual contenía todos aquellos relacionadas con la configuración del detector y, también se hacía referencia a una cabecera por archivo de datos, el cual contiene los metadatos que lo describen. Para este nuevo modelo, estos metadatos se mantienen en el archivo *metadata.properties*, y en el archivo *configuration.properties* se da el número de líneas de la cabecera en el archivo de datos. Es importante subrayar que estos metadatos son enviados seguido al envió del archivo de datos.

Tras el envío de los datos y los metadatos por parte del detector, el servidor inicia la creación de la estructura SAF dentro de un directorio temporal. Una vez terminada la estructura, el servidor ejecuta el comando *import* mencionado anteriormente. Cabe aclarar que este comando tiene varios parámetros que deben ser proporcionados por el cliente durante el inicio de la transmisión. Tales parámetros incluyen tanto el nombre de la colección de datos, el nombre de usuario quien desean realizar la ingestión, así como la ruta temporal de la estructura SAF. Existe otro parámetro dentro de este comando denominado *mapfile*, el cual es un archivo de salida que contiene la parte final de la url permanente que crea DSpace al ingerir un nuevo ítem. A este respecto, este archivo tiene la función de permitir al usuario modificar y borrar el archivo ingerido en el repositorio.

4. IMPLEMENTACIÓN DE LOS MODELOS DE TRANSFERENCIA DE DATOS

En el capítulo anterior se presentaron los dos esquemas de las estructuras de modelos de transferencia de datos basados en SWORD y en UDT. En donde para SWORD se presenta la estructura del cliente y para UDT un nuevo modelo cliente/servidor. En este capítulo se presenta como estos fueron implementados en SBC y la problemática presente a lo largo de esta tarea.

4.1. IMPLEMENTACIÓN DEL CLIENTE SWORD EN LAS SBC

A lo largo de esta sección se hablará sobre la técnica de implementación del cliente SWORD sobre los dispositivos SBC y de las tareas inherentes en este proceso.

Sobre las ideas expuestas y el diseño propuesto para el cliente SWORD en la sección 3.1 y teniendo presente que las SBC son dispositivos basados en arquitecturas SoC con procesadores ARM que pueden soportar un sistema operativo Linux, se decide desarrollar el cliente en Python después de realizar pruebas a esta librería y a la de Java. Con respecto a esta última, se encontraron problemas con la transferencia de archivos de datos de tamaño superiores a los 10 MB y por lo cual se descarta ésta para el desarrollo del nuevo cliente debido a no cumplir con el requisito de poder transferir archivos de gran tamaño. Esta restricción se presenta principalmente por el manejo del buffer de datos dado por ésta librería en donde se logra saturar fácilmente.

Una vez desarrollado el cliente en Python y de realizadas las pruebas de transferencia e ingestión desde una PC al repositorio, se obtienen los requerimientos necesarios para el funcionamiento de éste en una SBC. Los cuales se presentan a continuación:

:

1. Instalación de Python 2.7 sobre el sistema Linux para ARM. Dependiendo de la distribución implementada puede realizarse de diferentes formas, pero por facilidad, se aconseja realizarla mediante el gestor de paquetes de la distribución utilizada.
2. Creación de los archivos .properties. Sin estos archivos el cliente no puede iniciarse y además, ellos contienen toda la información necesaria para establecer la conexión al repositorio y la ingestión de los datos.

3. Creación de la carpeta (si no existe) en donde serán almacenados los archivos de datos a transferir. La ruta de este directorio debe estar en el archivo repository.properties.
4. Inicialización del cliente. Esta tarea puede realizarse mediante la simple ejecución del script Python, la creación de un servicio Linux o mediante una tarea cron. Para el caso de este trabajo se decide inicializar el cliente mediante la creación de un servicio que se inicia durante el arranque del sistema operativo de la SBC.

Finalmente tras concluir estas tareas, el dispositivo queda en la capacidad de realizar ingestiones de archivos de datos a un repositorio a través de la red y esto de forma autónoma.

4.2. IMPLEMENTACIÓN DEL MODELO CLIENTE/SERVIDOR BASADO EN UDT

4.2.1. Implementación del servidor basado en UDT sobre el nodo que aloja al repositorio

En esta sección se trabaja sobre las ideas expuestas en la sección 3.2.2 detallando la parte técnica de la implementación del servicio de recepción de archivos e ingestión en el nodo que tiene alojado el repositorio de datos.

La primera parte de la implementación es necesario contar con la instalación de un software para repositorios que cuente con herramientas de ingestión a través de la ventana de comandos, tal como DSpace[35], Fedora[36] o EPrints[37]. Para este trabajo se decide utilizar DSpace como base, por ser el más utilizado y diseminado a nivel mundial. Es importante subrayar que la utilización de UDT no tiene relación alguna con el tipo de repositorio implementado y los métodos de ingestión a cada software de repositorios son diferentes, por ello la rutina SAF debe ser adaptada cada vez; en este estudio solo se abordaron las relacionadas con DSpace.

Otras tareas prioritarias son la compilación de la librería de UDT, la ubicación del *header file* y la configuración de la variable de entorno, lo cual permitirá poner en ejecución la rutina que servirá de servidor y atenderá las peticiones de los diferentes clientes (detectores) que deseen ingerir datos al repositorio. Se debe tener en consideración que tanto la librería como las rutinas del servidor fueron programadas

[35] DSpace: http://www.dspace.org/
[36] Fedora: http://www.fedora-commons.org/
[37] EPrints: http://www.eprints.org/uk/

en C/C++ y por lo cual se es dependiente de la arquitectura hardware en donde se desee ejecutar. Ahora bien, para la compilación se deben colocar una serie de parámetros los cuales le indican al compilador en donde encontrar la librería UDT, además de otros, que le indican las librerías especializadas que usa la rutina. A continuación se presenta el comando usado para la compilación de la rutina *server*:

```
gcc -o server server.cpp -L path-to-library -ludt -lstdc++ -lpthread -lm
```

Tras concluir los requisitos de implementación se debe configurar el archivo que contiene el puerto en donde se escucharán las peticiones, la ruta en la cual se creará la estructura SAF y la ruta en donde se debe buscar las herramientas de ingestión. Finalmente, solo queda poner en ejecución el nuevo servicio que puede hacerse mediante la simple ejecución de la rutina o creando un nuevo servicio Linux.

4.2.2. Implementación del cliente basado en UDT sobre las SBC

Anteriormente se estableció los requerimientos para implementar y ejecutar el servicio encargado de la recepción e ingestión de datos al repositorio, mientras que en esta sección se discutirá particularmente la implementación del cliente en las SBC y de la correcta compilación de la librería para la ejecución de los clientes.

Cabe recordar que las SBC utilizadas en este trabajo poseen diferentes tipos de procesadores ARM (sección 2.3.2) y que la librería UDT no contiene dentro del *Makefile* los parámetros que soporten la compilación sobre este tipo de arquitecturas. Por tanto, este archivo debe ser modificado agregándole las siguientes líneas:

```
ifeq ($(arch) , ARM)
        CCFLAGS += −march=armv6
endif
```

El parámetro *-march* describe la arquitectura de procesador (para una lista completa de las arquitecturas soportadas puede verse el manual de *gcc* ARM[38]). Con respecto al valor dado para este parámetro, armv6 es usado para especificar SBC como la Raspberry Pi o la Cubieboard, mientras que para las Jetson TK1 es necesario cambiarlo a armv7.

Volviendo la mirada a la Figura 8, puede observarse que el cliente propuesto es bastante sencillo en comparación al cliente SWORD y su implementación es llevada a cabo mediante la configuración de los archivos de propiedades descritos en el modelo, la compilación de la librería usando la especificación de la nueva

[38] ARM-gcc: https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html

arquitectura, la ubicación del *header file*, la configuración de la variable de entorno y la compilación de la rutina cliente, que se realiza de forma similar a la del servidor, tal como se muestra a continuación:

```
gcc -o client client.cpp -L path-to-library -ludt -lstdc++ -lpthread –lm
```

Finalmente, solo queda ponerlo en ejecución y al igual que la rutina del servidor puede crearse un servicio Linux para la ejecución de este nuevo servicio.

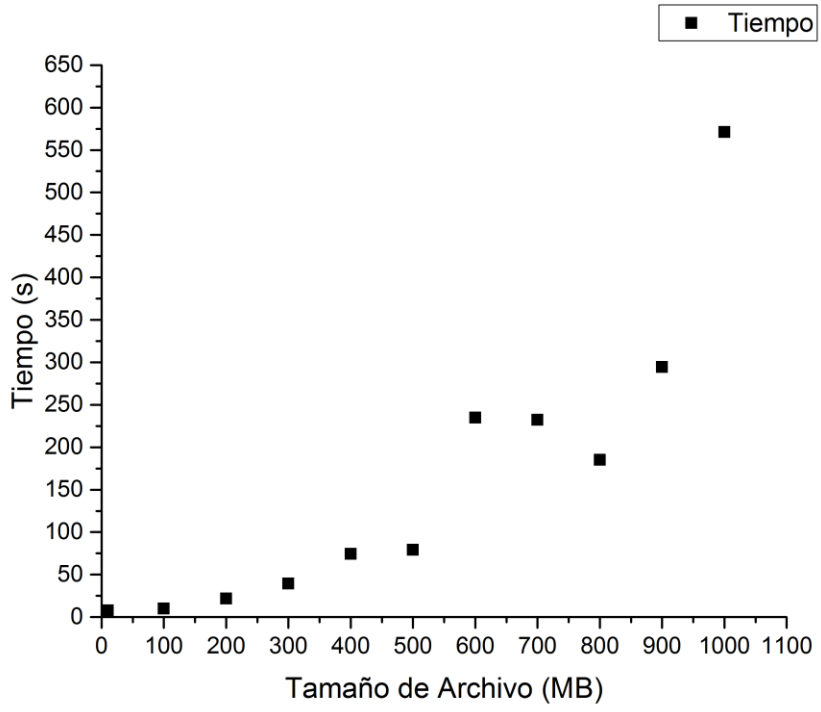
5. RESULTADOS Y EVALUACIÓN

5.1. MODELO BASADO EN SWORD

Para medir el rendimiento y comportamiento del cliente se realizaron varias pruebas desde diferentes plataformas hardware para verificar su funcionamiento y sobre todo para medir la capacidad de SWORD para transmitir datos de gran volumen.

La primera prueba fue ejecutada una única vez con el único fin de comprobar el funcionamiento del cliente SWORD y verificar su capacidad para crear los metadatos y transmitir los datos al repositorio. Ésta fue llevada a cabo sobre el mismo nodo que contiene al repositorio de prueba anulando de esta forma algún problema que pudiera producirse por la transmisión de los datos a través de la red. En la Figura 9 puede observarse los tiempos que le ha llevado al cliente transferir archivos de datos desde los 10 MB hasta 1 GB.

Figura 9. ***Test de control SWORD.*** *La prueba de control es ejecutada directamente sobre el nodo que aloja al repositorio de datos. La mayor tasa de transferencia se obtiene para el paquete de datos de 100 MB con un valor de 78,74 Mbps.*



En esta prueba que las mayores tasas de transferencia obtenidas fueron de 78,74 Mbps para el paquete de datos de 100 MB seguida por la del archivo de datos de

200 MB con una tasa de transferencia de 73,36 Mbps. Por otra parte, se observan incrementos bastante altos en el tiempo de ingestión de los datos para el resto de archivos en comparación con los resultados presentes para paquetes de datos hasta los 200 MB. Éste aumento en los tiempos hace que las tasas de transferencia sean cada vez menores, como por ejemplo para el paquete de datos de 1 GB que fue tan sólo de 14 Mbps. Según estos resultados, el cliente presenta sus tasas de transferencia más altas para archivos de datos entre los 100 MB y 200 MB, pero de igual manera, éste cumplió el objetivo de crear los metadatos e ingerir los datos al repositorio sin ningún inconveniente a pesar del tamaño del archivo.

Una vez comprobado el funcionamiento del cliente, se procedió a evaluarlo en dos equipos de diferente arquitectura conectados a una red Gigabit, con lo cual se pudo medir la capacidad de transferencia de datos hacia el repositorio. La primera prueba se realizó desde una PC de escritorio con un procesador Intel Pentium 4 de 3.2 GHz, 2 GB de memoria RAM y un controlador de red LAN Gigabit Ethernet Broadcom de 10/100/1000, mientras la segunda fue realizada en una SBC con un SoC Allwinner A20 el cual posee un procesador ARM Cortex-A7 dual-core de 1 GHz, 1 GB de memoria RAM y un controlador de red LAN Ethernet 10/100, denominada Cubieboard (ver sección 2.3.2).

Para la primera arquitectura se realizaron cinco test, los cuales fueron hechos uno por día (lunes a viernes); el primero de ellos a las 8 a.m., el segundo a las 10 a.m., el tercero a las 2 p.m., el cuarto a las 4 p.m. y el último a las 6 p.m. Esto fue hecho para sobrellevar inconvenientes que pudieran presentarse en la red, tales como son los aumentos en la latencia o problemas con el ancho de banda durante horas de alta congestión. De la misma manera se quiere evitar que los resultados se vean comprometidos y el desempeño del cliente SWORD sea mal evaluado. En la Figura 10 se muestran los tiempos obtenidos de los cinco test realizados a archivos de datos desde los 10 MB hasta los 300 MB. Igualmente se muestra en la gráfica el tiempo promedio para cada uno de los archivos de datos transferidos por el cliente.

Inicialmente al realizar las pruebas no se lograron sobrepasar la ingestión de archivos por encima de los 10 MB, esto fue debido a la configuración que viene por defecto en las librerías de desarrollo de Python y por lo cual se hizo necesario buscar y modificar la configuración que ocasionaba esta problemática. Al analizar las líneas de código fuente se observó que el *timeout* de la conexión *http* era de 10 y no se lograba mantener la conexión por suficiente tiempo para permitir la transferencia de datos. Por otro lado, un *timeout* muy grande producía el mismo efecto y no se logrará finalizar las ingestiones al repositorio. A continuación se presenta las líneas de código en donde se encuentran los *timeout* a modificar en dos de los archivos fuente de la librería de desarrollo para Python:

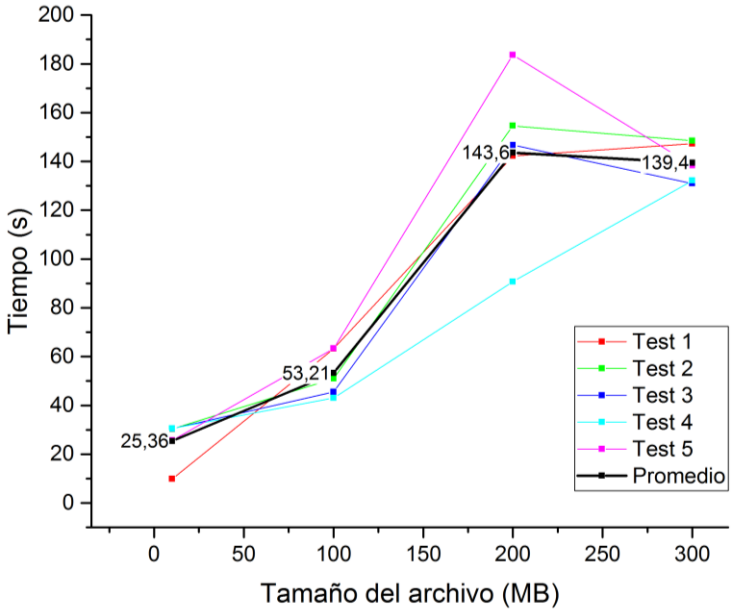
http_layer.py

```
Class
HttpLib2Layer(HttpLayer):
def __init__(self, cache_dir, timeout=180,
ca_certs=None):
self.h = httplib2.Http(".cache", timeout=180.0,
ca_certs=ca_certs)
```

Connection.py

```
self.h = http_layer.HttpLib2Layer(".cache", timeout=180.0,
ca_certs=ca_certs)
```

Figura 10. ***Test realizados al cliente SWORD desde una PC hasta el repositorio de datos.*** *Los cinco test son ejecutados en días y horas diferentes. Se presenta un promedió de estas cinco pruebas en donde la mayor tasa de transferencia fue de 17,12 Mbps para el archivo de 300 MB con un tiempo de 139,4 segundos.*



Después de las modificaciones realizadas a la librería, se logró realizar transferencias de archivos de datos por encima de los 10 MB pero sin lograr superar los 300 MB, tal como se presenta en la Figura 10. En ésta se observaron disminuciones en las tasas de transferencia de los datos en donde el mayor promedio fue para el archivo de 300 MB con un valor de 17,12 Mbps en contraste con los resultados anteriores en donde la tasa para este mismo archivo fue de 60,91 Mbps. Cabe recordar que las primeras pruebas no son realizadas a través de una red sino dentro del mismo nodo que contiene al repositorio y por ello, no se presentan perdidas de conexión por el *timeout* permitiendo tasas de transferencia mayores e ingestiones de archivos de datos de mayor tamaño.

Para la segunda arquitectura se realizaron 5 test de forma similar a los realizados para la arquitectura anterior y con los mismos fines. En la Figura 11 se presentan los resultados obtenidos para los 5 test y el promedio entre ellos. Al igual que para

el caso anterior, la mejor tasa de transferencia promedio se obtuvo para el archivo de datos de 300 MB con un valor de 11,78 Mbps, siendo este mucho menor que el presentado por la arquitectura anterior. En la Figura 12 se observa que en la gran mayoría de los casos se presentaron mejores tiempos desde la PC.

Figura 11. ***Test realizados al cliente SWORD desde la SBC Cubieboard hasta el repositorio de datos.*** *Los cinco test son ejecutados en días y horas diferentes. Se presenta un promedió de estas cinco pruebas en donde la mayor tasa de transferencia fue de 11,78 Mbps para el archivo de 300 MB con un tiempo de 203,8 segundos.*

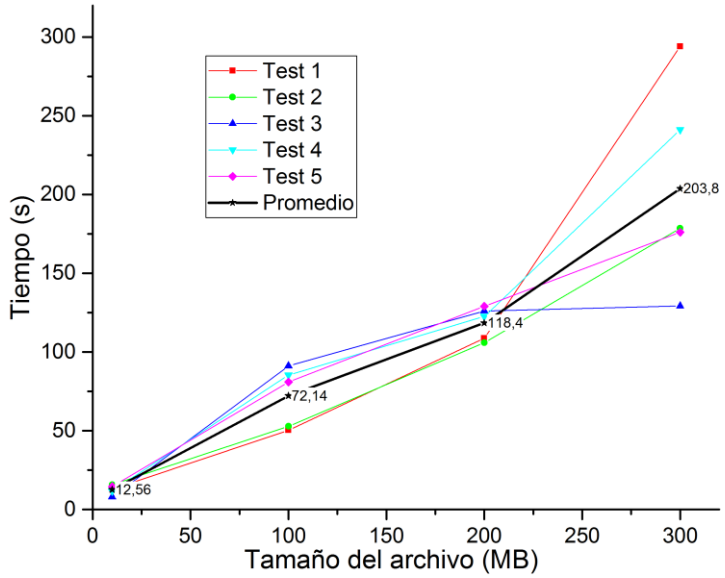
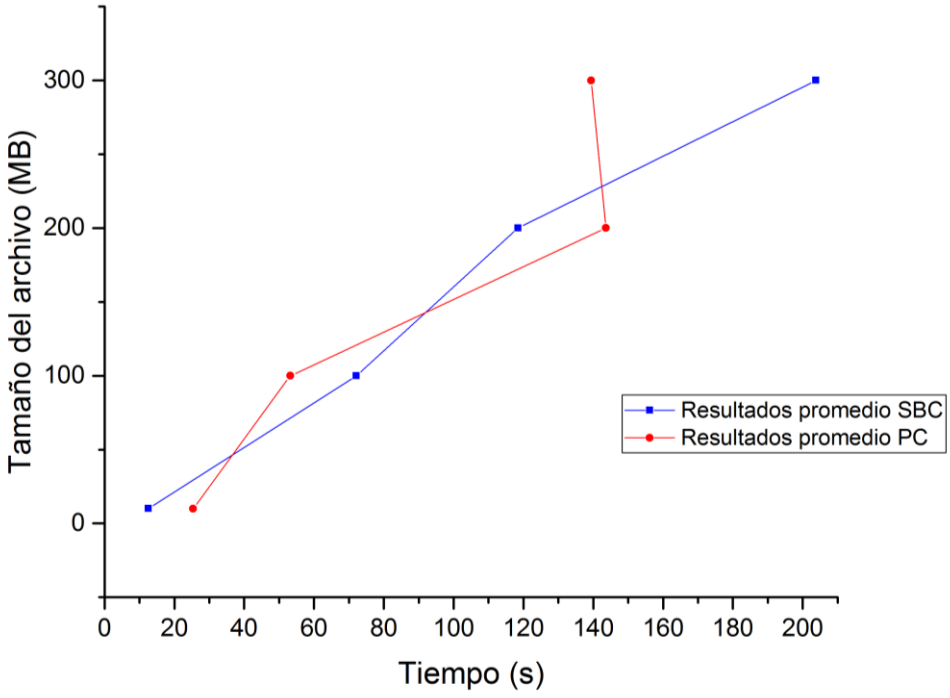


Figura 12. ***Comparación de los promedios obtenidos en la PC y la SBC.*** *En Azul se observa los resultados promedios obtenidos de los test realizados al cliente SWORD para transferir datos desde una PC hacia el repositorio. En rojo se observan los resultados promedios obtenidos desde una SBC hacia el repositorio.*



5.2. MODELO BASADO EN UDT

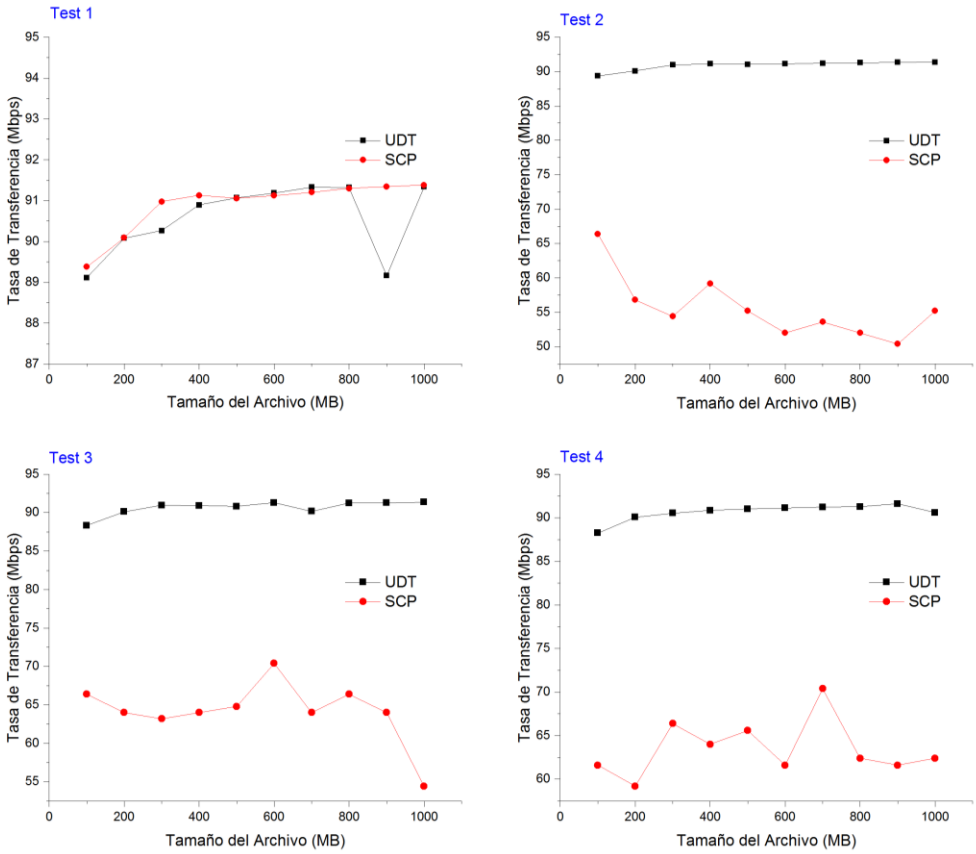
Antes de revisar los resultados obtenidos, cabe recordar que UDT cuenta con una librería que permite la utilización de sockets para el envío/recibo de paquetes de información y que ésta cuenta con un algoritmo de control de transmisión que ajusta la tasa de envió según la calidad de la red en donde se transmite. Esto quiere decir que en redes de baja calidad (alta latencia y poco ancho de banda), UDT transmitirá con la misma velocidad que lo haría cualquier herramienta de transferencia de datos soportada por el protocolo TCP, mientras que en redes de alta calidad, las velocidades de transferencia pueden llegar a oscilar cerca a la velocidad máxima permitida, tanto por la red como por el controlador de red que se esté usando. En base a lo anterior, se decide comparar las tasas de transferencia obtenidas por el modelo cliente servidor propuesto con las obtenidas por alguna herramienta de transferencia de datos que este soportada por el protocolo TCP, tal como *scp*.

Con respecto a lo anterior, el modelo cliente/servidor basado en UDT fue evaluado mediante la obtención de la tasa de transferencia de los datos y los metadatos al repositorio. Éstos fueron comparados con los producidos por herramientas estándar como *scp (Secure Copy Protocol)* o *rsync* permitiendo hacer evidente el incremento de la velocidad de transferencia con la utilización de UDT.

Para este modelo se realizaron cuatro test bajo las mismas condiciones que para el cliente SWORD, en donde cada uno de ellos fue realizado en un día y hora diferente con respecto al anterior, con el objeto de medir su funcionalidad y desempeño bajo diferentes condiciones de red. Las pruebas fueron realizadas de los días lunes a jueves; la primera de ellas realizada a las 10 a.m., la segunda a las 2 p.m., la tercera a las 4 p.m. y la última a las 6 p.m. En la Figura 13 se presentan cuatro gráficas, una por cada test, en donde se observa las diferencias entre el modelo basado en UDT y los resultados obtenidos al transferir archivos de tamaños iguales con *scp*. Estos test fueron realizados usando la misma SBC que para los realizados al cliente SWORD.

Una de las cosas a destacar es que UDT logra mantener una tasa de transferencia casi homogénea a diferencia de *scp* que como bien puede apreciarse oscila a lo largo de cada uno de los test. Otra cosa que resalta en los resultados obtenidos, es el hecho de que UDT como lo indica la literatura, siempre trata de transferir a la mayor velocidad permitida por el medio, con tasas de transferencia que oscilan en los 90 Mbps, en un medio en donde las condiciones son un controlador Ethernet de 10/100 conectado a una red Gigabit.

Figura 13. ***Comparación de las tasa de transferencia del modelo UDT – SBC Cubieboard.*** *Se presentan cuatro test utilizando tanto el modelo UDT como el estándar scp para la transferencia de datos. En la mayoría de los test se presenta un incremento en la transferencia de datos de alrededor de los 30 Mbps usando el modelo UDT.*

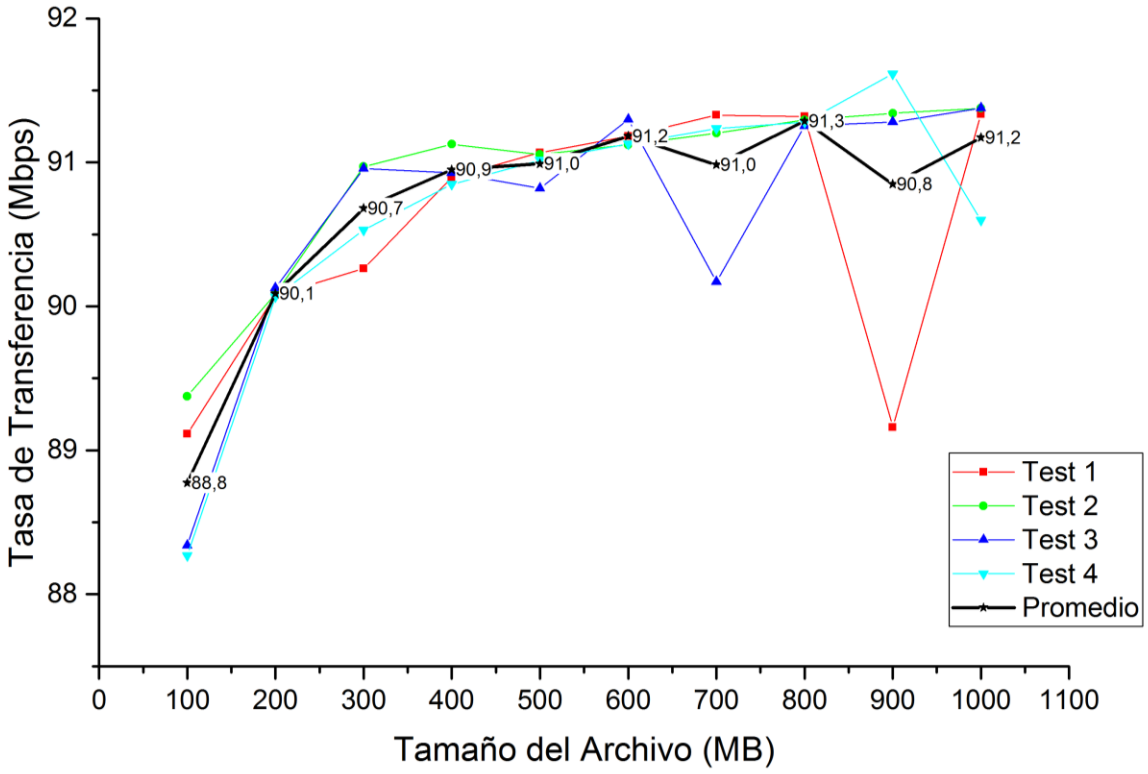


Con respecto a las comparaciones anteriores, se puede observar también un incremento de aproximadamente 30 Mbps en la tasa de transferencia dada por el modelo planteado en UDT con respecto al estándar *scp*, excepto para el primer test en donde los comportamientos son muy similares.

La razón de este comportamiento se debe al modo de verificación que presenta el algoritmo de control de congestión de TCP, el cual, al encontrarse en un medio que le proporcione bajas latencias, alto ancho de banda y sobre todo, mínima perdida de paquetes transmitidos, es capaz de incrementar su tasa de transferencia, manteniendo una comprobación constante de la llegada de paquetes.

Por otra parte, el algoritmo de control de congestión de UDT es adaptativo mostrando en este test, un comportamiento similar al de TCP. Este tipo de respuesta se debe a las fluctuaciones presentadas por las condiciones de la red, comportándose acorde con la descripción mostrada en la sección 2.1.1.

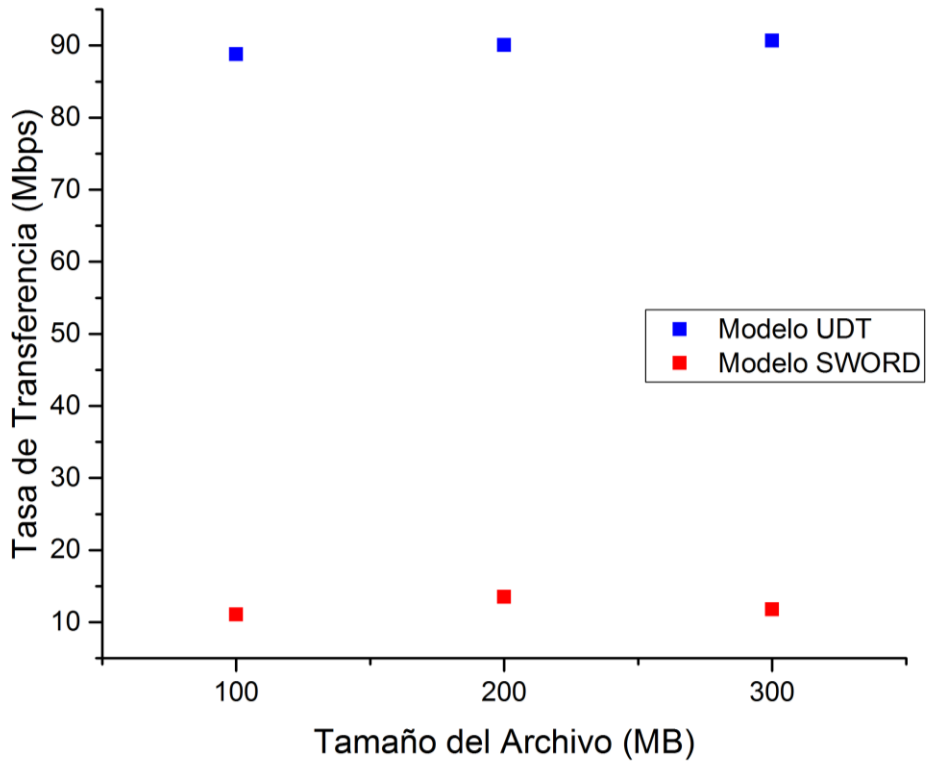
Figura 14. ***Comparación de los test y promedio de los resultados del modelo UDT – SBC Cubieboard.*** *Del promedio obtenido de los cuatro test la mayor tasa de transferencia 91,3 Mbps pero de forma general los resultados de todas las pruebas oscilan a valores cercanos a éste.*



En la Figura 14 se observan los cuatro test realizados y las tasas de transferencia promedio para cada uno de los datos transmitidos. En resumen puede verse que la mayor tasa de transferencia fue de 91,3 Mbps pero en general, el modelo planteado con UDT manejó tasas que oscilaron en valores cercanos a éste.

Volviendo la mirada hacia los resultados obtenidos en cuanto a la ingestión de datos mediante el modelo basado en SWORD y al compararlos con el basado en UDT, puede apreciarse que la tasa de transferencia máxima que presenta el primero modelo, tan solo fue de 18,57 Mbps en contraste con la más baja tasa obtenida para UDT de 90,26 Mbps para el mismo paquete de datos de 300 MB (Ver Figura 15). Es decir, el tiempo de ingestión de un archivo de datos de 300 MB para SWORD es de 129,18 segundos mientras que para el modelo UDT es de solo 26,58 segundos.

Figura 15. ***Comparación entre las tasas de transferencia de SWORD y de UDT en la SBC.*** *Se observan promedios de las tasas de transferencia obtenidos para archivos de datos de 100, 200 y 300 MB. En azul se presentan los resultados para el modelo UDT mientras que en rojo se presentan los resultados para el cliente SWORD.*



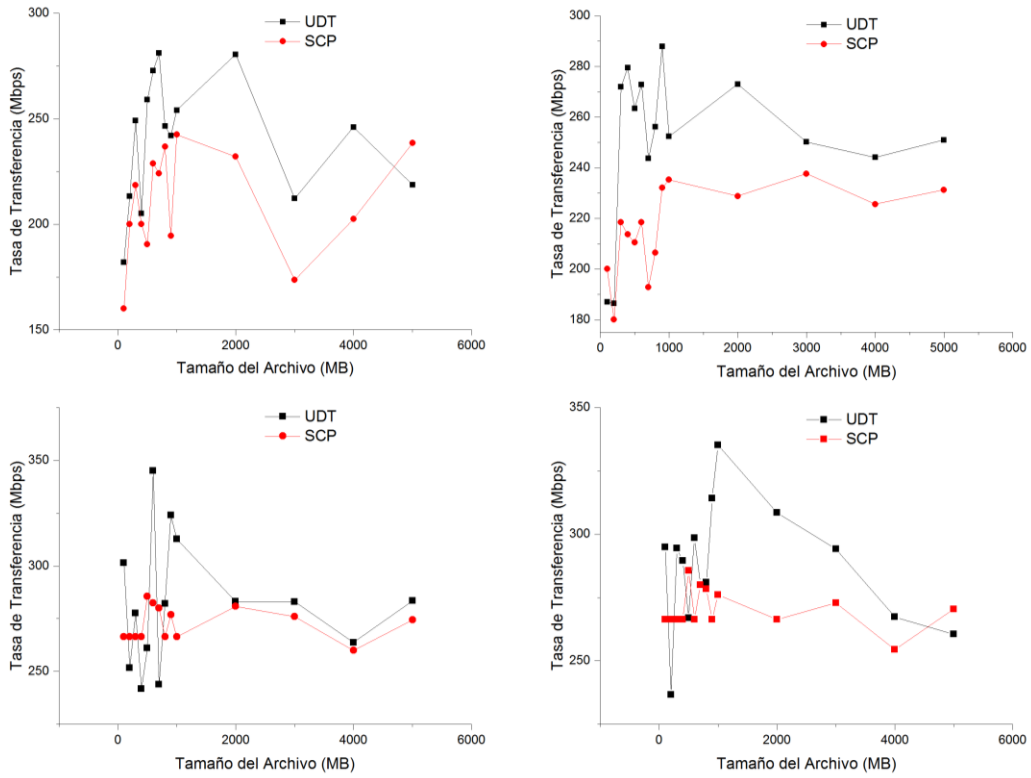
Igualmente, es importante subrayar el hecho que para SWORD no fue posible superar tamaños de archivos de datos por encima de los 300 MB, mientras que para modelo UDT se evaluaron hasta de 1 GB. Cabe aclarar que este límite se tomó para contrastar con el modelo basado en SWORD y no como limitante máxima para el modelo basado en UDT.

Hasta el momento se ha evaluado una arquitectura SBC con bajas prestaciones hardware, pero siendo suficientes para demostrar las ventajas del modelo de ingestión basado en UDT sobre el modelo basado en SWORD. Ahora bien, lo siguientes resultados se obtuvieron sobre la SBC Jetson TK1 de NVidia la cual provee características hardware mucho mayores en comparación con la anterior (ver sección 2.3.2).

Para esta arquitectura se siguió la misma metodología de evaluación que se ha mencionado anteriormente, realizando 4 test diferentes, en donde se contrasta los resultados obtenidos de las tasas de transferencia de UDT con los obtenidos por los

de *scp*. En la Figura 16 se observa de manera general que estos test se realizaron para archivos de datos desde los 10 MB hasta los 5 GB, lo cual permitió medir de forma más específica el comportamiento de UDT para archivos de gran tamaño.

Figura 16. ***Comparación de las tasa de transferencia del modelo UDT – SBC Jetson TK1.*** *Se presentan cuatro test utilizando tanto el modelo UDT como el estándar scp para la transferencia de datos. De forma general el modelo UDT presenta mejores resultados sobre el estándar scp, en donde la mayor tasa de transferencia de las pruebas fue de 344,9 Mbps.*



Con respecto a la Figura 16 se observa que a diferencia de la primera SBC, las tasas de transferencia son mucho mayores debido al controlador de red de gigabit que posee esta placa. Aun así, la tasa máxima de transferencia fue de tan sólo 344,9 Mbps en una red gigabit con un controlador capaz de transferir a esta velocidad. Este decremento en las tasas de transferencia se presenta principalmente porque a pesar de que la infraestructura de red posee un canal de un gigabit, el ancho de banda real es mucho menor y el algoritmo de control de congestión basa sus cálculos sobre el real y no el teórico. Por otra parte cabe resaltar que las tasas de transferencia de UDT fueron mayores que para las obtenidas para *scp* en la gran mayoría de los casos.

En la Figura 17 de presentan los cuatro test realizados y el promedio obtenido de cada una de las mediciones realizadas, siendo la mayor tasa de transferencia promedio de 297,27 Mbps.

Figura 17. ***Comparación de los test y promedio de los resultados del modelo UDT – SBC Jetson TK1.*** *Del promedio obtenido de los cuatro test la mayor tasa de transferencia 297,27 Mbps.*

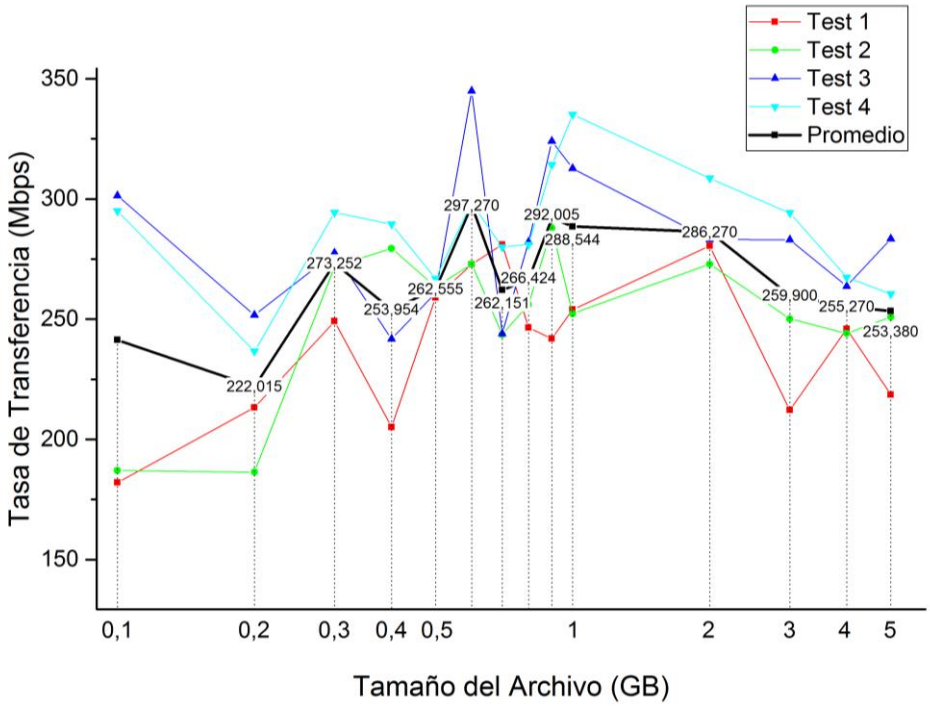
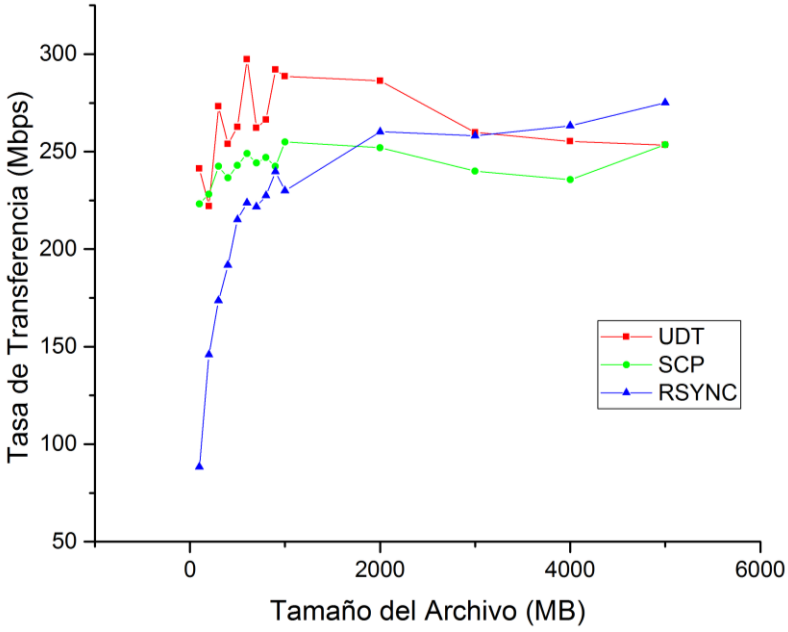


Figura 18. ***Comparación entre el modelo UDT, scp y rsync.*** *Las gráficas de UDT y scp son los promedios obtenidos de los cuatro test. La gráfica de rsync muestra los resultados de una única prueba realizada usando esta herramienta de transferencia. UDT presenta mejores tasas de transferencia para archivos de datos por debajo de los 3 GB en comparación con rsync mientras que supera siempre a scp.*



Como complemento, en la Figura 18 se muestran los resultados promedios de la evaluación de UDT, *scp* y *rsync*. La medición de este último se realizó una única vez y solo se muestra a manera de comparación, aclarando que este estándar de transferencia también es basado en el protocolo TCP. Finalmente, los resultados demuestran las ventajas del uso de UDT en el protocolo de transferencia e ingestión de datos, aún sobre el estándar de interoperabilidad SWORD.

Por último, es conveniente recordar que el algoritmo de control de congestión de UDT es adaptativo, lo que quiere decir que en redes de buena calidad las tasas de transferencia son altas y viceversa. Esto permite ver de manera general la calidad de la red donde se realizan las diferentes pruebas pero en redes de mejor calidad, según plantea Yunhong Gu y Rrobert L. Grossman, diseñadores del protocolo, pueden acercarse a los 940 Mbps sobre una conexión de 1 Gbps [16].

6. CASO DE IMPLEMENTACIÓN: LA COLABORACIÓN LAGO

El proyecto LAGO (Latin American Giant Observatory) es una colaboración iberoamericana de más de 80 investigadores en el área de astropartículas motivada por la experiencia del Observatorio Pierre Auger en Argentina. LAGO para la fecha, posee una red de 10 detectores de agua Cherenkov localizados en diferentes latitudes a lo largo del continente suramericano (Ver Figura 19).

Figura 19. ***Ubicación de los detectores LAGO a lo largo de América Latina.*** *Los detectores LAGO por lo general se encuentran en lugares de alta montaña a lo largo de la cordillera de los Andes. Sin embargo, existen otras instalaciones ubicadas en universidades pertenecientes a la colaboración.*

Normalmente, cada detector genera 150 GB de datos/mes y en todo el proyecto puede llegar a generar 1.5 TB/mes. Referente al tema de generación de datos, el proyecto puede producir alrededor 50 GB/sitio de datos simulados mediante la utilización de CORSIKA (COsmic Ray SImulation for KAscade)[39], un software para simular de forma detallada lluvias de partículas cósmicas de altas energías. Esta gran cantidad de datos son preservados en el repositorio del proyecto denominado LAGOData, siendo éste parte de un proyecto más ambicioso denominado LAGOVirtual[40], en donde además de los servicios de almacenamiento, catalogación y búsqueda de datos, contendrá herramientas de análisis y simulación.

[39] Corsika: http://www-ik.fzk.de/corsika/
[40] LAGOVirtual: http://lagoproject.uis.edu.co/LAGOVirtual/

6.1. REPOSITORIO DE DATOS DEL PROYECTO LAGO

Para los investigadores de la colaboración LAGO, se ha hecho fundamental catalogar y preservar de forma adecuada los datos producidos por los detectores en búsqueda de cumplir con los objetivos del proyecto. El análisis en tiempo real de los datos puede llegar a ser bastante complejo y llevar largos periodos de tiempo dependiendo de lo que se busque en ellos y aún más si son correlacionados entre varias instalaciones. Esta situación fue la que llevó a la creación del primer repositorio de la comunidad, además del hecho de poder intercambiar y compartir datos entre las diferentes instituciones participantes de varios países.

En el 2010 se adapta el software de código abierto DSpace para crear la primera versión del repositorio de datos de LAGO. Estas adaptaciones consistieron principalmente en modificar su interfaz gráfica y la creación de un pequeño esquema de metadatos que describía ciertas características importantes para los investigadores involucrados en el proyecto [9]. Esta versión del repositorio se mantuvo durante varios años con actualizaciones constantes y migración de datos con cada nueva versión de DSpace.

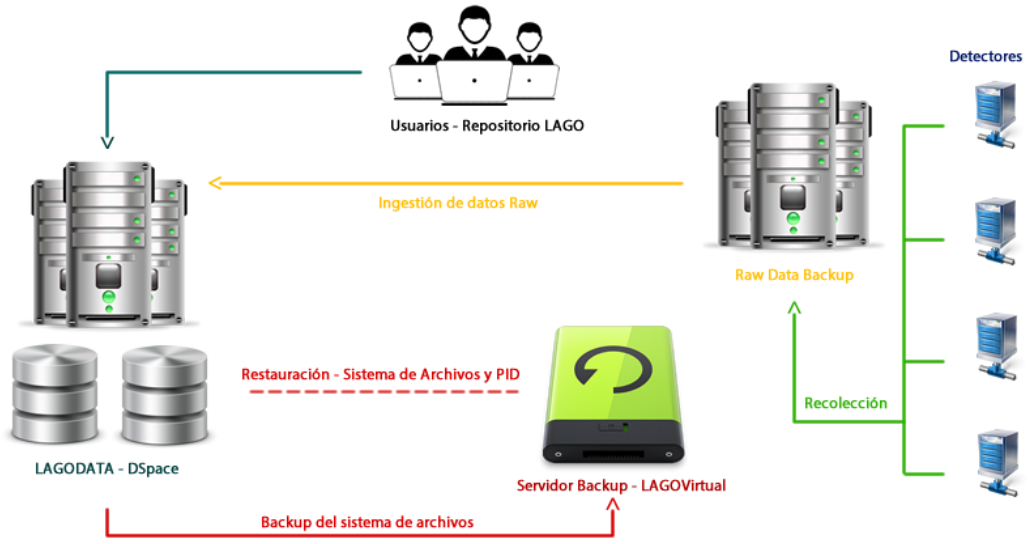
En la actualidad, el repositorio de datos LAGOData ha sufrido varias modificaciones significativas, no solo en su interfaz gráfica sino en su código fuente y archivos de configuración para implementar el nuevo sistema PID (*persistent identifiers*) de GRNET. Al inicio del proceso de desarrollo del nuevo repositorio fueron modificados los archivos de Bootstrap personalizando la interfaz a la imagen presentada por el proyecto LAGO y posteriormente con la colaboración de GRNET[41] se adaptó el repositorio para que en cada ingestión de datos se le proveyera un identificador único con el fin de poder ser citados en investigaciones futuras.

Seguido a las modificaciones hechas a DSpace, se diseña una infraestructura donde pudiera preservarse el PID de cada dato ingerido en el repositorio. En la Figura 20 se observa que los datos de los detectores son recolectados por un servidor que posterior a la recolección realiza las tareas de ingestión al repositorio. Por otra parte, se cuenta con otro servidor que se encarga de respaldar el sistema mediante la realización de una copia espejo de todo el sistema LAGOData, tarea que permite preservar el identificador único de GRNET cuando se deba realizar una recuperación del sistema. El servidor encargado de la recolección mantiene los datos como respaldo por si los otros dos servidores llegaran a fallar. Cabe recalcar

[41] PID-GRNET: Es un servicio dedicado a brindar, resolver mantener identificadores persistentes. http://epic.grnet.gr/

que para este caso solo se conservarían los datos en bruto y el identificador único deberá recuperarse por otro medio.

Figura 20. ***Estructura computacional de LAGODATA.*** *Los datos de los detectores son recolectados por el servidor Raw Data Backup y posteriormente son ingeridos al repositorio LAGODATA. Este servidor es respaldado de forma completa por el servidor de Backup y en caso de caída del repositorio es restaurado en base a este.*



Finalmente, esta nueva versión se puso en producción a finales del 2014 y cuenta actualmente con más de veinticinco mil archivos de datos producidos por los detectores del proyecto LAGO. Se espera que para finales de 2015 el repositorio albergue la totalidad de los datos producidos por los detectores (incluyendo aquellos que por su localización en sitios de alta montaña no pueden ser conectados a una red) junto con datos simulados y analizados.

6.2. INGESTIÓN MASIVA DE DATOS EN EL PROYECTO LAGO

LAGOData cuenta con las ventajas y desventajas de DSpace por ser una adaptación de éste y entre las que más destaca es la imposibilidad de ingerir múltiples archivos a través de su *workflow* en la interfaz gráfica. Debido a esto, LAGO realiza un script con el cual se hace posible realizar ingestiones masivas de datos mediante la utilización de la herramienta *import*. Posteriormente se realiza un programa en Java que logro cumplir las mismas funciones del script, pero con las ventajas de poder extraer la información necesaria para construir los archivos de metadatos y de generar la estructura SAF que se presenta en la Figura 21.

Figura 21. ***Estructura SAF implementada en LAGO****. Los archivos descritos aquí contienen la misma información de la estructura típica SAF con la excepción que para la estructura de LAGO se utiliza el esquema personalizado de metadatos metadata_LAGO.xml.*

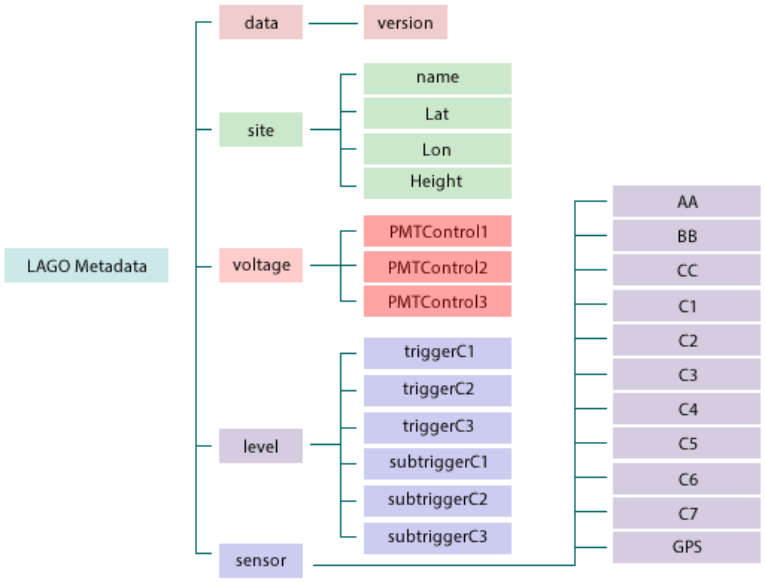
```
./item993:
total 281M
-rwxrwxrwx 1 lago lago   31 Apr 17 10:36 contents
-rwxrwxrwx 1 lago lago  353 Apr 17 10:36 dublin_core.xml
-rwxrwxrwx 1 lago lago 281M Feb 19  2014 guane3_2014_02_19_04h00.dat.bz2
-rwxrwxrwx 1 lago lago 1.5K Apr 17 10:36 metadata_LAGO.xml

./item994:
total 32M
-rwxrwxrwx 1 lago lago   31 Apr 17 10:36 contents
-rwxrwxrwx 1 lago lago  353 Apr 17 10:36 dublin_core.xml
-rwxrwxrwx 1 lago lago  32M Jan  9  2014 guane3_2014_01_09_05h00.dat.bz2
-rwxrwxrwx 1 lago lago 1.5K Apr 17 10:36 metadata_LAGO.xml

./item995:
total 233M
-rwxrwxrwx 1 lago lago   31 Apr 17 10:36 contents
-rwxrwxrwx 1 lago lago  353 Apr 17 10:36 dublin_core.xml
-rwxrwxrwx 1 lago lago 233M Dec 10  2013 guane3_2013_12_10_13h00.dat.bz2
-rwxrwxrwx 1 lago lago 1.5K Apr 17 10:36 metadata_LAGO.xml
```

En esta aplicación se utiliza tanto el esquema de datos DublinCore como el esquema de metadatos personalizado LAGO (ver Figura 22), el cual proporciona información acerca de las configuraciones de los detectores y de los parámetros de captura de los datos.

Figura 22. ***Esquema de metadatos LAGO.*** *Describe principalmente la configuración con la cual fueron obtenidos los archivos de datos. Esta información es contenida dentro de la cabecera de cada uno de los archivos generados y almacenada dentro de la estructura SAF en el archivo metadata_LAGO.xml.*

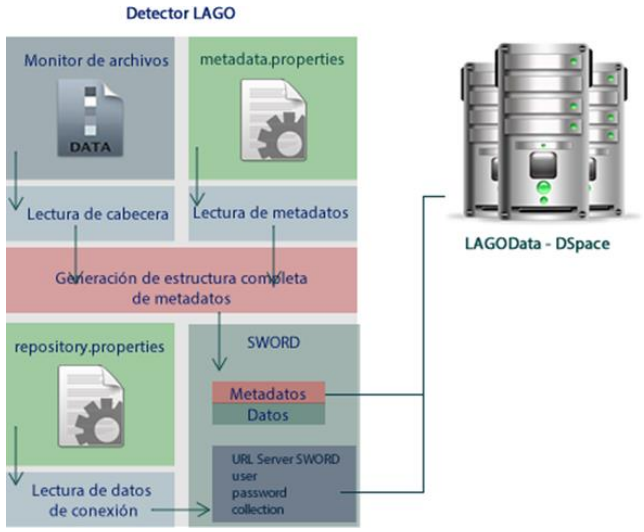


Aplicación del modelo Cliente/Servidor en el proyecto LAGO

La aplicación desarrollada por LAGO permitió ingerir el contenido actual del repositorio, pero aún se contaba con el factor humano en las tareas de catalogación e ingestión de datos y por tanto, se decide verificar los beneficios de SWORD en tareas de ingestión masiva de datos y sobre todo en tareas de ingestión de datos de gran volumen.

Después de resumir las características generales de SWORD en los capítulos anteriores y al ver que las aplicaciones desarrolladas por LAGO y descritas en la sección anterior, no permitían mantener el repositorio actualizado, se decide brindarle “inteligencia” a los detectores, permitiéndoles realizar ingestiones directamente al repositorio sin la necesidad de alguien en medio que realizara las tareas de catalogación e ingestión de datos.

Figura 23. ***Esquema del cliente SWORD para LAGO.*** *Este esquema está basado en el diseño propuesta en la sección 3.1.2 y su funcionamiento es como el descrito en tal sección.*



Con el planteamiento anterior, se desarrolla un nuevo cliente para el servidor SWORD provisto en LAGOData. En la Figura 23 se muestra de forma detallada el esquema para la implementación del cliente en los detectores LAGO los cuales están soportados por un sistema de adquisición basado en FPGAs y además con un SBC como unidad de comunicación. Finalmente, el modelo cliente/servidor basado en SWORD fue probado con los archivos de datos generados por LAGO y como se vio en el capítulo anterior los tiempos de trasferencia así como el tamaño de los datos a transmitir no fueron óptimos.

La utilización de otros esquemas de metadatos por parte de SWORD no es posible como ya se ha mencionado en varias ocasiones a lo largo de este trabajo, por tanto, la descripción detallada de los datos de LAGO queda reducida a los metadatos que proporciona DublinCore y mucha de la información de los experimentos no puede ser agregada haciendo que la búsqueda dentro del repositorio se vea restringida.

Por otra parte, se plantea un modelo cliente/servidor basado en UDT el cual retoma la idea original de LAGO, en donde las ingestiones se realizaban a través de la herramienta *import* y se es posible la utilización de cualquier esquema de metadatos.

Manteniendo el mismo enfoque con el cual se desarrolló el cliente SWORD, el modelo cliente/servidor UDT proporcionó resultados satisfactorios, tal cual se presentaron en el capítulo anterior. Cabe resaltar que el modelo UDT presenta mejores tiempos de transferencia de datos respecto al modelo SWORD.

7. CONCLUSIONES Y TRABAJO FUTURO

7.1. CONCLUSIONES

Se concluye que SWORD basado en el estándar Atom Publising, no está diseñado para transferir datos de gran volumen y además de ello, subutiliza el ancho de banda de la red, causando demoras en la transmisión e impidiendo la alta disponibilidad de los datos de investigación, en particular aquellos de gran tamaño.

La herramienta original de SWORD es una herramienta que proporciona una gran flexibilidad para la ingestión hacia repositorios digitales, siempre y cuando se mantenga el uso del esquema DublinCore y los datos a transferir no sean de gran tamaño. Cabe recordar que incluso con modificaciones a los archivos de la librería no fue posible superar los 300 MB.

Dentro de este marco, las herramientas proporcionadas para el desarrollo de clientes SWORD se encuentran bastante limitadas y no proporcionan todas las funcionalidades presentadas por el estándar de interoperabilidad planteado por JISC. Entre éstas la que mayor destaca es la imposibilidad de ingerir datos con esquemas de metadatos diferentes para las librerías Java y Python. En resumen, si es necesaria la utilización de otros esquemas de metadatos, debe recurrirse a algún otro medio de ingestión remota.

Analizando las tasas de transferencia, puede observarse que a pesar de que la SBC Cubieboard conectada a una red Gigabit, no logra sobrepasar las 18,58 Mbps, Por otra parte, los resultados obtenidos al ejecutarse el cliente en el mismo nodo que contiene al repositorio de datos, no sobrepasaron los 78,74 Mbps, con lo cual se ve que SWORD a pesar una herramienta de gran flexibilidad para ingestiones hacia repositorios, no es una buena herramienta de transferencia de datos.

En atención a la problemática expuesta en la implementación de SWORD, se realizó una nueva búsqueda para hallar una alternativa viable que permitiera cumplir con los objetivos de ingerir datos de gran volumen y de forma masiva a un repositorio de datos. Por lo cual, se decide unir un protocolo como UDT junto con las herramientas de ingestión de consola que proporcionan los software de repositorios. Ahora bien, se observó que UDT supera en gran medida el uso de cualquier herramienta basada en TCP tal como *scp* o *rsync*, produciendo tasas de transferencia bastante altas a pesar de que la red en la que se ejecutaron las pruebas presenta problemas de estabilidad.

Por tanto, el uso de este protocolo es viable para la transferencia de datos y metadatos, obteniendo tiempos bajos en la transmisión y asegurando la integridad de los datos al ser ingeridos al repositorio.

Para finalizar, el uso de UDT y del modelo cliente/servidor planteado permite la ingestión de datos de gran volumen y siendo el servidor propuesto una aplicación multihilo, es viable la ingestión masiva de forma concurrente en el repositorio de datos y sin las limitaciones presentadas en la implementación de SWORD.

De otro lado, durante el planteamiento inicial del trabajo, se buscaba poder realizar de forma directa la implementación de estos protocolos sobre detectores que fueran soportados por SBC o FPGA, pero durante el desarrollo del mismo sólo fue posible realizarlas en las SBC.

Esta problemática sobre el uso de FPGA se debió principalmente por limitaciones con el hardware con el que se contaba para la realización de las implementaciones. Profundizando al respecto, las FPGA son dispositivos que pueden soportar lo que se conoce como *softprocessor*; procesadores basados en su gran mayoría en la arquitectura RISC[42] y programados en un lenguaje especializado para la descripción de circuitos tal como VHDL[43] y/o Verilog[44]. Sin embargo, no todas las FPGA cuentan con las suficientes compuertas lógicas programables para soportarlos.

Por otro lado, no solo es necesario un *softprocessor* en la FPGA para poder realizar las implementaciones de los protocolos descritos en este trabajo, también es necesaria la instalación de un Linux con un *kernel* muy particular que soporte este tipo de procesadores. Tal es el caso de µClinux[45], un derivado del *kernel* original de Linux adaptado para sistemas que carecen de MMU (*Memory management unit*) pero con toda la pila de protocolos TCP/IP y con soporte para sistema de archivos tales como ext2, NFS, FAT32 entre otros.

De las afirmaciones anteriores cabe concluir que la implementación del protocolo basado en SWORD y el basado en UDT es posible. Al respecto, las FPGA de bajo costo, como las disponibles para el desarrollo de este trabajo, no logran proporcionar los suficientes recursos hardware para soportar un *softprocessor*, un Linux embebido y un sistema de transferencia de datos ejecutándose en forma

[42] RISC (Reduced Instruction Set Computer) es un diseño de CPU usado principalmente en microcontroladores o microprocesadores.
[43] VHDL: Lenguaje definido por la IEEE para la descripción de circuitos digitales. Aunque puede ser usado de forma general para describir cualquier circuito digital es usado principalmente para programar PLD (Programable Logic Device), FPGA, ASIC y similares.
[44] Verilog: Lenguaje de descripción de hardware usado para modelar sistemas electrónicos y con la capacidad de programar FPGA, ASIC entre otros. Tiene la particularidad de tener una sintaxis similar a la del lenguaje de programación C.
[45] µClinux: http://www.uclinux.org/

simultánea. Sin embargo, las FPGA de mediando y alto costo cuentan con los suficiente recursos para implementar lo anteriormente descrito.

7.2. TRABAJO FUTURO

Como se explicó a lo largo del trabajo y se resumió en las conclusiones, SWORD es una alternativa viable para la ingestión remota de datos siempre y cuando estos no sean de un tamaño considerable. Por tanto, se plantea como trabajo futuro en esta línea:

- Corrección de la librearía Java para el desarrollo de nuevos clientes y finalización de ésta para la total implementación del estándar SWORD presentado por JISC. Estas correcciones incluyen el uso de esquema de metadatos diferentes al DublinCore.
- En cuanto a Python, se vio la posibilidad de corregir los *timeout* permitiendo aumentar el tamaño de los archivos a transferir. Sin embargo existe la posibilidad de modificar el servidor en búsqueda de aumentar el tamaño de datos que puedan ser transferidos, aunque debe tenerse en cuenta que estas modificaciones pueden llegar a usar mucho más otros recursos, tales como procesador o memoria, para lo cual deben realizarse toda una serie de mediciones y verificar su viabilidad.

En cuanto a UDT se plantean los siguientes puntos:

- Integración directa del modelo cliente/servidor UDT al repositorio sin la necesidad de implementar el uso de la herramienta *import*. Con esto se proveería a herramientas como DSpace, una alternativa viable para la ingestión masiva de datos de gran volumen a los repositorios de investigación.
- Por otra parte cabe mencionar que para este trabajo no se evaluó el uso de CPU, ni memoria por parte de UDT en los clientes que se ejecutaron en las SBC debido a que no era parte del presente trabajo. Sin embargo, si las SBC prestan otra función además de la transferencia, puede llegar a obtenerse valores totalmente diferentes, hecho interesante a evaluar en diferentes tipos de redes con diferente calidad.

Por otra parte, se plantea evaluar estos protocolos en FPGA que cuenten con los recursos suficientes para su implementación, con el fin de observar sus rendimientos y verificar la viabilidad de uso directamente sobre estos dispositivos.

BIBLIOGRAFÍA

Alachiotis, N., Berger, S. A. and Stamatakis, A. “Efficient PC-FPGA Communication over Gigabit Ethernet,” in 2010 10th IEEE International Conference on Computer and Information Technology, 2010, p. 1727–1734.

Albrecht, M. “Design of a Data Repository for a Long-running Physics Experiment,” M.S. Thesis, Computer Science and Engineering, University of Notre Dame, 2010.

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D. and Tuecke, S. “Data management and transfer in high-performance computational grid environments,” Parallel Comput., vol. 28, no. 5, May 2002, pp. 749–771.

Allinson, J., François, S., Lewis, S. and Project, J. S. “SWORD : Simple Web-service Offering Repository Deposit,” no. 54, 2008.

Bailey, D., Hughes-Jones, R. and Kelly, M. “Using FPGAs to Generate Gigabit Ethernet Data Transfers and Studies of the Network Performance of DAQ Protocols,” in 2007 15th IEEE-NPSS Real-Time Conference, 2007, pp. 1–6.

Barakat, C., Altman, E. and Dabbous, W. “On TCP performance in a heterogeneous network: a survey,” IEEE Commun. Mag., vol. 38, no. 1, 2000, p. 40–46.

Barros, H. “The LAGO Collaboration: Searching for high energy GRB emissions in Latin America,” in AIP, vol. 151, 2012, p. 151–158.

Beagrie, N. and Carpenter, L. “Development of Digital Preservation Environments by the UK Joint Information Systems Committee (JISC),” in 2005 IEEE International Symposium on Mass Storage Systems and Technology, 2005, p. 74–77.

Burckle, R. A. and WinSystems, V. P. “The Evolution of Single Board Computers,” WinSystems Inc, 2006.

Compton, K. and Hauck, S. “Reconfigurable computing: a survey of systems and software,” ACM Comput. Surv., vol. 34, no. 2, Jun. 2002, pp. 171–210.

DAVILA, J. A., NÚNEZ, L. A., SANDIA, B. and TORRENS, R. “Los repositorios institucionales y la preservación del patrimonio intelectual académico,” Interciencia, vol. 31, 2006, p. 22–28.

Fowers, J., Brown, G. Cooke, P. and Stitt, G. “A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications,” in Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA ’12, 2012, pp. 47–56.

Gu, Y. and Grossman, R. L. “UDT : UDP-based Data Transfer for High-Speed Wide Area Networks”, 2012, p. 1–19.

Hirose, J., Baba, K. and Abe, H. “A Protocol-Aware Network Control Method for Large-Scale Data Transfer in Long-Distance Broadband Networks,” 2011 Proc. 20th Int. Conf. Comput. Commun. Networks, Jul. 2011, p. 1–6.

Jacobson, V., Braden, R. and Borman, D. “TCP Extensions for High Performance.” RFC Editor, United States, 1992.

Kowalczyk, S. T. “Towards a model of the e-science data environment,” in Proceeding of the 11th annual international ACM/IEEE joint conference on Digital libraries - JCDL ’11, 2011, p. 399–400.

Iafolla, L., Balla, A., Beretta, M., Ciambrone, P., Gatta, M., Gonnella, F., Mascolo, M., Messi, R., Moricciani, D. and Riondino, D. “FPGA-based Time to Digital Converter and Data Acquisition system for High Energy Tagger of KLOE-2 experiment,” Nucl. Instruments …, vol. 718, May 2012, pp. 213–214.

Lewis, S. and Jones, R. “SWORD: Facilitating Deposit Scenarios,” D-Lib Mag., vol. 18, no. 1/2, Jan. 2012.

Lewis, S., Hayes, L., Newton-Wade, V., Corfield, A., Davis, R., Donohue, T. and Wilson, S. “If SWORD is the answer, what is the question?: Use of the Simple Web-service Offering Repository Deposit protocol,” Progr. Electron. Libr. Inf. Syst., vol. 43, no. 4, 2009, p. 407–418.

Marcial, L. H. and Hemminger, B. M. “Scientific data repositories on the Web: An initial survey,” J. Am. Soc. Inf. Sci. Technol., vol. 61, no. 10, Oct. 2010, pp. 2029–2048.

Prabhakaran, V. “High throughput data transfers using the Tornado transport protocol,” [online]. EEUU: University of Wisconsin-Madison, 2002. Disponible en: http://www.cs.wisc.edu/-pb/np_tr.pdf.

Riedel, M., Eickermann, T., Habbinga, S., Frings, W., Gibbon P., Mallmann, D., Wolf, F., Streit, A., Lippert, T., Wolf, F., Schiffmann, W., Ernst, A., Spurzem, R. and Nagel, W. E. “Computational Steering and Online Visualization of Scientific Applications on Large-Scale HPC Systems within e-Science Infrastructures,” in Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007), 2007, p. 483–490.

Sawyer, S., Kaziunas, E. and Øesterlund, C. “Social scientists and cyberinfrastructure,” in Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW ’12, 2012, p. 931–934.

Sayre, R. “Atom: The Standard in Syndication,” IEEE Internet Comput., vol. 9, no. 4, Jul. 2005, pp. 71–78.

Sun, J., Bittner, R. and Eguro, K. “FPGA side-channel receivers,” Proc. 19th ACM/SIGDA Int. Symp. F. Program. gate arrays - FPGA ’11, p. 267, 2011.

Thomas, A. “SWORD2 Project Final Report,” [online]. JICS, pp. 1–9, Jun 2009. Disponible en: http://www.jisc.ac.uk/media/documents/programmes/reppres/sword2finalreport.pdf

Tierney, B., Kissel, E., Swany, M. and Pouyoul, E. “Efficient data transfer protocols for big data,” in 2012 IEEE 8th International Conference on E-Science, 2012, p. 1–9.

Tomura, T., Hayato, Y., Ikeno, M., Nakahata, M., Nakayama, S., Obayashi, Y., Okumura, K., Shiozawa, M., Suzuki, S. Y., Uchida, T., Yamada, S. and Yokozawa, T. “Development of New Data Acquisition System for Nearby Supernova Bursts at Super-Kamiokande,” Phys. Procedia, vol. 37, Jan. 2012, p. 1398–1405.

Torres, L. A., Nuñez, L. A. and Torréns, R. “Implementación de un Repositorio de Datos Científicos usando Dspace,” e-colabora, vol. 1, no. 2, 2011, p. 101–117.

Witt, M. “Institutional Repositories and Research Data Curation in a Distributed Environment,” Libr. Trends, vol. 57, no. 2, 2008, p. 191–201.

Zhang, H., Wu, W. and Li, Z. “Open Social based group access control framework for e-Science data infrastructure,” in 2012 IEEE 8th International Conference on E-Science, 2012, p. 1–8.

REFERENCIAS BIBLIOGRÁFICAS

[1] M. Riedel, T. Eickermann, S. Habbinga, W. Frings, P. Gibbon, D. Mallmann, F. Wolf, A. Streit, T. Lippert, F. Wolf, W. Schiffmann, A. Ernst, R. Spurzem, and W. E. Nagel, “Computational Steering and Online Visualization of Scientific Applications on Large-Scale HPC Systems within e-Science Infrastructures,” in Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007), pp. 483–490, 2007.

[2] M. Albrecht, “Design of a Data Repository for a Long-running Physics Experiment,” M.S. Thesis, Computer Science and Engineering, University of Notre Dame, 2010.

[3] S. T. Kowalczyk, “Towards a model of the e-science data environment,” in Proceeding of the 11th annual international ACM/IEEE joint conference on Digital libraries - JCDL ’11, pp. 399–400, 2011.

[4] H. Zhang, W. Wu, and Z. Li, “Open Social based group access control framework for e-Science data infrastructure,” in 2012 IEEE 8th International Conference on E-Science, pp. 1–8, 2012.

[5] S. Sawyer, E. Kaziunas, and C. Østerlund, “Social scientists and cyberinfrastructure,” in Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW ’12, pp. 931–934, 2012.

[6] T. Tomura, Y. Hayato, M. Ikeno, M. Nakahata, S. Nakayama, Y. Obayashi, K. Okumura, M. Shiozawa, S. Y. Suzuki, T. Uchida, S. Yamada, and T. Yokozawa, “Development of New Data Acquisition System for Nearby Supernova Bursts at Super-Kamiokande,” *Phys. Procedia*, vol. 37, pp. 1398–1405, Jan. 2012.

[7] L. Iafolla, A. Balla, M. Beretta, P. Ciambrone, M. Gatta, F. Gonnella, M. Mascolo, R. Messi, D. Moricciani, and D. Riondino, “FPGA-based Time to Digital Converter and Data Acquisition system for High Energy Tagger of KLOE-2 experiment,” *Nucl. Instruments …*, vol. 718, pp. 213–214, May 2012.

[8] V. Prabhakaran, “High throughput data transfers using the Tornado transport protocol,” [online]. EEUU: University of Wisconsin-Madison, 2002. Disponible en: http://www.cs.wisc.edu/-pb/np_tr.pdf.

[9] L. A. Torres, L. A. Nuñez, and R. Torréns, “Implementación de un Repositorio de Datos Científicos usando Dspace,” *e-colabora*, vol. 1, no. 2, pp. 101–117, 2011.

[10] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul, “Efficient data transfer protocols for big data,” in 2012 IEEE 8th International Conference on E-Science, pp. 1–9, 2012.

[11] J. Hirose, K. Baba, and H. Abe, “A Protocol-Aware Network Control Method for Large-Scale Data Transfer in Long-Distance Broadband Networks,” *2011 Proc. 20th Int. Conf. Comput. Commun. Networks*, pp. 1–6, Jul. 2011.

[12] N. Beagrie and L. Carpenter, “Development of Digital Preservation Environments by the UK Joint Information Systems Committee (JISC),” in 2005 IEEE International Symposium on Mass Storage Systems and Technology, pp. 74–77, 2005.

[13] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, “Data management and transfer in high-performance computational grid environments,” *Parallel Comput.*, vol. 28, no. 5, pp. 749–771, May 2002.

[14] C. Barakat, E. Altman, and W. Dabbous, “On TCP performance in a heterogeneous network: a survey,” *IEEE Commun. Mag.*, vol. 38, no. 1, pp. 40–46, 2000.

[15] V. Jacobson, R. Braden, and D. Borman, “TCP Extensions for High Performance.” RFC Editor, United States, 1992.

[16] Y. Gu and R. L. Grossman, “UDT : UDP-based Data Transfer for High-Speed Wide Area Networks,” pp. 1–19.

[17] J. A. DAVILA, L. A. NUNEZ, B. SANDIA, and R. TORRENS, “Los repositorios institucionales y la preservación del patrimonio intelectual académico,” *Interciencia*, vol. 31, pp. 22–28, 2006.

[18] L. H. Marcial and B. M. Hemminger, “Scientific data repositories on the Web: An initial survey,” *J. Am. Soc. Inf. Sci. Technol.*, vol. 61, no. 10, pp. 2029–2048, Oct. 2010.

[19] M. Witt, “Institutional Repositories and Research Data Curation in a Distributed Environment,” *Libr. Trends*, vol. 57, no. 2, pp. 191–201, 2008.

[20] J. Allinson, S. François, S. Lewis, and J. S. Project, “SWORD : Simple Web-service Offering Repository Deposit,” no. 54, 2008.

[21] S. Lewis, L. Hayes, V. Newton-Wade, A. Corfield, R. Davis, T. Donohue, and S. Wilson, “If SWORD is the answer, what is the question?: Use of the Simple Web-service Offering Repository Deposit protocol,” *Progr. Electron. Libr. Inf. Syst.*, vol. 43, no. 4, pp. 407–418, 2009.

[22] R. Sayre, “Atom: The Standard in Syndication,” *IEEE Internet Comput.*, vol. 9, no. 4, pp. 71–78, Jul. 2005.

[23] A. Thomas, “SWORD2 Project Final Report,” [online]. JICS, pp. 1–9, Jun 2009. Disponible en: http://www.jisc.ac.uk/media/documents/programmes/reppres/sword2finalreport.pdf

[24] S. Lewis, P. de Castro, and R. Jones, “SWORD: Facilitating Deposit Scenarios,” *D-Lib Mag.*, vol. 18, no. 1/2, Jan. 2012.

[25] N. Alachiotis, S. A. Berger, and A. Stamatakis, “Efficient PC-FPGA Communication over Gigabit Ethernet,” in 2010 10th IEEE International Conference on Computer and Information Technology, pp. 1727–1734, 2010.

[26] J. Sun, R. Bittner, and K. Eguro, “FPGA side-channel receivers,” *Proc. 19th ACM/SIGDA Int. Symp. F. Program. gate arrays - FPGA ’11*, p. 267, 2011.

[27] D. Bailey, R. Hughes-Jones, and M. Kelly, “Using FPGAs to Generate Gigabit Ethernet Data Transfers and Studies of the Network Performance of DAQ Protocols,” in 2007 15th IEEE-NPSS Real-Time Conference, pp. 1–6, 2007.

[28] H. Barros, “The LAGO Collaboration: Searching for high energy GRB emissions in Latin America,” in AIP, vol. 151, pp. 151–158, 2012.

[29] K. Compton and S. Hauck, “Reconfigurable computing: a survey of systems and software,” *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, Jun. 2002.

[30] J. Fowers, G. Brown, P. Cooke, and G. Stitt, “A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications,” in Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA ’12, pp. 47–56, 2012.

[31] R. A. Burckle and V. P. WinSystems, “The Evolution of Single Board Computers,” *WinSystems Inc*, 2006.