ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Desarrollo de un entorno de análisis sistemático de algoritmos de control de congestión

(Development of a framework for systematic analysis of congestion control algorithms)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Manuel San Emeterio de la Parte

Febrero - 2020

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

1 CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por:
2 Director del TFG:
Título: “ ”
Title: “ “

Presentado a examen el día:

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre):
Secretario (Apellidos, Nombre):
Vocal (Apellidos, Nombre):

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Resumen

La primera pregunta que surge al documentarse sobre el 5G es, ¿se trata de un avance incremental de la tecnología predecesora, el 4G?, la respuesta es no. Al igual que sus antecesores la red de comunicación móvil de quinta generación representa un cambio de paradigma, esto rompe la compatibilidad con las tecnologías ofertadas por las redes anteriores. La segunda pregunta se formula desde la perspectiva de su situación actual, ¿se trata de una tecnología madura?; la respuesta nuevamente es no. El campo de desarrollo actual sobre este importante avance tecnológico contempla aun muchos aspectos "traducidos" que pertenecen a las tecnologías maduras, por lo tanto, el despliegue de esta red se encuentra sumido en una etapa de análisis y adaptación.

En este trabajo se presenta un marco de análisis de una de las principales características: se trata de la capa de transporte dentro del modelo OSI, comúnmente implementada a través del protocolo TCP. El control de congestión ofertado por TCP y desplegado actualmente sobre las redes de acceso inalámbrico presenta serias carencias para adaptarse a las variaciones de este tipo de canal. Esta limitación se verá acentuada en las redes 5G debido a la adopción de bandas de frecuencia más altas, tales como las bandas milimétricas (millimeter-waves, mmWave). Ante esta situación, recientemente han aparecido diversas soluciones tanto genéricas como específicas para situaciones concretas, sin embargo, resulta complicado comparar el rendimiento de estas de forma justa. En este trabajo se propone el desarrollo de un entorno que permita llevar a cabo esta comparativa.

Abstract

The first question that arises when documenting about 5G is, Is it an incremental advance of the predecessor technology, 4G? The answer is no; Like its predecessors, the fifth-generation mobile communication network represents a paradigm shift, this concept breaks the compatibility with the technologies offered by the previous networks. The second question is formulated from the bowels of its current deployment, is it a mature technology? The answer again is no, the current development field on this important technological advance still includes many "translated" pillars that belong to the technologies mature, therefore, the deployment of this network is mired in a stage of analysis and adaptation.

This document offers a set of intelligent and flexible solutions to one of the main characteristics, inherited from the original mobile communication networks; it is the one that has been defined in the world of telematics, thanks to the well-known OSI model, as the transport layer. The congestion control offered by TCP and currently deployed on the young fifth-generation network has failed to adapt to the challenging demands of a transmission channel that provides its transfers thanks to the use of millimetre waves. The characteristics of the deployment require a new paradigm that dorms the pillar that must support a communication that acts on highly variable channel.

Índice de contenidos

Índice de ilustraciones

Índice de Tablas

Introducción

Este capítulo conforma la referencia estructural que guía al lector sobre las diversas secciones aportadas en el documento, ilustrando un camino sobre el que realizar las consultas requeridas. En los siguientes epígrafes se da cabida al problema a resolver, así como las soluciones sugeridas y los procesos desencadenados para posibilitar la obtención y posterior análisis de los resultados del trabajo.

En primer lugar, en este capítulo se introduce el entorno en el que nos encontramos, concretando el apuntamiento sobre el que se desarrolla el trabajo. Para ello es esencial entender que la comunicación de alta velocidad es el futuro de muchos de los servicios desplegados en la actualidad, es el caso de algunos dentro del Internet of Things (**IoT)** o los basados en Mobile **Cloud Computing**. Sin embargo, la solución presentada por la última generación de redes de comunicación móvil no es actualmente una tecnología madura; esto conlleva el estudio de posibles soluciones que permitan explotar toda su capacidad, revirtiendo en el campo de desarrollo de numerosas aplicaciones que son y serán desarrolladas gracias a las ventajosas capacidades ofrecidas.

La red de comunicación móvil de quinta generación presenta problemas de rendimiento a causa de la naturaleza de su despliegue; él envió de paquetes de datos a tan altas velocidades ha de ser trasmitido sobre ondas milimétricas, **mmWave** [1], lo que supone un problema que no casa con ciertas tecnologías que han sido reutilizadas sobre el 5G y que provienen de las redes de generaciones predecesoras. Como resultado podemos afirmar que una de ellas es el control de congestión. El canal de transmisión ha sido cambiado y por lo tanto el control de congestión ofertado por TCP no está dotado de la adaptabilidad que requiere una red altamente variable. Para solventar este problema en el pasado se han propuesto multitud de soluciones de control de congestión. Sin embargo, estas soluciones suelen ser evaluadas de forma aislada o sobre escenarios muy concretos, lo que dificulta so comparativa y prever su comportamiento en términos generales. En este sentido, en este trabajo se desarrolla un entorno de evaluación sistemática de soluciones de control de congestión para facilitar su análisis automático sobre múltiples escenarios. Concretamente, el trabajo tiene las siguientes aportaciones:

- Modificación de herramientas de emulación existentes para automatizar el análisis de algoritmos sobre varios escenarios. En concreto se han explotado las herramientas MahiMahi y Pantheon que se describirán en los siguientes capítulos.
- Diseño, desarrollo y validación de funcionamiento de un entorno de recolección y representación de estadísticas de rendimiento. Este entorno hace uso de bases de datos basadas en series temporales, concretamente Influx, y el conocido entorno de representación Grafana.
- Sobre los desarrollos anteriores se ha analizado el rendimiento de 12 algoritmos de control de congestión, sobre múltiples escenarios de comunicación basados en mmWave, y se han obtenido las correspondientes estadísticas.

En este primer capítulo del documento se exponen los conceptos introducidos posteriormente sobre los distintos capítulos del documento, aclarando la referencia con los índices adjuntos en el comienzo del documento, y facilitando su búsqueda particular. En primer lugar, en el Capítulo 2 se enmarca el trabajo, aportando cierta información que facilita al lector datos sobre el comportamiento de las diversas redes desplegadas desde el inicio de la comunicación móvil**.** Este capítulo también presenta el control de congestión aportando una descripción conceptual del mismo. Además, en este Capítulo, se aporta un estudio que caracteriza los problemas percibidos en la capa de transporte desplegada sobre la configuración actual del 5G, presentando además una breve caracterización del perfil que define a los distintos protocolos de control de congestión elegidos para el

estudio. Finalmente, este capítulo concluye aportando la información que expone los problemas actuales en esta red, prestando especial atención a la tecnología **mmWave** [1].

El Capítulo 3 del documento introduce conceptos esenciales para el despliegue de las herramientas adoptadas. En primer lugar, el tercer capítulo presenta una introducción en la que se desglosa la información integrada en el despliegue de las trazas. La herramienta denominada **MahiMahi** [2], es especificada en la siguiente sección, adjuntando una breve definición de sus principales funcionalidades. El siguiente punto de este capítulo presenta una herramienta para la evaluación del control de la congestión; la solución descrita se denomina **Pantheon** [3], y representa una pieza fundamental de la solución desarrollada en el trabajo.

Este capítulo concluye con la descripción de otra herramienta que ha sido fundamental en el desarrollo del trabajo y que ha permitido un estudio sobre los resultados de rendimiento de control de congestión. Se trata de **Grafana** [4], una herramienta que permite la consulta sobre bases de datos no relacionales y orientadas a series temporales, representando los datos posteriormente en gráficos organizados y sencillos.

El Capítulo 4 del documento representa la descripción de la solución propuesta, comenzando con un epígrafe que denota la forma en la que fue desplegada la arquitectura de la solución, adjuntando por medio de ilustraciones referencia sobre el texto descriptivo que expone tanto la visión global del trabajo, como una representación de todos los módulos que hacen posible su despliegue. Este capítulo describe el funcionamiento, tanto de las herramientas desplegadas, únicamente para la consecución de las fases del desarrollo, como las herramientas que simplemente han requerido de adaptación.

En el Capítulo 5 del trabajo se muestran los resultados obtenidos, y que se representan gracias a el proceso descrito sobre los capítulos anteriores. Este capítulo no solo contempla las soluciones descritas en el módulo final del desarrollo, sino que también presenta un estudio de los resultados obtenidos a lo largo de los pasos intermedios, añadiéndolos en cómputo final para la obtención de unas métricas que permitan tomar decisiones sobre la mejor opción para la implementación de la capa de transporte.

El documento presenta además un sexto capítulo que resume el trabajo y los resultados obtenidos, así como los posibles trabajos futuros derivados de este. Finalmente, se han añadido unos anexos facilitando datos como un manual de usuario y características adicionales de las redes de comunicación móvil de quinta generación.

Marco Tecnológico

En este capítulo se realizará un breve resumen de la evolución de la tecnología y su situación actual para contextualizar el trabajo que se ha desarrollado. En primer lugar, se describirá la evolución de las redes de acceso celular hasta llegar al actual 5G. Asimismo, se presentará la situación actual de las soluciones de control de congestión. A continuación, se describirá el uso de tecnología basada en ondas milimétricas*, millimeter waves* (**mmWave** [1]), en 5G y la problemática que esto puede causar en los esquemas de control de congestión.

4.1 Evolución de las Redes de comunicación móvil

Debido a la importancia de la función de las redes de comunicación móvil en nuestro entorno, los ingenieros de Telecomunicaciones han centrado sus esfuerzos en mejorar los parámetros que actualmente definen, sobre dichas redes, la calidad de la comunicación. Varias son las generaciones que han precedido al 5G, por eso es importante, en primer lugar, contextualizar este trabajo con un breve resumen que ilustrará la importancia de esta nueva generación, su posible desarrollo y parametrización, que nos permitirá sacar el máximo provecho a sus ventajosas especificaciones.

La **Red de comunicación móvil de Primera Generación** conocida como **1G,** fue desarrollada entre los años 1970-1980, y sus primeros despliegues tuvieron lugar en el año 1979 en Japón por NTT. Asimismo, en el año 1981 se implementó el sistema de Telefonía Móvil Nórdica (NMT), en Dinamarca, Noruega, Finlandia y Suecia. Esta red solo proporcionaba servicios de voz, soportados sobre tecnología analógica con una multiplexación FDMA [5] con conmutación de circuitos sobre los 800-900 MHz, lo que nos permitía establecer enlaces para la comunicación de voz a velocidades de entre los **1kbps** a los **2,4kbps**.

No fue hasta los años 1990 cuando apareció la **Red de comunicación móvil de Segunda Generación**, **2G**, o *Golbal System for Mobile communications* (**GSM** [6]), que proporcionaba un rango de velocidades de entre **14kbps** a los **64kbps**. Se trata de una tecnología digital basada en tecnología de acceso *Time Domain Multiple Acces* (**TDMA** [7]), que funciona sobre las bandas 850-1900MHz. Gracias a este aumento en la capacidad GSM nos permite tanto servicios de voz digital como SMS, conferencias, roaming internacional, llamadas en espera, etc. Funcionalidades que aumentaron la demanda de esta red y permitieron su rápido avance hacia las siguientes generaciones de redes móviles. En torno a los años 2000-2003, apareció una generación denominada **2.5 G** o como fue definida sobre los estándares **Servicio General de Paquetes de Radio** (**GPRS** [8]). Esta nueva generación, basada en GSM, permitió, alcanzar velocidades de entorno a los **115 kbps** y gracias a una mejora denominada Enhanced Data rates for GSM Evolution (**EDGE** [9]), se obtuvieron velocidades próximas a los **384 kbps**. Gracias a esta nueva red los servicios multimedia fueron integrados en el sistema, videoconferencias, acceso al correo electrónico, búsqueda y direccionamiento, etc.

En el año 2000 nacieron los estándares de **UMTS** [10] o **Red de comunicación móvil de Tercera Generación (3G)**, que poseía como objetivo, además del propio aumento en la tasa binaria, un bajo coste. Esta tecnología está basada en conmutación de paquetes y tecnología de acceso *multipre Code Division Multiple Access* (**CDMA** [11]). **UMTS** proporciona unas tasas binarias de entre **384 kbps** a los **2 Mbps**, lo que permitió introducir nuevos servicios que facilitaron el desarrollo de las nuevas generaciones móviles. Fue en 2008 cuando la **UIT-R**, especificó los requisitos que darían soporte a la nueva generación de comunicación móvil, denominada **4G** e implementada principalmente por la tecnología celular **UMTS** [10] *Long Term Evolution, (**LTE** [12])*. Esta nueva red aporta velocidades comprendidas entre los **100 Mbps** a **1 Gbps** en el caso de que el receptor se encuentre inmóvil. Está basada en tecnologías de acceso Orthogonal *Frequency Division Multiple*

Access (**OFDMA** [5]), con anchos de banda entre los 5-20 MHz y opcionalmente los 40 MHz. El aumento de capacidad permitió el soporte de nuevos servicios como la telefonía IP, TV móvil de alta calidad, servicios para juegos, Digital Video Broadcasting (DVB), acceso móvil web, etc.

Por último, la **Red de comunicación móvil de Quinta Generación**, conocida comúnmente como **5G**, comenzó su desarrollo en 2015 y en la actualidad los investigadores siguen desarrollando nuevos métodos para conseguir explotar al máximo su rendimiento.

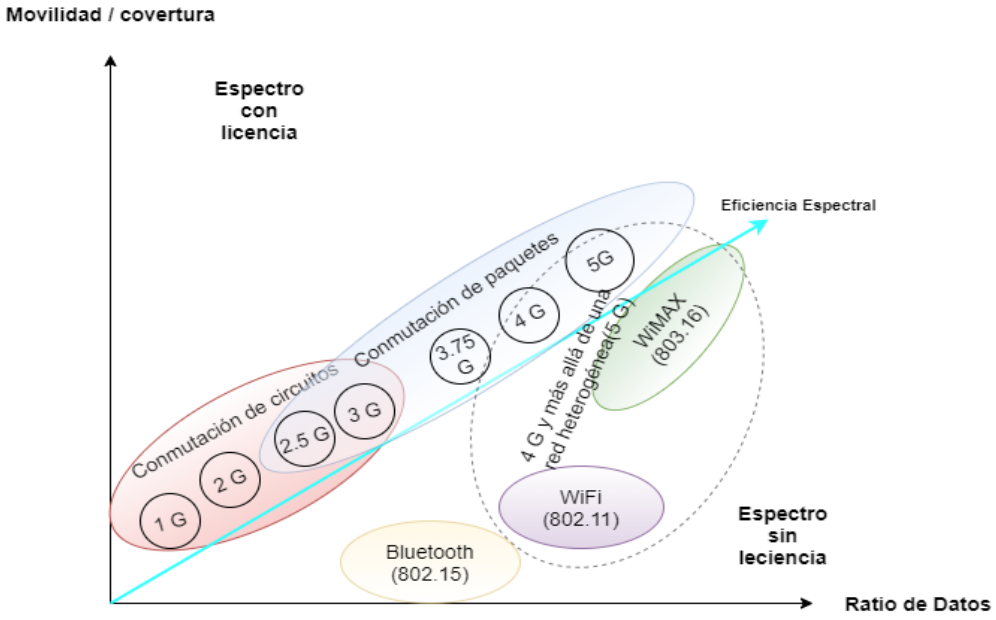

Ilustración 1:Análisis de prestaciones de las redes de comunicación móvil

Aunque se están estudiando alternativas, esta nueva tecnología continúa usando OFDMA [5] como técnica de acceso al medio. Se espera que las tecnologías 5G permitan un aumento de capacidad de hasta **100 Mbps** por usuario y una gran densidad de dispositivos conectados por unidad de superficie. Esto, a su vez, permitirá el despliegue de soluciones de *Internet of things* (**IoT**), en el que multitud de dispositivos deben estar conectados a la red con un coste energético muy reducido. Actualmente, se han definido bandas para el despliegue de 5G que van desde los 700 MHz hasta los 45 GHz en el llamado *frequency range* 2. A modo de resumen, se muestra un gráfico axial, representado sobre la **Ilustración 1**, que denota la evolución de las tecnologías de conexión inalámbricas, mostrando también cuáles de ellas funcionan sobre un espectro con licencia o cuáles en cambio utilizan el espectro sin licencia.

En la Tabla 2, se expone la información anterior en forma de tabla como medio para facilitar la comprensión y alcanzar una perspectiva que permite comparar sobre todo el límite de velocidad alcanzada por cada tecnología.

Red de comunicación móvil	Año de implementación	Velocidad Binaria	tecnologías de acceso
1 G	1980	Hasta 2.4 kbps	FDMA
2 G (GSM)	1990	Hasta 64 kbps	TDMA
2.5 G (GPRS)	2000 - 2003	Hasta 144 kbps	TDMA
3 G (UMTS)	2000	5 - 30 Mbps	CDMA
3.75 G (LTE)	2006	Hasta 100 Mbps	OFDMA
4 G (LTE-Advanced)	2008	100 Mbps - 1 Gbps	OFDMA
5 G	2015	10 Gbps – 50 Gbps	OFDMA

Tabla 2: Resumen de capacidades de las distintas redes de comunicación móvil.

4.2 Evolución del Control de congestión

En esta sección se describe la evolución de las técnicas de control de congestión. Asimismo, se plantean los potenciales problemas de estas técnicas cuando coexistan con tecnologías 5G. Se puede definir el control de congestión como una herramienta que intenta minimizar el impacto del crecimiento de la congestión de la red en determinados momentos. Esto ocurre cuando, en la red o en parte de ella, el tráfico ofrecido es superior al tráfico cursado. Esto provoca que uno o varios nodos de la red se vean afectados por un desbordamiento de trafico de paquetes, lo que desemboca en la perdida de paquetes o el bloqueo de posibles nuevas conexiones en los peores casos; retrasos en la entrega de paquetes también pueden ser debidos a congestión si los nodos intermedios poseen mucha capacidad de almacenar los paquetes que no se pueden enviar. En general, la pérdida de paquetes, o los retrasos excesivos, provoca que algunos paquetes deban ser reenviados al no recibir una confirmación por parte del receptor.

Los casos más comunes que definen el comienzo de un decremento en la tasa de envió son debidos a problemas relacionados con los siguientes aspectos: En primer lugar, la posible insuficiencia de memoria en los computadores encargados del procesamiento de paquetes, por lo que si un paquete llega al nodo, el tiempo de procesado desencadena una cola de paquetes a procesar, que termina con un decremento del ***throughput*** [13], y eventualmente la perdida de paquetes o retraso en la entrega. En segundo lugar, un fenómeno intrínsicamente relacionado con el anterior es el problema que genera tener insuficiente CPU, retrasando el procesamiento de paquetes y por lo tanto llenando el buffer de memoria con colas que no tienen tiempo de ser procesadas. Ambos fenómenos se gestionan mediante el control de flujo, en el que el receptor informa al transmisor para que este reduzca la tasa en envío. Dadas las altas prestaciones de los dispositivos actuales, la reducción de tasa debido al control de flujo no es un problema remarcable en la mayoría de los escenarios.

Finalmente, la velocidad insuficiente en el camino de datos a través de los diferentes segmentos de red provoca problemas de congestión, y es en este supuesto en el que nos centraremos. La capa de transporte implementa diferentes algoritmos para el control de congestión, modelados mediante supuestos matemáticos. A continuación, se definirá de forma breve las variables de diseño para las soluciones de control de congestión en redes, así como el control de congestión en TCP [14].

4.2.1 Bucle abierto

Esta técnica es denominada como solución pasiva [14], y trata de combatir la congestión en redes mediante un diseño adecuado de las mismas, es decir, intenta, de alguna forma, prevenir los posibles errores que aparezcan debido a su disposición, entorno y uso. Existen infinidad de variables con las que cualquier diseñador ha de lidiar cuando se enfrenta al diseño de una red, y muchas de ellas son en este entorno.

4.2.2 Bucle cerrado

Se tratan con el nombre de soluciones de bucle cerrado a aquellas que desempeñan una labor en activo para mejorar el rendimiento de la red frente a posibles problemáticas en la congestión. Muchos diseñadores de este tipo de herramientas se guían por ciertos parámetros para detectar los problemas y lo hacen mediante tres fases.

4.2.2.1 Monitorización de parámetros

En esta fase es necesario atender a los siguientes parámetros. En primer lugar, la ocupación de los enlaces y de los buffers o colas de espera de cada nodo. En segundo lugar, el porcentaje de paquetes que se descartan y por consiguiente el número de retransmisiones, retardos y jitters, medida muy importante sobre todo cuando se trata de dar servicio a videoconferencias o sincronismo de audio y video.

4.2.2.2 Reacción: envió de información en puntos de necesidad

Para la segunda fase es fundamental proporcionar paquetes especiales que no estén sometidos al control de congestión de manera que permiten saltarse colas de espera en los nodos que gracias a la monitorización hayan detectado congestión. Esto puede apoyarse en los bits de cabecera en los cuales los paquetes enviados alertan de una posible congestión. En tercer lugar, la información específica solicitada cuando se recibe un paquete que alerta de congestión.

4.2.2.3 Ajuste del sistema

Para esta fase hay tres medidas que deben cumplirse. Los nodos deben reducir la velocidad de envió para completar los paquetes en cola, controlar el acceso de forma que no se permitan más conexiones hasta liberar el nodo de su colapso y por último descartar los paquetes para conseguir controlar las ráfagas de paquetes que llegan.

4.2.3 Control de congestión en TCP

El protocolo **TCP** se trata de una solución a nivel de transporte, por lo que las entidades extremas de la comunicación (origen y destino) no establecen comunicación con los nodos intermedios. Teniendo esto en cuenta, el origen debe predecir el nivel de congestión en la red a través de parámetros medibles por él, entre los que típicamente se encuentra en *Round Trip Time* (**RTT** [15]) que es el tiempo transcurrido desde el envío de un segmento de datos hasta la recepción de su reconocimiento (Acknowledgement, ACK).

En la **Ilustración 2**, se muestra un ejemplo de la implementación básica del control de congestión en TCP. La fuente controla el flujo de datos con la finalidad de evitar la congestión y por consiguiente el bloqueo de la red. Para ello hace uso de varios conceptos importantes: **cwnd** o ventana de congestión, **SSTH** o Slow Start Threshold, y **RTO** o Retransmission Time Out. El control de congestión se basa en limitar la tasa a la que se envía la información, y se hace en función de los parámetros recibidos de la red, lo que afecta al buffer de recepción de control de flujo. La detección de grandes cantidades de flujo se advierte por medio de un proceso en el que el emisor sufre una perdida y por lo tanto reenvía el paquete hasta obtener 3 ACK duplicados o expira en temporizador RTO. Los principales algoritmos utilizados en esta técnica son **Slow Start**, **Congestion Avoidance** y **Fast Recovery.**

Para poder entender el proceso mediante el cual se desarrolla el control de congestión en TCP el documento define ciertos conceptos clave, representados sobre la **Ilustración 2**:

- La **ventana de congestión** es uno de los conceptos que resulta fundamental para el control de congestión, este parámetro define un valor limite sobre la ventana del emisor, que indica la cantidad de bytes que el emisor puede emitir sin la necesidad de esperar una confirmación (**ACK**). El objetivo de un algoritmo de control de congestión radica en una buena estimación del valor que ha de tomar la ventana de congestión en todo momento. Este parámetro ha de tomar el máximo valor posible para no afectar a la velocidad real de transmisión, sin que se traduzcan en congestión sobre la red.
- El segundo concepto clave es el **Umbral de congestión (**Slow Start Threshold**)**, este valor determina una estimación de la frontera a partir de la cual existe riesgo de congestión. El protocolo de control de congestión propuesto por TCP determina que este valor sirve como límite, cuando la ventana de congestión se encuentra por debajo de este *threshold*, se empleara el algoritmo **Slow Start**, definido a continuación, pero cuando dicha ventana de congestión supera el umbral impuesto por este campo se aplicara el algoritmo **Congestion Avoidance**.

De forma muy genérica el algoritmo utilizado por **TCP** para el cálculo de la ventana de congestión consiste en la consecución de los siguientes pasos:

- En primer lugar, el algoritmo inicia la ventana de congestión con el valor de un segmento de tamaño máximo (MMS).
- El siguiente paso consiste en el aumento progresivo del valor que toma la ventana de congestión en función del número de ACKs recibidos y los bytes reconocidos sobre ellos. Esto da lugar a un crecimiento exponencial, debido a que la ventana de congestión será el doble cada RTT.
- Cuando el valor de la ventana excede el indicado por el umbral de congestión SSTH se procede al uso del algoritmo Congestion Avoidance en el cual el crecimiento de esta es lineal en el tiempo.
- En caso de darse una pérdida, se entiende que hay congestión y la ventana se reduce siguiendo diferentes algoritmos. Posteriormente, se retoma el crecimiento que también puede seguir diferentes patrones.

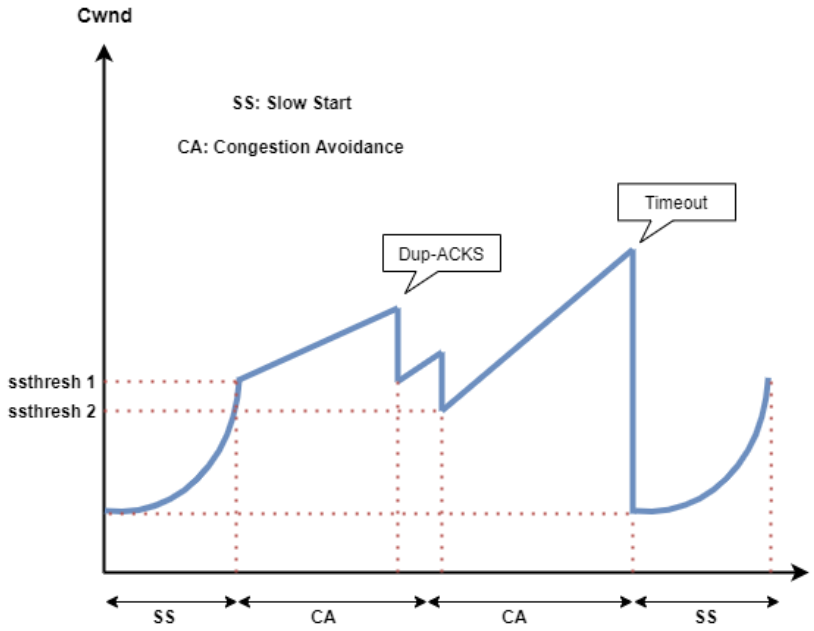

Ilustración 2: Control de congestión sobre TCP.

A partir de las técnicas tradicionales, se han desarrollado numerosas soluciones que intentan solventar los problemas que se encuentran en escenarios concretos. Entre estos escenarios se encuentran las redes de acceso inalámbrico, donde las variaciones del canal radio puede interpretarse como congestión, provocando la reducción de la tasa de envío. En este sentido, las soluciones tradicionales de control de congestión como las que están basadas en pérdidas no presentan un buen comportamiento.

Para solventar las diferentes problemáticas encontradas se han propuesto algoritmos que usan diferentes técnicas. Algunos de ellos basan sus decisiones en el retardo percibido por el transmisor, mientras otros tratan de explotar técnicas más avanzadas. A pesar de que algunos ofrecen un buen rendimiento en redes celulares, dicho comportamiento se ve enormemente afectado por el entorno, de forma que puede empeorar en ciertos escenarios altamente variables, como es el caso de las comunicaciones basadas en ondas milimétricas o *milllimeter waves* (**mmWave** [1]). Este problema se verá expuesto con mayor detalle en el siguiente apartado.

4.3 Control de congestión y redes inalámbricas (mmWave)

En este epígrafe se profundiza en la problemática del control de congestión sobre canales inalámbricos. A pesar de la aparición de nuevos algoritmos de control de congestión, las redes inalámbricas siguen definiéndose como un entorno altamente variable lo que requiere de un esfuerzo mayor para poder abordar las problemáticas añadidas por los escenarios inalámbricos, en especial **mmWave** [1].

En las redes de comunicación móvil de generaciones anteriores siempre se ha utilizado como canal de transporte el uso de ondas inalámbricas. A frecuencias más bajas la calidad de la señal es mayor, sin embargo, la alta frecuencia del canal mmWave hace que exhiba una alta absorción de oxígeno, perdidas de penetración y difracción, lo que la define con una capacidad física altamente variable.

Como se puede ver en la **Ilustración 3** el rango de frecuencias de trabajo de la red de comunicación móvil de cuarta generación se encuentra al igual que sus predecesoras en el denominado rango **sub 6 GHz**, en cambio las ondas denominadas como milimétricas trabajan en el rango de frecuencia de los 24 GHz a los 100 GHz. Es por esto por lo que permite un aumento considerable en la capacidad del canal, pero como se ha explicado anteriormente, este rango de frecuencias es muy volátil, lo que se traduce en una gran variabilidad en las métricas de retardo.

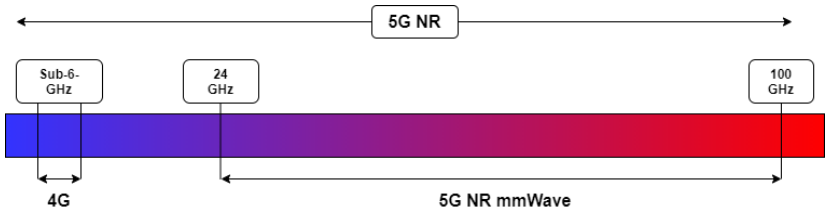

Ilustración 3: Frecuencia de comunicación mmWave

Existen diversos trabajos que han analizado el desempeño de soluciones de control de congestión sobre canales **mmWave** [1], midiendo el rendimiento de diferentes algoritmos TCP. Dichos estudios han evidenciado una mala adaptación de las soluciones de control de la congestión, provocando frecuentemente situaciones de bloqueo o fases de arranque lento. Protocolos de la capa de transporte como TCP no pueden aprovechar al completo la capacidad de la comunicación ofertada por la nueva red de comunicaciones móviles. Para afrontar de una forma más apropiada el escenario en el que nos vemos envueltos varios estudios han analizado muchos algoritmos de control de congestión incluyendo aspectos como el ancho de banda de cuello de botella, tiempo de propagación de ida y vuelta; lo que evidencia la estrecha interacción entre una configuración determinada de red y el rendimiento de la capa de transporte.

Especificaciones y restricciones

En este capítulo se ofrece una visión general que permite al lector clarificar los conceptos tanto de los escenarios desplegados como de los algoritmos de control de congestión a evaluar. Además, se aportan ciertas secciones que definen las herramientas **MahiMahi** [2] y **Pantheon** [3].

5.1 Trazas de canales mmWave

En este apartado se describen los escenarios **mmWave** [1] sobre los que se evaluarán los distintos algoritmos de control de congestión.

5.1.1 Generación de escenarios

Para la simulación de conexiones 5G **mmWave** extremo a extremo se ha hecho uso del módulo **mmWave** del simulador de redes **NS-3**. **NS-3** [16] se trata de un simulador de red de código abierto que se encuentra implementado sobre los lenguajes C++ y Python. El módulo **mmWave** en **NS-3** es una adaptación del módulo **4G-LTE** que implementa al simulador. A diferencia del módulo LTE este módulo permite realizar la conexión sobre frecuencias de espectro superiores a la banda de 6 GHz. En este trabajo se ha usado el módulo **mmWave** para generar trazas de capacidad de comunicación extremo a extremo sobre diferentes escenarios. Para ello, se han establecido comunicaciones UDP (sin control de congestión) entre entidades cliente y servidor y realizado el envío de datos saturando la capacidad del canal. De esta forma, las trazas obtenidas proporcionan el máximo de capacidad que estaría disponible para la comunicación y que serán posteriormente utilizadas para la evaluación de diferentes algoritmos de control de congestión.

Para el desarrollo de este trabajo se han considerado, en primer lugar, un conjunto de escenarios que poseen los elementos básicos que nos permitirán estudiar el verdadero comportamiento de la red móvil de quinta generación sobre factores probables que influyen en el enlace 5G real. Dichos elementos básicos son entre otros una o más estaciones base de onda milimétrica y elementos de antena para cada **Equipo de Usuario** (**UE**). En cada escenario podemos observar que los usuarios se conectan al servidor enviando paquetes de **1472 Bytes** de **MSS** cada uno con una tasa de 1 Gbps durante 3 minutos. Eventualmente las trazas son generadas usando el instante de recepción de paquetes de un usuario en la capa de transporte.

Un ejemplo de las trazas que se han utilizado para emular el escenario es la siguiente en la cual un usuario se mueve siguiendo el camino indicado por la flecha roja punteada, siendo el circulo azul de la figura, **Ilustración 4**, la ubicación de la estación base **mmWave** y las cajas grises representan los edificios de la zona.

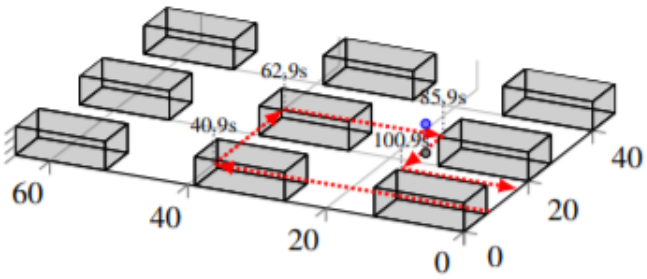

Ilustración 4: Posición mmWave

El escenario anterior aporta la siguiente grafica (**Ilustración 5**) en la que se muestra la evolución de la capacidad de comunicación, con un rendimiento medio de **989 Mbps**:

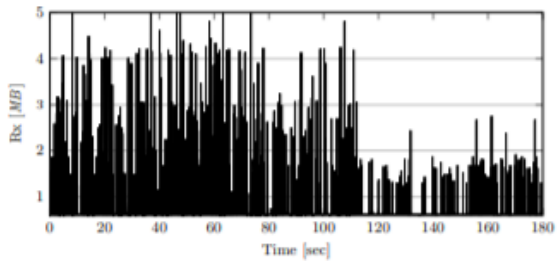

Ilustración 5: Ratio en movimiento

Estos escenarios con edificios se obtienen mediante la conexión de estaciones base **mmWave** y la construcción de obstáculos entre ellos. Los equipos de usuario de mueven para obtener conexiones **LOS** y **NLOS** a la estación base **mmWave**. En esta comunicación solo encontramos un **eNB** (Estación Base) ubicado detrás de uno de los edificios. Desde el punto de vista del UE, se posee en algunos momentos una línea de visión directa contra el **eNB** mientras que otras veces se encuentra bloqueado por los edificios.

5.1.2 Trazas

A continuación, se expone una breve descripción de los escenarios sobre los que se han obtenido las trazas que servirán para analizar el rendimiento de los diferentes algoritmos. Cabe resaltar que estas trazas definen escenarios específicos que pondrán a prueba las conexiones inalámbricas sobre ondas milimétricas definidas previamente.

5.1.2.1 Edificios 1

Este escenario simula dos edificios y una avenida del mundo real, junto con un equipo de usuario (**UE**), que cruza la avenida. En este escenario solo hay un **eNB** que se encuentra ubicado detrás de uno de los edificios (desde el punto de vista del equipo de usuario), de modo que durante parte de la simulación el equipo de usuario tendrá línea de visión directa contra la estación base, pero en algunos lugares de su ruta la visión se verá

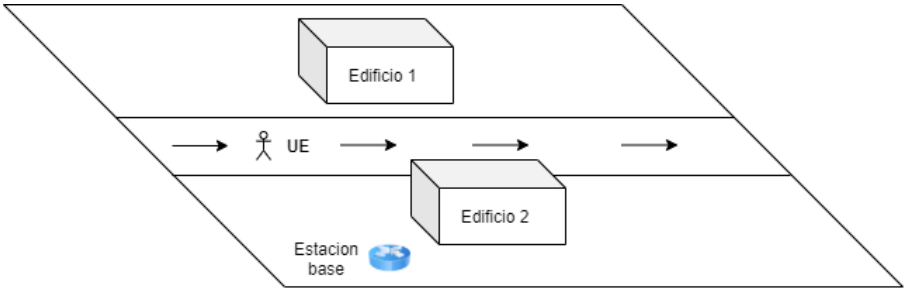

Ilustración 6: Edificios 1

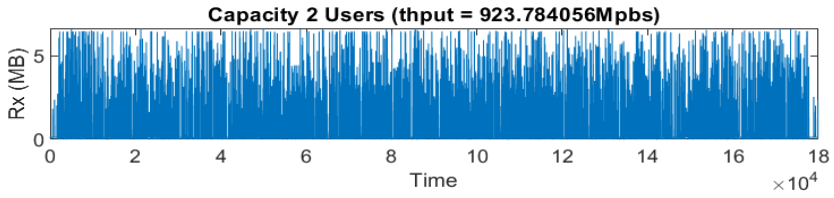

Ilustración 7: Capacidad del enlace Build1

bloqueada por los edificios definidos. A continuación, se muestra un gráfico, **Ilustración 6**, que representa el escenario. En la **Ilustración 7**, se muestran la evolución de la capacidad asociada al escenario en cuestión. En concreto se indica la cantidad de bytes recibidos a nivel de transporte a lo largo del tiempo, así como el troughput final obtenido.

5.1.2.2 Edificios 2

En este escenario se repite la misma situación que se ha definido para la traza anterior, su diferencia radica en que en este caso el equipo de usuario comenzara su ruta alejado de la estación base e ira acercándose a la misma. Este escenario nos permitirá ver el caso opuesto al anterior y en computo sabremos la distancia a la que el usuario perdió la conexión, en el caso del escenario anterior, y en este caso, la distancia a la que el usuario a de acercarse a la estación base para comenzar a recibir señal. A continuación, se muestra una **Ilustración 8**, que trata de clarificar la situación expuesta. En la siguiente **Ilustración 9**, se muestran los resultados del procesado de la traza correspondiente al escenario, los datos aportados permiten hacerse una idea del comportamiento del enlace en función de la situación del **EU**.

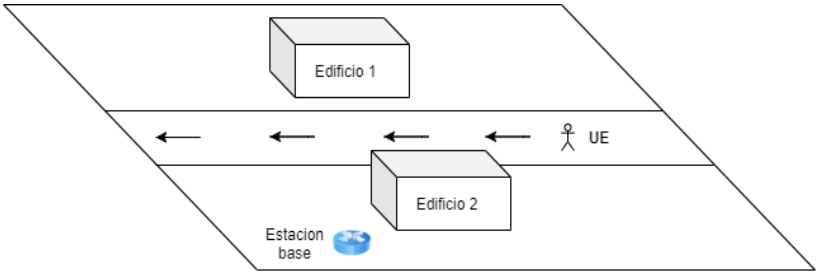

Ilustración 9: Edificios 2

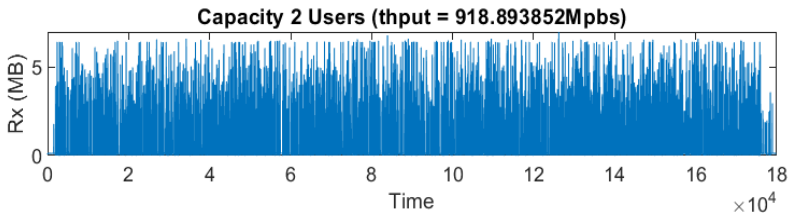

Ilustración 8: Capacidad del enlace Build 2

5.1.2.3 Edificios 3

Para el escenario Edificios 3 la implementación consta de una traza más compleja que las dos anteriores, en este caso el escenario está compuesto por nueve edificios en forma de cuadricula 3x3 con una estación base **mmWave** [1]. En este caso los usuarios se mueven por los diferentes bordes de la construcción, siguiendo un patrón, y cruzándose entre sí debajo de una estación base. Como se puede observar en la siguiente ilustración, el usuario sigue un camino definido a lo largo de las calles y entre los diferentes edificios. En este escenario, **Ilustración 10**, la estación base está montada a pocos metros de un edificio con cornisa de techo. En la siguiente **Ilustración 11**, se muestra el resultado del procesado, en este caso del escenario representado sobre la ilustración anterior.

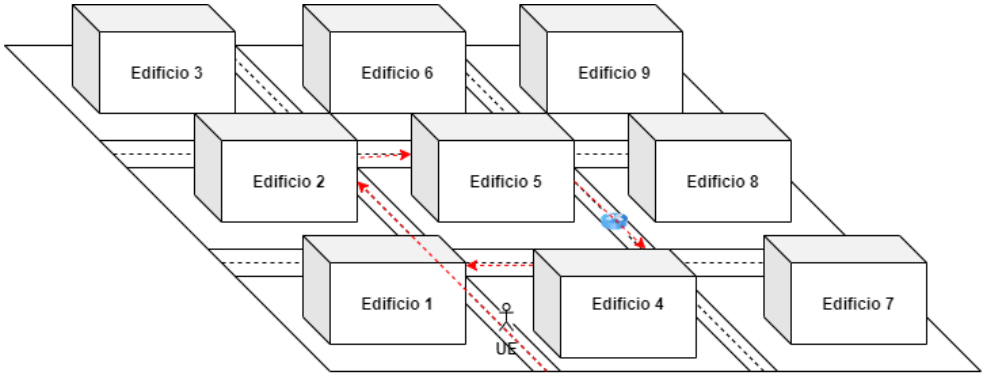

Ilustración 11: Edificios 3

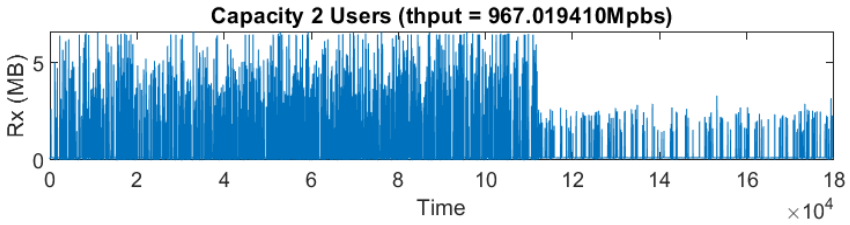

Ilustración 10: Capacidad del enlace Build 3

5.1.2.4 Edificios 4

Este escenario posee una diferencia con las anteriores distribuciones, en este caso se incluyen dos estaciones base **mmWave** [1] separadas a una distancia de 100 metros, con ocho edificios de forma y tamaños aleatorios entre ellos. Los equipos de usuario en este caso siguen su ruta atravesando los edificios, pasando cerca de sendas estaciones base y cruzándose entre sí. En el siguiente gráfico, **Ilustración 12** se muestra el escenario descrito.

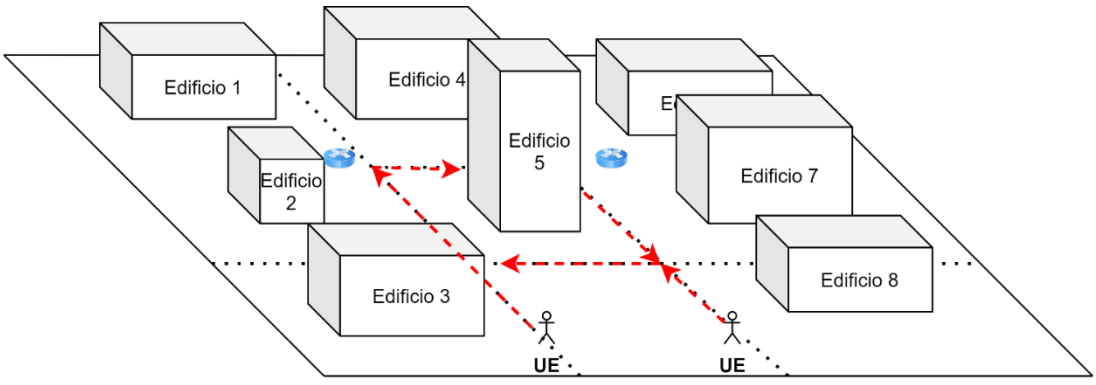

Ilustración 13: Edificios 4

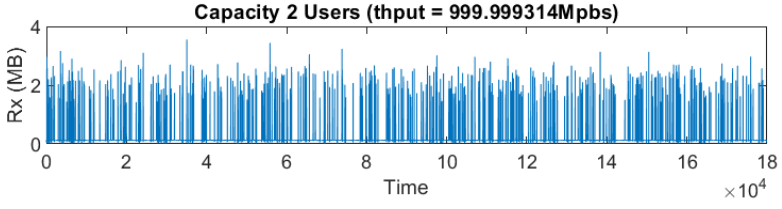

Ilustración 12: Capacidad del enlace Build4

El procesado del canal, que representa el estudio sobre esta traza viene adjunto sobre la **Ilustración 13**. El último escenario del tipo edificio, presentado como entrada del sistema no ha sido descrito, esto es debido a que su baja calidad en el canal no garantiza ninguna conexión, sin embargo, en esta sección se aportan sobre la **Ilustración 14**, los resultados de la medida de capacidad sobre este escenario.

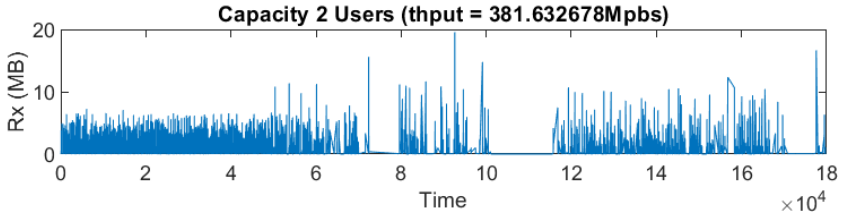

Ilustración 14: Capacidad sobre el enlace Build 5

5.1.2.5 Indor Hotspot (InH)

Las trazas de este escenario denominadas como **InHM**, están basadas en el modelo de **3GPP** [20], que caracteriza la pérdida de canal en las zonas interiores. En este tipo de escenarios es común encontrarse con objetos inmóviles, zonas con fuertes reflexiones para las señales y en los que normalmente están plagados de obstáculos a lo largo del camino. En los escenarios **InHM** los equipos de usuario se mueven en diferentes rutas aleatorias en una habitación de 30x30 m^2 a una distancia media de 10m con respecto a la estación base **mmWave** [1]. En la **Ilustración 15**, se muestra la evolución de la capacidad para uno de los escenarios.

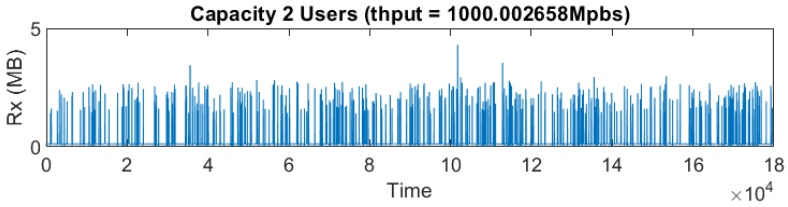

Ilustración 15: Capacidad sobre el enlace InHM

5.1.2.1 Macro-celda Rural y Urbana (RMa y UMa)

En este escenario las trazas, tanto **RMa** como **UMa**, están basadas en modelos **Macrocell Rural** y **Urban** de **3GPP** [20], respectivamente. El objetivo de estos modelos es retratar las pérdidas del canal de las zonas rurales y urbanas, lo que posee una amplia gama de alturas con respecto a las estaciones base, terminales de usuario, edificios, anchos de calles y altura ambiental efectiva. En los dos escenarios generados, es decir, **RMa** y **UMa**, los usuarios se mueven dando de un paseo aleatorio sobre un cuadrado de 30m x 30m, a 25m de la estación base.

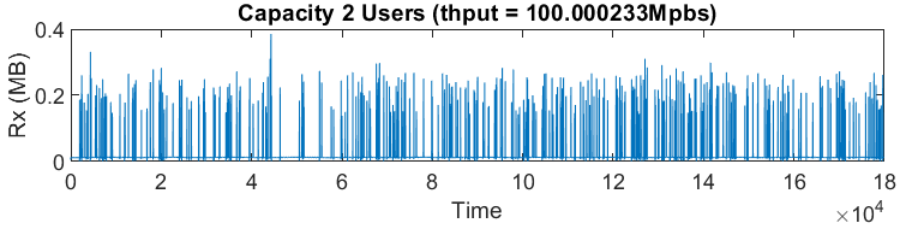

Ilustración 16: Capacidad sobre el enlace RMa

En el caso de las trazas **RMa** como ya se defino anteriormente están basadas en el modelo **Rural Macrocell** de **3GPP** [19], que retrata la perdida de canal en las zonas rurales con una altura de estación base de entre los 10m a los 50m de altura, además define una altura de usuario de entre los 1m a los 10m de altura, una altura de edificio de entre los 5m a los 50m de altura y una anchura de calle de entre los 5 a los 50 metros. En los escenarios **RMa**, los usuarios se mueven dando un paseo aleatorio sobre un cuadrado de 30m x 30m, a 25m de la estación base. **Ilustración 16** muestra la evolución de la capacidad para uno de estos escenarios.

Por otro lado, las trazas **UMa** basan su operatividad en el modelo **3GPP** [19] diseñado para entornos **Urban Macrocell** (UMa), que define unas pérdidas de canal asociadas a zonas en las cuales el despliegue de antena tuvo que realizarse sobre tejados. Las diferentes alturas a las que nos podemos encontrar las estaciones base en este tipo de despliegue oscilan entre los 10 y los 50 metros de altura, pudiendo encontrar los equipos de usuario entre los 1,5 a los 22,5 metros de altura. Además, los equipos de usuario siguen un comportamiento similar al que se define para los entornos rurales, es decir, después de moverse por un paseo aleatorio el equipo de usuario realiza su ruta dentro de un cuadrado de 30x30m, a 25m de la estación base. La **Ilustración 17** muestra la evolución de la capacidad de comunicación en uno de estos escenarios.

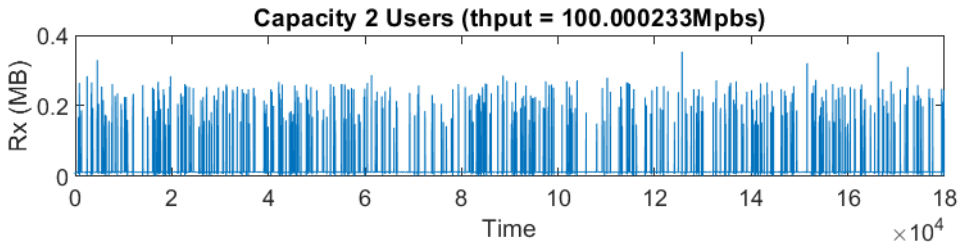

Ilustración 17: Capacidad sobre el enlace UMa

5.2 Emulador de enlace Mahimahi

Mahimahi [2] es el emulador de enlace que se usa en el entorno **Pantheon** [3], que se describirá en el siguiente apartado. Esta herramienta permite generar contenedores ligeros que mediante el uso de cierto dispositivo de red sintético permite observar los paquetes en tránsito, así como determinar un comportamiento específico para dicha comunicación (retardos, pérdidas, etc.). A continuación, se describen las herramientas de emulación proporcionadas por **Mahimahi**.

5.2.1 Link Shell

Link Shell es una herramienta de emulación de redes que utiliza como entrada ficheros de trazas. Esta será la principal herramienta utilizada en este trabajo, ya que permite utilizar las trazas generadas en los diferentes escenarios, descritos en la sección anterior, para ser usadas por **Mahimahi**. Gracias a la emulación de enlaces será posible emular redes reales de forma realista, tales como enlaces celulares o enlaces con velocidades fijas. Los enlaces direccionan los paquetes que llegan en dos colas, dependiendo de la dirección prevista, la cola Uplink o enlace ascendente y la cola DownLink o enlace descendente.

Cada fichero de entrada tiene un formato simple, donde cada línea representa una oportunidad de entrega, es decir el tiempo en el que un paquete de tamaño **MTU** puede ser entregado al destino. Concretamente, en cada línea del fichero de traza se indica el instante temporal (con granularidad de milisegundo) en el que se pueden entregar **1500 bytes** al destino. Además, es importante destacar que puede haber varias líneas con un mismo valor temporal (varias entregas de 1500 Bytes en el mismo milisegundo). También es interesante destacar que una oportunidad de entrega se desprecia cuando los bytes no están disponibles en el mismo instante en el que

se ofrece la oportunidad, funcionando además en forma de *looper*, pues cuando el fichero de entrada se acaba vuelve al principio del archivo. Gracias a otra herramienta proporcionada por Mahimahi y denominada **DelayShell**, se pueden crear enlaces más flexibles con un retardo unidireccional.

Los ficheros de salida contienen información del envío de paquetes del enlace ascendente, así como del enlace descendente. En sendos ficheros lo que podemos observar es que primero el documento aporta el comando insertado para correr ***mm-link***, información sobre la cola, el ***timeStamp*** [20] inicial en el que se empezó el enlace *mm-link* así como el *timeStamp* base o ***starting log timestamp***, y el ***mahimahi config*** o prefijo de Shell. Después de este encabezado cada línea está formada por el *timeStamp,* con un símbolo que posee ciertos significados descritos a continuación y el tamaño de paquete. Por lo tanto, cada línea representa así mismo un posible uso de oportunidad de envío. En cuanto a los símbolos usados por esta herramienta, en cada línea podemos encontrarnos un #, un +, un – o una **d**. El símbolo de # entre el *timestamp* y el packet_size representa una oportunidad de entrega no usada, el + representa que un paquete ha llegado al enlace, pero quizás sea tirado posteriormente, el – representa la entrega de un paquete de forma correcta y por último el símbolo **d** cambia el formato de tres valores habitual en las líneas formadas por los símbolos descritos anteriormente. En este caso la línea que contenga el símbolo **d** estará formada por cuatro valores, el primero es el *timestamp* y al igual que en los demás después viene el símbolo **d**, pero los dos valores que muestra a continuación son diferentes: en primer lugar, nos encontramos el número de paquetes que se tiran y en segundo lugar el número de bytes tirados. Por lo tanto, si nos encontramos con una línea que contenga el símbolo **d** sabemos que significa que uno o varios paquetes que estaban esperando en la cola han sido tirados.

LinkShell integra además una herramienta grafica que permite visualizar el proceso del uso de la red. Esta herramienta incluye gráficas de la tasa de transferencia, así como los paquetes que se encuentran en la cola por el tráfico que entra y sale de los enlaces ascendente y descendente.

5.2.2 LossShell

Esta herramienta nos permite crear un contenedor que gestiona los paquetes que se tiran, mediante una probabilidad entre 0 y 1, tanto al salir mediante el enlace ascendente, como al entrar mediante el enlace descendente. Esta herramienta por lo tanto permite evaluar algoritmos en la red como disciplinas de cola. De forma determinada, **LossShell** implementa una cola de desecho en orden **FIFO**, implementada de forma que permite el uso de CoDel, un esquema de gestión de colas activo.

5.2.3 RecordShell

Esta herramienta permite la grabación (generación de trazas) de conexiones **HTTP** [20]. Esto es posible mediante el almacenamiento de las solicitudes y respuestas correspondientes junto con la IP de cada servidor web contactado mediante el directorio dado. La seguridad aportada por esta herramienta es gracias a que utiliza un certificado **TLS** auto firmado en su proxy **HTTPS** [20]. Esto puede suponer un problema pues los navegadores web habituales pueden rechazarlo, por lo tanto, para fines de prueba o depuración **Mahimahi** permite desactivar este comportamiento.

5.2.4 ReplayShell

ReplayShell es capaz de reproducir todo el tráfico grabado mediante la herramienta anterior gracias al uso de servidores locales que emulan el uso de los servidores de aplicaciones. Esta herramienta se diferencia fácilmente de las demás aportadas por **Mahimai** ya que es la única que no posee una conexión exterior con la red, en cambio posee interfaces de red virtuales vinculadas a cada dirección IP en la que se respondieron las solicitudes respuestas por servidores y guardadas mediante el comando **RecordShell**. Esta herramienta nos permite también

medir el rendimiento de navegadores web alocados en sitios complejos a diferencia del **RecordShell** esta conserva una estructura fragmentada como la del sitio web.

5.3 Herramienta Pantheon

Una de las herramientas esenciales para el estudio de los diferentes algoritmos de control de congestión es **Pantheon**, desarrollada por la universidad de Stanford y el instituto de tecnología de Massachusetts [3], este instrumento nos permite simular tanto un cliente como un servidor que utilizan como protocolo de congestión una gran variedad de algoritmos desarrollados tanto para redes móviles de generaciones anteriores como, en este estudio, para ser implementadas sobre la red de quinta generación.

Los algoritmos desarrollados sobre la capa de transporte son fundamentales para el correcto comportamiento de las aplicaciones en redes. Pero es complicado, aun estando en un entorno académico, poder evaluar el comportamiento de nuevas ideas, así como de algoritmos reproducidos o mejorados. Gracias a esta herramienta la comunidad puede nutrirse de los beneficios de lo que los mismos creadores definen como campo de entrenamiento. Para los investigadores de redes el beneficio supone no solo contribuir a un conjunto de algoritmos de referencia y una evaluación compartida gracias al archivo de resultado publico implementado por **Pantheon**, si no que permite el desarrollo o mejora de los nuevos algoritmos.

En el pasado los esquemas de control de la congestión predominantes se desarrollaban y probaban en redes académicas, pero hoy en día es mucho más difícil generar un buen algoritmo con este tipo de pruebas pues Internet se ha vuelto mucho más diversa con el desarrollo de nuevas tecnologías, modulaciones y despliegues, por ello es necesario el uso de herramientas de investigación que permitan la emulación de un comportamiento mucho más semejante al que podemos encontrar en la red real.

Pantheon consta de cuatro partes definidas por sus mismos desarrolladores, que son las citadas a continuación [3]:

1. Una librería de software que contiene una creciente colección de protocolos de transporte y algoritmos de control de congestión, todos verificados para ser compilables y poder ser ejecutados sobre sistemas de integración continua, en la que cada uno posee una interfaz similar cara iniciar o detener un flujo a máxima potencia.

2. Un banco de pruebas diverso de nodos de red en redes inalámbricas y redes cableadas de todo el mundo, entre las que se encuentran redes en Stanford (EE. UU.), Guadalajara (México), São Paulo (Brasil), Bogotá (Colombia), Nueva Delhi (India) y Beijing (China), y redes cableadas en todas las ubicaciones anteriores, así como Londres (Reino Unido), Iowa (Estados Unidos), Tokio (Japón) y Sydney (Australia).

3. Una colección de emuladores de red, cada uno de los cuales esta calibrado para igualar el rendimiento de una ruta de red real entre dos nodos, o para capturar alguna forma de comportamiento patológico de la red.

4. Un sistema continuo de testeo que evalúa regularmente los protocolos de Pantheon sobre redes reales de Internet entre pares de nodos del banco de pruebas, a través de rutas de red parcial o totalmente cableadas y sobre todos los emuladores de redes, en escenarios de flujo único o de flujo

múltiple, y el archivo público de resultados de trazas de paquetes y análisis en ://pantheon.standford.edu.

Para su uso **Pantheon** permite tanto la evaluación de algoritmos de control de congestión entre nodos remotos, o en entornos locales mediante la emulación de enlace. En concreto, en los entornos locales los enlaces se emulan mediante **Mahimahi**. En este sentido, **Pantheon** permite el análisis sistemático de los algoritmos que posee sobre los diferentes escenarios descritos anteriormente. En este trabajo se automatizará la ejecución de Pantheon para permitir el análisis sistemático de los diferentes algoritmos que este incluye.

5.3.1 Algoritmos de control de congestión

En este apartado se describen los distintos algoritmos de control de congestión elegidos para formar parte del estudio. Antes de comenzar con la especificación de cada uno es importante resaltar que cada uno posee una cualidad semejante, que lo define en su rol de conexión, esto se refiere al campo denominado **first_run**, lo que define este parámetro es la forma en la que se realizara el saludo a tres vías para establecer la conexión cliente-servidor. Este campo está compuesto por una cadena de caracteres que definen quien será el primero en iniciar al comienzo de una conexión, es decir, las respuestas posibles son ***Receiver*** o ***Sender***, por lo tanto, será uno de estos dos los que inicie la conexión en primer lugar.

5.3.1.1 Cubic

Cubic [24] es uno de los algoritmos de control de congestión más populares en el mundo, este algoritmo funciona sobre **TCP** y es parte fundamental en productos como Windows 10 y Windows Server 2019 y es utilizado de forma predeterminada por los “kernels” o núcleos de Linux.

Es importante resaltar que **Cubic** aumenta su ventana de congestión de acuerdo con una función cubica que calcula el tiempo absoluto desde los últimos segmentos caídos en lugar de basar su cálculo en el tiempo de ida y vuelta de los segmentos (**RTT**). Esto proporciona a la infraestructura una mejor equidad entre los flujos TCP, además, este algoritmo es capaz de aumentar el tamaño de ventana de forma más agresiva cuando este lejos de alcanzar el punto de saturación medido anteriormente, regulando esta velocidad cuanto mayor sea la proximidad con dicho punto de referencia, esto permite que la red se estabilice antes de buscar más ancho de banda.

5.3.1.2 Vegas

Se trata de un algoritmo de control de congestión que introduce mecanismos avanzados, basados en la estimación de retrasos de ida y vuelta [25]. Sin embargo, es bien sabido que Vegas posee limitaciones en entornos de redes mixtas.

5.3.1.3 BBR

BBR [26] (Bottleneck Bandwidth and Round-trip propagation time) se trata de un algoritmo de control de congestión emergente, que en lugar de utilizar la perdida de paquetes como señal de congestión, como muchos otros algoritmos, utiliza una estimación del ancho de banda del enlace de cuello de botella disponible para determinar la velocidad de envío. Este algoritmo intenta proporcionar una alta utilización de los enlaces mientras trata de evitar las colas en los búferes que suponen un cuello de botella en la determinada topología.

Propuesto por un equipo de Google como nuevo algoritmo de control de congestión basado en la estimación del ancho de banda, **BBR** solo requiere modificaciones del lado del remitente, lo que facilita su implementación enormemente.

5.3.1.4 Verus

Se trata de un algoritmo de control de congestión basado en el aprendizaje de la relación entre el retraso percibido y la ventana de transmisión en las redes celulares [28]. En concreto **Verus** [28] utiliza mediciones extremo a extremo para reaccionar rápidamente al cambio repentino específicamente celulares.

5.3.1.5 Sprout

Al igual que Verus, **Sprout** [27] fue propuesto para inferir la variación del canal inalámbrico en las redes celulares a partir de los tiempos medidos entre las llegadas de los paquetes y adaptando la velocidad de transmisión. **Sprout** ha centrado sus esfuerzos en reducir el retardo autoinfligido que afecta al TCP y sus variantes para un canal altamente variable, sobre el que vamos a trabajar.

5.3.1.6 Scream

Scream [29] permite la adaptación de frecuencia con reloj automático para actividades multimedia, es decir proporciona un algoritmo que permite la adaptabilidad de la frecuencia de operación en función de las necesidades del servicio. Este algoritmo fue diseñado originalmente con el propósito de controlar la congestión en comunicaciones extremo a extremo en tiempo real, tanto sobre audio como video para **WebRTC** (**Web Real-Time Communication** [30]), usado por múltiples servicios y popularizado en la comunidad de juegos.

5.3.1.7 PCC

Para solventar las nombradas deficiencias en la familia TCP nace **PCC** [31] (*Performance oriented Congestion Control*), una nueva arquitectura de control de congestión mediante la cual cada remitente es capaz de observar continuamente el canal que porta sus acciones y el desempeño de estas empíricamente, permitiendo por lo tanto la adopción de acciones que resulten en un alto comportamiento de la red.

5.3.1.8 Vivace

Este algoritmo hace uso de optimización en línea (convexa) en el aprendizaje automático [32]. **Vivace** fue diseñado dentro del marco PCC, descrito en el apartado anterior, estableciendo un rendimiento mucho mejor que el ofertado por los algoritmos clásicos. Entre las ventajas que este esquema ofrece nos encontramos con unos buenos términos de rendimiento, referidos velocidad de convergencia, alivio del bufferbloat, así como reactividad sobre las condiciones cambiantes.

5.3.1.9 PCC_experimental

PCC-experimental [33] es un esquema de control de congestión análogo al PCC descrito anteriormente, por lo tanto, consta de una nueva arquitectura de control de congestión mediante la cual cada remitente es capaz de observar continuamente el canal que porta sus acciones y el desempeño de estas experimentado empíricamente, permitiendo por lo tanto la adopción de acciones que resulten en un alto comportamiento de la red.

5.3.1.10 Fillp

Fillp [34] es un algoritmo que explota técnicas de **machine learning** [16] permitiendo a este algoritmo anticiparse a las variaciones. Esta solución se encuentra en el marco denominado popularmente como control de congestión de alto rendimiento (HTCC), que aprovecha las técnicas de aprendizaje por refuerzo y un algoritmo de crecimiento rápido para controlar de forma adaptativa los bytes que se transportan a través de los enlaces de la red y así adaptarse a las condiciones predominantes de esta.

5.3.1.11 Fillp_sheep

Fillp_sheep [34] es una variante del algoritmo de control de congestión propuesto en el apartado anterior, que nuevamente se basa en el cálculo de ventana de congestión con técnicas de **machine learning** [16] que tratan de anticiparse al estado de la señal.

5.3.1.12 LedBat

El último algoritmo es **LedBat** [35], al igual que los algoritmos definidos anteriormente, esta categorizado como control de congestión de alto rendimiento (HTCC), y fue propuesto por BitTorrent. Se implementa sobre UDP teniendo en cuenta un escenario de cuello de botella simple, donde el nuevo flujo compite contra TCP u otros flujos de datos, y pretende asegurar la equidad del recurso compartido, así como la eficiencia de este protocolo.

5.4 Representación de Datos Influx & Grafana

En este apartado se definen las principales herramientas utilizadas para la representación de datos en el trabajo. El objetivo es proporcionar herramientas que permitan gestionar la potencialmente alta cantidad de datos generados durante el análisis de los algoritmos de control de congestión y facilitar su posterior comparativa.

En primer lugar, este epígrafe introduce, de forma breve, la herramienta **Influx** [36] con la que organizaremos nuestros resultados en bases de datos de series temporales, esto quiere decir que nuestras tablas tendrán como clave el **TimeStamp** [21] en el que los datos fueron generados, lo que nos permitirá crear un flujo de información en tiempo real. Es importante destacar que la herramienta creada en este trabajo gracias a la unión de **Influx** [36] y **Grafana** [4], no solo está centrada en el desarrollo de este problema, sino que, por el contrario, gracias a la base de datos utilizada para el trabajo podremos almacenar datos futuros acerca del comportamiento de la red y monitorearlos en tiempo real.

5.4.1 Bases de datos de series temporales InfluxDB

Para entender de la mejor forma posible el funcionamiento de **Influx** [36], es imprescindible conocer una breve definición que nos permita identificar las propiedades intrínsecas y el funcionamiento de las **Series Temporales**, es por eso por lo que este apartado comienza aportando una breve descripción de estas: las Series temporales están constituidas por una secuencia de los valores que toma una determinada variable en intervalos de tiempo equiespaciados, por lo tanto, podemos definir estas series como una secuencia de datos en tiempo discreto. Es por esto por lo que nuestro estudio se nutre de estas herramientas centradas en la recopilación de datos en tiempo real, que nos permiten dotar a nuestro sistema de una de las cualidades más importantes en todo sistema de telecomunicación, la escalabilidad y adaptabilidad.

InfluxDB es una base de datos de series de tiempo de código abierto, aunque posee componentes de código propietario, pero son totalmente opcionales y están desarrollados por **InfluxData**. *Influx* está totalmente desarrollado sobre el lenguaje de programación **Go** y esta optimizado para manejar datos de series temporales.

El lenguaje utilizado para las consultas en esta tecnología es muy similar a **SQL**. Es importante destacar que la versión de código abierto utilizada para el desarrollo de este trabajo es la versión 8, que proporciona una base de datos de serie de tiempo completo con varios servicios, incluido el núcleo *InfluxDB* que puede ejecutarse tanto de forma local como en la nube. En la siguiente imagen, **Ilustración 18**, se muestra el diagrama de flujo de la plataforma de base de datos de series temporales de código abierto, tanto el módulo denominado **InfluxDB**, como el **Kapacitor**, son indispensables para la implementación de este trabajo, dotándolo de datos en tiempo real.

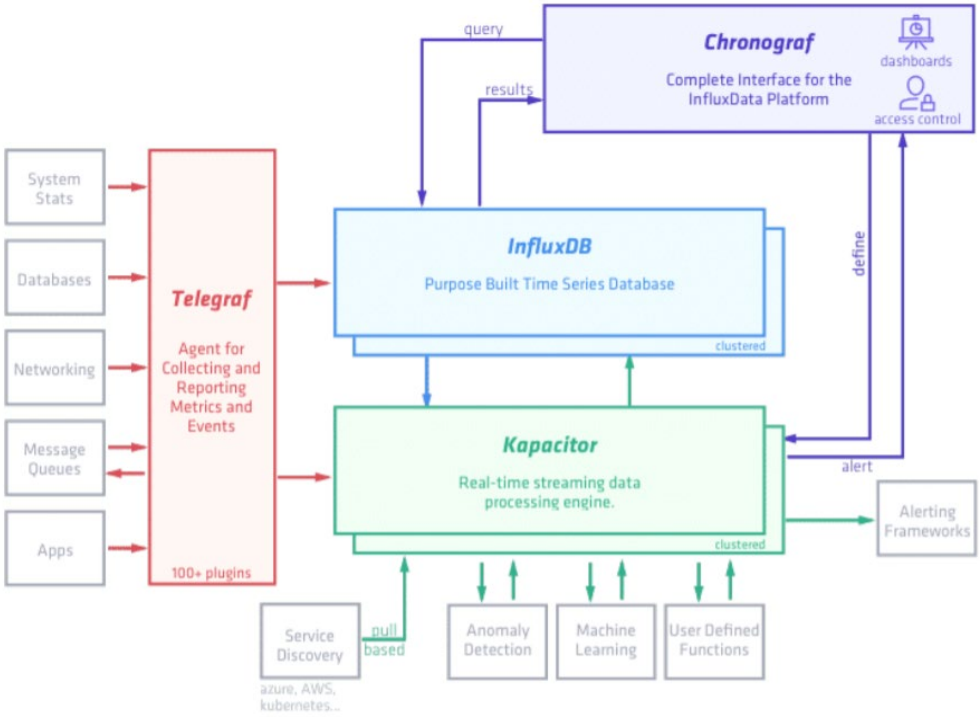

Ilustración 18: Ejemplo de flujo de datos en Influx

5.4.2 Grafana

Una vez definido **Influx** [32], túnel que nos permitirá almacenar los datos obtenidos mediante las herramientas **Mahimahi** [2] y **Pantheon** [3] en una base de datos de serie temporal, se procederá a presentar la herramienta que se utilizará para la representación y análisis de los resultados. El objetivo es simplificar la obtención de métricas de rendimiento de los diferentes algoritmos y presentar una solución uniforme y sistemática que simplifique el añadir más algoritmos de control de congestión a la comparativa.

Grafana [4] es una herramienta que permite componer paneles de control con multitud de funcionalidades. Esta herramienta permite al usuario consultar, visualizar, alertar y comprender sus métricas sin importar donde se encuentren almacenadas. En nuestro caso particular **Influx** se encargará de inyectar los datos en esta herramienta y así posteriormente podremos organizar cualquier valor de forma sencilla aportando por lo tanto una aplicación de representación de los resultados. Las principales características de **Grafana** se introducen a continuación.

5.4.2.1 Visualización

Grafana aporta una potente herramienta de visualización donde podremos representar gráficos del lado del cliente, con los parámetros de rapidez y flexibilidad requeridos por este trabajo y con multitud de opciones de despliegue. Los complementos que constituyen el panel ofrecen muchas formas diferentes de visualizar las métricas y los registros inyectados.

5.4.2.2 Paneles dinámicos

En cuanto a los Paneles, **Grafana** permite al usuario crear paneles dinámicos y reutilizables con variables de plantilla que aparecen como menús desplegables en la parte superior del panel. Esta cualidad es realmente importante en nuestro estudio pues permite que el usuario elija el dato que pretende obtener y mediante el simple uso de esta herramienta podrá mostrar los datos de un algoritmo u otro sin necesidad de crear una tabla para cada uno de ellos.

5.4.2.3 Exploración de métricas

Grafana posee además una herramienta que facilita la exploración de los datos a través de consultas ad-hoc. Gracias a esta funcionalidad el usuario puede visualizar y comparar cómo se comportan los datos en diferentes rangos de tiempo, mediante consultas y fuentes de datos que actúan en paralelo.

La herramienta aporta facilidades para el procesamiento de fuentes de datos, en este estudio es parte fundamental a la hora de la obtención de resultados a partir de las métricas que constituyen nuestra base de datos.

5.4.2.4 Exploración de registros

La exploración de registros aporta además filtros de etiquetas preservados, permitiendo por lo tanto cambiar entre la visualización de métricas y/o registros, aportando unas grandes facilidades para cualquier usuario que no tenga conocimientos en bases de datos o incluso para los mismos, permitiendo que la búsqueda sea rápida y más fiable entre todos los registros del trabajo en cuestión. **Grafana** se encuentra actualmente en desarrollo y en este apartado aportara compatibilidades extra con nuevas fuentes de datos.

5.4.2.5 Alertas

Grafana [4] permite la definición de ciertas reglas de alerta para las métricas de mayor importancia. Con esta herramienta las métricas serán evaluadas y se enviarán notificaciones continuamente a sistemas como Slack, PagerDuty, VictorOps, OpsGenie. El futuro de este trabajo pretende hacer uso de estas alertas para facilitar la monitorización de los parámetros de red en redes sensibles.

5.4.2.6 Fuentes de datos mixtas

Las fuentes de datos mixtas son una de las principales herramientas que hacen posible la simplicidad de nuestro entorno de monitoreo. La mezcla de diferentes datos en el mismo gráfico pudiendo especificar la fuente de datos por consulta. En este estudio se elaboran diferentes gráficos introducidos por secciones posteriores del documento y aportando datos en común de los diferentes escenarios superpuestos, esto da lugar a un análisis sencillo y eficaz que permite comparar las observaciones descritas en el apartado “3.1.1 Escenarios”.

Descripción de la solución desarrollada

En este capítulo se describe en detalle la solución propuesta. La siguiente sección aporta una visión sencilla y estructurada de la arquitectura de la solución, la información se subdivide en varios subcapítulos que conforman cada uno de los componentes que han sido necesarios para la obtención posterior de resultados clarificadores.

4.1 Arquitectura de la Solución

Como se define en la introducción del capítulo, esta primera sección descompone el trabajo en los diferentes módulos que lo conforman y se detallan importantes características como sus conexiones, comunicaciones, estructura, etc. En primer lugar, en este epígrafe, se aporta un diagrama que trata de clarificar la forma en la que la arquitectura es desplegada para sustentar la autonomía de trabajo y aportar datos que permitirán que en capítulos posteriores se desarrolle la comparativa entre los diferentes algoritmos.

Antes de comenzar es importante resaltar que, para la obtención de los datos que conforman el trabajo, ha sido indispensable un previo paso al desarrollo y puesta en marcha de la solución. Se trata la elección del escenario de trabajo previo al desarrollo e implementación de las aplicaciones. Numerosas aplicaciones y servicios alojados en repositorios bien sean **GitHub** [38], **BitBucket** [39], o cualquier otro entorno de reposición, fueron testeados para esta primera fase del trabajo, así mismo varios entornos de programación, herramientas de conexión remota, virtualización, etc. Finalmente se escogen las herramientas que mejor se ajustan a las necesidades del estudio, permitiendo que los resultados sean lo más clarificadores y sencillos posible. Otra de las principales características que aportan las herramientas escogidas para este desarrollo es la compatibilidad entre ellas. A lo largo de la primera fase del desarrollo el trabajo fue implementado en diversos lenguajes de programación, tales como **Bash** [40] o **Python** [41]. A medida que se incorporaban cada uno de los bloques, dichas implementaciones arrojaron por sí solas, información sobre cuál de ellas estaba más preparada para manejar los datos necesarios.

Como podemos observar en el grafico siguiente, **Ilustración 19**, la estructura del trabajo está conformada por diversos módulos que operan en conjunto para ofrecer una solución final. En los siguientes apartados se describe de forma breve cómo han sido desplegados y algunas de sus principales competidores en la mencionada primera fase de desarrollo.

En la última fase del trabajo se expondrán los resultados obtenidos mediante este estudio. Cómo se puede observar en el grafico que compone la arquitectura, a pesar de tratarse de una mera abstracción de los verdaderos componentes que forman el sistema, permite vislumbrar sus principales funcionalidades e interconexiones, por lo tanto, es importante resaltar que el módulo denominado **Equipo de gestión,** es en realidad el servicio que nace intrínsecamente del desarrollo del trabajo, es decir, **Grafana** [4]. Este aporta, no solo una plataforma local para visualizar los datos, sino que permite que cualquier usuario se registre en el servicio y pueda acceder a los datos, ordenándolos tanto por fecha como en tiempo real. Esta funcionalidad como ya se expuso anteriormente dota al estudio de escalabilidad permitiendo, por lo tanto, que futuros estudios o simples aplicaciones de monitoreo aprovechen nuestra estructura y flujo de datos, permitiéndoles migrar sus sistemas y adaptando los parámetros para su correcto funcionamiento.

En cuanto al módulo conformado por el Servidor del proyecto, aunque en la **Ilustración 19** puede intuirse como un simple almacén de datos, es el corazón del despliegue, en otras palabras, todo el trabajo desarrollado sobre el entorno de los escenarios y las trazas es desplegado en este servidor mediante el acceso remoto permitido por el módulo **MyEnTunnel** [42]. En el caso particular del despliegue, tanto el servidor de bases de datos como el servidor de **Grafana**, se encuentran alojados en la maquina local, esto facilita el despliegue en las primeras

fases de desarrollo, sin embargo, como conclusión al estudio, esta herramienta pretende ser desplegada sobre un servidor, posibilitando su acceso mediante un usuario y contraseña. En la **ilustración 19**, se expone la arquitectura que sostiene el trabajo expuesto sobre el documento, los diversos módulos que conforman esta arquitectura global son descritos, tanto sobre este capítulo como sobre el apartado de "**Anexos**", en el cual se incluye un manual de usuario.

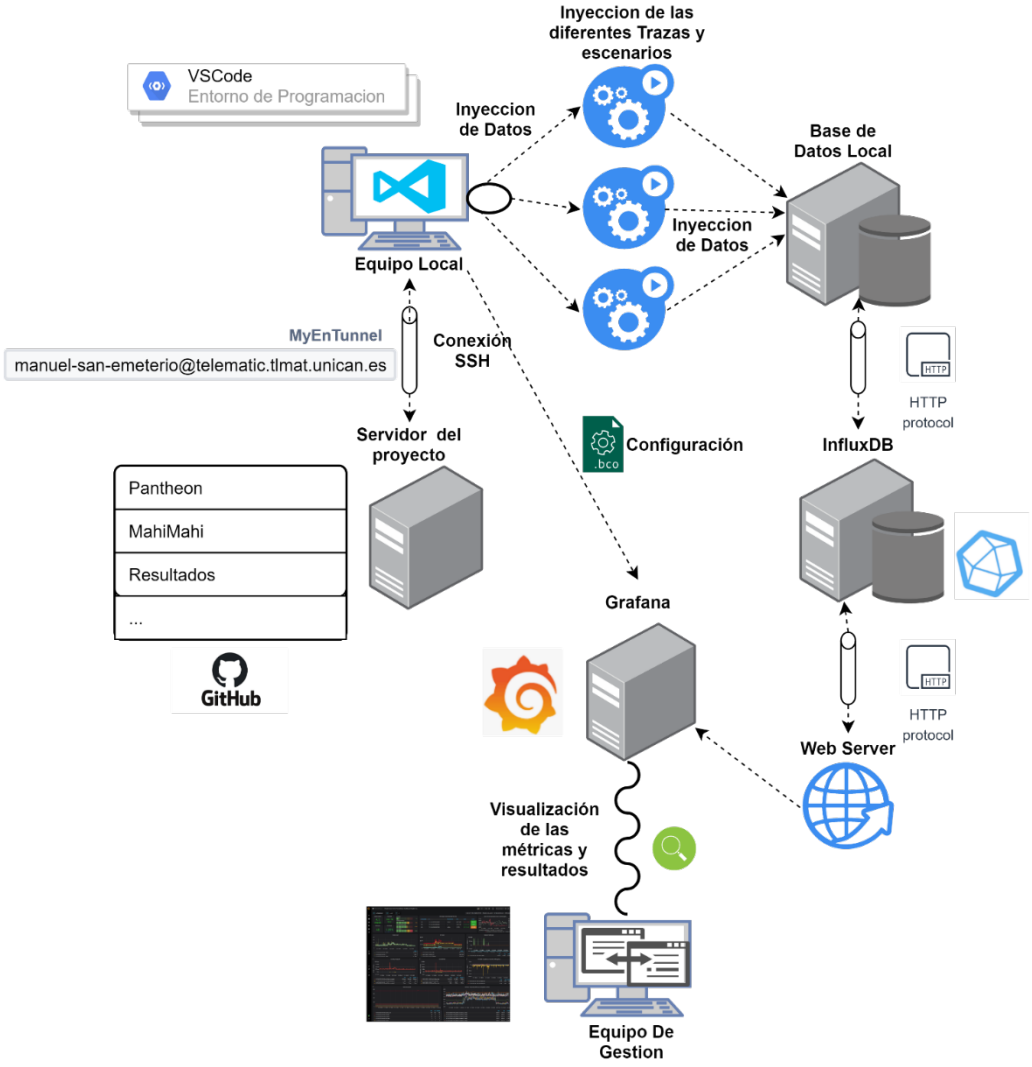

Ilustración 19: Arquitectura del sistema

4.1.2 Modulo de Repositorios

En este apartado se desarrollan las diferentes herramientas que posibilitan el estudio, así mismo se introduce como gracias a entornos de reposición se facilita su clonado.

En primer lugar, el entorno de reposición utilizado por la imprescindible herramienta **Pantheon** [16], se encuentra desarrollado sobre GitHub [3]. Este repositorio posee documentación que resulta imprescindible para el desarrollo del proceso de despliegue local. Dentro de él se aporta información relevante acerca del contenido del repositorio y datos acerca de la preparación de este. Es importante destacar que **Pantheon**, contiene "*Wrappers*" o envoltorios que caracterizan numerosos esquemas populares de control de congestión, tanto prácticos como algunos en desarrollo, todos ellos fueron descritos en el apartado "**3.3.1 Algoritmos de Control de Congestión**". Gracias al despliegue de esta herramienta podremos acceder al código de numerosas métricas

de control de congestión, lo que convierte a **Pantheon** en la herramienta idónea sobre la que desarrollar la evaluación sistemática de los algoritmos. Este primer repositorio aporta además una herramienta que, junto con el despliegue de los algoritmos de control de congestión sobre interfaz común, permite testear y comparar los diferentes rendimientos. En este trabajo se ejecutan las pruebas de forma local, para ello será necesario el uso de enlaces emulados explotando la herramienta **Mahimahi**. El repositorio de **Pantheon** aporta además cierta información sobre estudios que se realizaron anteriormente, lo que permite esclarecer el tipo de análisis que se efectuaron en el albor de esta herramienta.

En segundo lugar, para el desarrollo de este trabajo ha sido imprescindible el uso de la herramienta **MahiMahi** [18]. Como fue descrito en el apartado "**3.2 Mahimahi emulador de Efectos de Red**", Mahimahi conforma el túnel de comunicación sobre el que posteriormente montaremos la comunicación bidireccional sustentada por los protocolos de control de congestión definidos en el apartado "**3.3.1 Algoritmos de Control de Congestión**". El uso de las diferentes funcionalidades administradas por **MahiMahi**, ha sido modificado, al igual que **Pantheon**, de forma que se ajusten lo más posible al entorno de trabajo. En la **Ilustración 20**, se adjunta un sencillo esquema que permite clarificar cómo cada una de estas herramientas componen el entorno emulado.

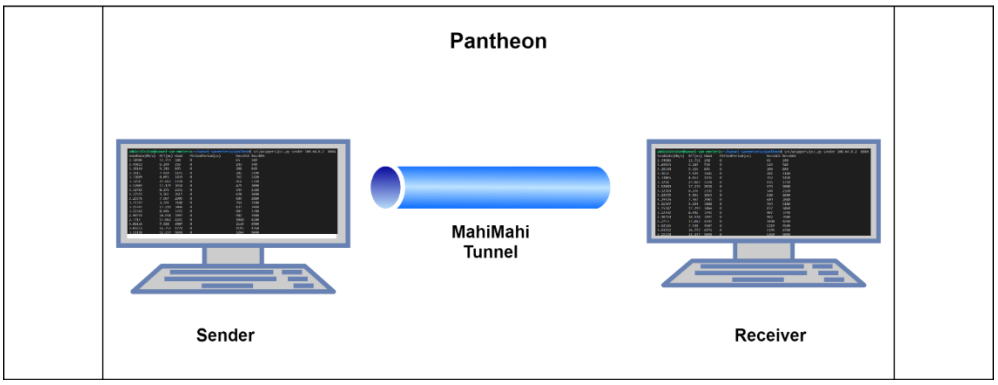

Ilustración 20: Arquitectura MahiMahi + Pantheon

Como se puede apreciar en el gráfico, ambos equipos, **sender** y **receiver** envían paquetes de datos sobre el enlace construido con **MahiMahi**. Este enlace, a su vez, se rige por un comportamiento especificado previamente, mediante el comando de inicialización, en el cual las trazas definidas servirán como patrón de envío. Las diferentes partes que computan en la comunicación se encuentran desplegadas sobre el servidor remoto de la **Universidad de Cantabria**, por lo tanto, el resultado proporcionado ha de ser trasladado posteriormente a la maquina local.

El número de casuísticas en este estudio viene determinado tanto por las trazas que dan entrada al sistema, como por el conjunto de algoritmos escogidos finalmente como paquete de soluciones. De este modo cada vez que un nuevo túnel **MahiMahi** se abre, comienza simultáneamente la grabación del canal, indispensable para el posterior estudio. Esto, por lo tanto, ha de repetirse una vez por cada vez que se cambie de algoritmo, escenario, delay, o cualquier otro parámetro regulado sobre el enlace, lo que implica no solo desplegar un túnel cada vez que se requiera de una nueva configuración de canal, si no que el túnel deberá cerrarse y abrirse también cuando un nuevo algoritmo valla a ser montado, esto es de crucial importancia y deriva en un trabajo ordenado y listo para evaluarse y así obtener conclusiones que faciliten el despliegue de la nueva generación en un entorno concorde al escenario novedoso en el que nos encontramos. En el siguiente capítulo "**5 Resultados**", se definirá el proceso de obtención de métricas y resultados mediante herramientas y fórmulas matemáticas con el fin de clarificar el método empleado.

4.1.4 Inyección y tratamiento de Datos

El módulo de inyección y tratamiento de datos podría definirse como el segundo proceso necesario para la obtención de unos datos específicos, dinámicos y realistas, que permitan estudiar en detalle el propósito de este trabajo. Este apartado estará a su vez dividido en dos partes, la primera define la arquitectura utilizada para desplegar una herramienta capaz de automatizar el proceso de lectura de todos y cada uno de los ficheros CSV que conforman el resultado del trabajo realizado en el módulo anterior "**4.1.4 Entorno de desarrollo**". Esta sección aporta además en la **Ilustración 21** un esquema del despliegue total de la sección. La segunda parte define el proceso de representación gráfica de la información extraída, en esta subdivisión documenta de forma muy breve el proceso de almacenamiento de datos y su consulta, todo ello gracias al uso de bases de datos no relacionales orientadas a series temporales.

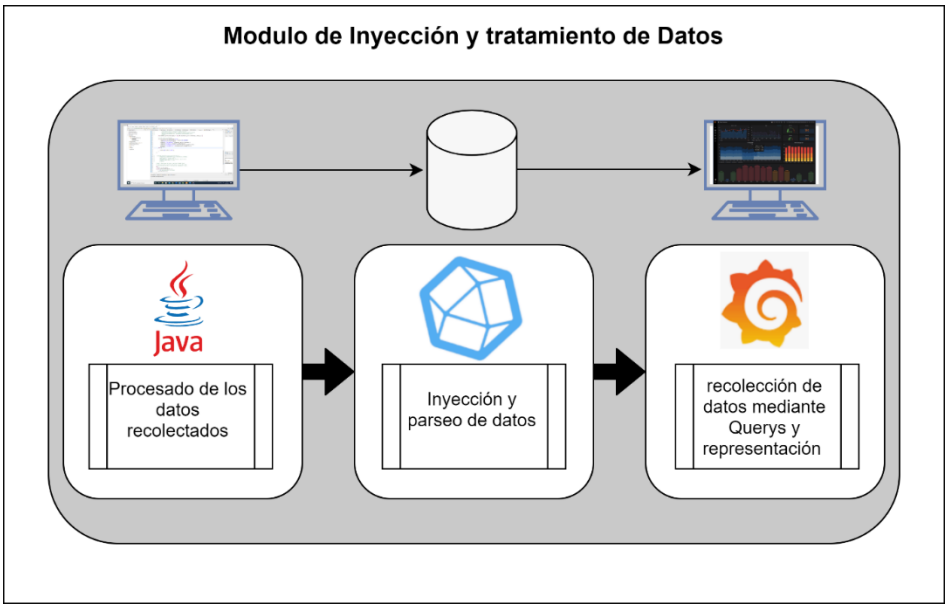

Ilustración 21: Modulo de Inyección y Representación de Datos

6.1.4.1 Procesamiento de Datos

El primer proceso descrito para el desarrollo de las labores asociadas a este módulo está formado por el submódulo de procesado de datos. Como se puede apreciar en la **Ilustración 21**, la herramienta creada para el proceso de los datos recolectados en forma de CSV por los módulos anteriores, se ha desplegado en lenguaje Java [24], concretamente gracias al entorno de desarrollo **Eclipse** [25], en él se ha desplegado un proyecto **Maven** [26] que permite el desarrollo de este código.

Una vez descritas las principales herramientas necesarias para el desarrollo de esta fase del trabajo es hora de introducir una breve descripción sobre el proceso. En primer lugar, mediante la herramienta **Xftp 6** [42] recolectaremos todos los datos hallados gracias a los procesos de automatización generados en el repositorio del trabajo. Una vez importados al entorno local es la hora de añadirlos como recurso en nuestro trabajo Maven.

Con todos los documentos CSV ligados al trabajo es el momento de procesar los datos y obtener cierto conocimiento acerca del comportamiento de estos. A continuación, se añade la **Ilustración 22**, en ella se aporta la captura de un pequeño tramo que introduce uno de los archivos de resultado.

	A	B	C	D
1	# mahimahi mm-link (Downlink) [/home/administrator/tra			
2	# command line: 'mm-link' '/home/administrator/traces/E			
3	# queue: infinite			
4	# init timestamp: 1609089286307			
5	# base timestamp: 210			
6	211 # 1504			
7	211 # 1504			
8	211 # 1504			
9	211 # 1504			
10	211 # 1504			
11	211 # 1504			

Ilustración 22: Downlink Cubic Escenario 1

En la **Ilustración 22**, se puede apreciar como cada captura en formato CSV aporta 5 líneas introducidas por el carácter **#**, este carácter al principio de la línea delimita las especificaciones del túnel **MahiMahi** [18], así como el comando mediante el que se desencadeno la captura y configuración del túnel. La cuarta línea del fichero aporta un dato crucial en el cálculo final de los resultados, por último, en la 5 línea del fichero se especifica el ***timestamp*** que conformara la base o comienzo de la cuenta durante el intercambio de tramas. Una vez descrito el encabezado de los documentos CSV, es hora de explicar los diferentes datos expresados en las líneas que continúan los ficheros. Para ello en la **Tabla 2** se especifica el significado de los diferentes caracteres que podemos encontrar en las líneas de resultado.

Carácter	Ejemplo	Significado
#	211 # 1504	El primer valor especifica el ***timestamp*** de la oportunidad de transmisión, el carácter # implica que la oportunidad de transmisión no ha sido utilizada. El ultimo valor presente en la línea simboliza el número de bytes transportables en el socket.
-	245 - 1504	El comienzo de esta línea al igual que para los otros tipos indica el ***timestamp*** en el que se desarrolla esta oportunidad de transmisión. El carácter – en esta ocasión simboliza que un paquete ha sido deliberado.
+	1145 + 1504	En el caso de que la línea incluya el carácter + significa que un paquete, en este caso de 1504 bytes, ha llegado al enlace y espera para ser deliberado, por lo tanto, está en cola, limitando el buffer.
d	22056 d 2 450	En el caso de incluir el carácter d, la línea muestra que ciertos paquetes han sido descartados, probablemente porque el buffer está lleno. El número que sucede al carácter d simboliza el número de paquetes tirados, y el último número indica el número de bytes en computo de los paquetes que han sido descartados en esta oportunidad de intercambio.

Tabla 1:Comandos CSVs

6.1.4.2 Representación Grafica

Para la representación automática de resultados se ha usado la herramienta **Grafana** [7], expuesta anteriormente en el subcapítulo "**3.4.2 Grafana**". La forma en la que se recolectan los datos con esta herramienta es gracias a ***Querys*** o peticiones que realizadas desde el entorno de Grafana, permiten la obtención de los resultados y su posterior representación gráfica. Los resultados se encuentran almacenados después de su procesado, en forma de puntos de medida sobre la base de datos**.** El proceso se ha hecho de forma meticulosa mediante un estudio

previo de los datos obtenidos y su relevancia, este análisis previo posibilita su representación y garantiza un resultado dinámico en el que los conjuntos de valores presentados conforman diferentes graficas a esto se le suma un entorno de usuario sencillo y ágil en su consulta desarrollado sobre **Grafana**.

Para este estudio es imprescindible entender que el trabajo ha sido desarrollado sobre bases de datos no relacionales orientadas a series de tiempo, lo que quiere decir que uno de los *tags*, o etiquetas que el propio sistema entiende y predefine como tal es la marca temporal, esto facilita enormemente la parametrización temporal que posteriormente se utilizara sobre el entorno de representación.

```
63          // Write points to InfluxDB.
64          influxDB.write(Point.measurement("esc01")
65              .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
66              .tag("algoritmo", algoritmo[i])
67              .addField("firstTimeStamp", (DataReader.getFirstTimestamp()))
68              .addField("Throughput", DataReader.getThroughput())
69              .addField("percentageOfArrived", DataReader.getPercetageOfArrived())
70              .addField("PercentageOfLoss", DataReader.getPercentageOfLoss())
71              .build());
72          }else {
73
74              System.out.println("ERROR");
75          }
76      }
```

Ilustración 23: Measurement points

En la **Ilustración 23**, se adjunta un breve trozo del codigo que permite la inyeccion de los resultados, en el se puede distingir los demas campos que conformaran las tablas dentro de la base de datos. El primer valor añadido a la tabla y definido sobre el código como un campo del tipo etiqueta, es el nombre del algoritmo que se esta inyectando, ademas se añaden los valores del primer timestamp en el que se inicio la comunicación, asi como la tasa de envio, el porcentage de perdida y su inverso, y el porcentage de llegada. En este último solo toman partido los paquetes que hayan sido correctamente enviados por el enlace y no los que se quedan en el buffer pero finalmente se descartan. Todos los campos descritos previamente formaran parte de las diferentes tablas que conforman el resultado del estudio, pero solo aquellos campos que esten definidos bajo el tipo *tag*, permitiran su busqueda mediante comandos propios de bases de datos.

Los resultados introducidos en la base de datos son estructurados en diferentes tablas, cada una de ellas encierra el comportamiento de todos los algoritmos del estudio sobre un escenario concreto, por lo tanto el estudio tiene tantas tablas como escenarios, y dentro de ellas se podran ontener los diferentes paramentros descritos anteriormente para cada uno de los protocolos definidos en el aparatado "**3.3.1 Algortimos de Control de congestión**". Cabe destacar que gracias a esta organización el tratamiento posterior de los datos permite su representacion cruzada, es decir, podremos acceder a los datos de un solo algoritmo sobre todos los escenarios desplegados, asi como todos los algoritmos sobre un mismo escenario, o simplemente un conjunto de ellos, podremos elegir además qué datos queremos representar, y todo ello se hará de forma dinamica y en tiempo real, lo que facilita enormemente la tarea de estudio, que conforma ademas el último paso para el completo desarrollo del trabajo.

6.2 Casos de Uso

En este apartado se exponen ciertos gráficos que procuran facilitar el entendimiento del flujo que toman los diferentes procesos desplegados para llegar a la obtención de los resultados, además se aportan breves descripciones que esclarecen las tareas que han de ir completándose para el paso de fase. Cabe destacar que en este subcapítulo se expone toda la información de forma estandarizada y mediante el uso de diagramas de flujo **UML** [30]. Estos diagramas pretender dar una aclaración visual del flujo de los procesos, sin lugar a ambigüedad, por lo tanto, intentan ser lo más precisos posibles y la descripción aportada previamente esclarece aún más dicho proceso.

En la **Ilustración 24** se describen los distintos procesos, que han de ir desplegándose a lo largo del desarrollo, desde la obtención de las trazas que conforman la entrada del estudio, hasta llegar a su obtención sobre el repositorio local que más tarde desembocara en la **Fase 2**, correspondiente con la **Ilustración 25**. El proceso de obtención de resultados comienza con la entrada al sistema de las diferentes trazas que conforman los diversos escenarios en los que tomara lugar nuestra simulación. La puesta en marcha del túnel **MahiMahi** resulta un punto crítico en este flujo, muchos de los procesos que siguen en el desarrollo han de volver a este punto retroalimentando la aplicación en el caso del fracaso en su operatividad. En caso de que el proceso de creación de dicho túnel resulte satisfactorio, el terminal sobre el que fue desplegado poseerá las capacidades delimitadas por este, y habrá llegado el momento de escoger uno de los algoritmos del proyecto. Posteriormente un comando devuelve un importante parametro en la configuracion de la comunicación sobre la metrica escogida, se trata del comando **run_first**, que devolverá al sistema una de las dos posibilidades de despliegue implementadas en los algoritmos. Como se introdujo anteriormente si los procesos resultasen fallidos en la **Ilustración 24**, se muestra el flujo que llevará al sistema al punto de partida; por otro lado, en caso de que el analisis resulte efectivo, al cabo de un tiempo delimitado anteriormente la conexión se cerrara y se pasara al proceso definido en el diagrama como reconocimiento de la captura. Una vez esclarecido este flujo solo quedaría la descarga de los archivos a un entorno local mediante la herramienta definida en el comienzo de este capitulo y denominada **Xftp 6** [14], este ultimo paso en el flujo conformara por lo tanto la entrada en la **Fase 2**. Todos los procesos que conforman este flujo han sido automatizados gracias al desarrollo de una herramienta de script, definida sobre el lenguaje **Python**, esto permite el uso paralelo del tunel contra el servidor, llevando la totalidad de estos procesos a background, y facilitando por lo tanto la obtencion de resultados, que en este caso resulta ademas imprescindible, dado inmenso volumen de los datos, esta herramienta de automatizacion ademas se encarga de definir un bucle que lleva a cabo el flujo definido de forma iterativa hasta concluir con todas las combinaciones posibles entre las trazas y los algoritmos a evaluar.

El documento continúa aportando otro diagrama de flujo, que en este caso representa el flujo de la información en la **Fase 2** del despliegue. Esta fase, como se ha descrito anteriormente, parte de los datos extraídos en la **Fase 1**, gracias a los cuales como se muestra en la **Ilustración 25**, se despliega el procesado. Para entender el funcionamiento de este diagrama de flujo es imprescindible tener claros los conceptos recogidos en la **Tabla 2**, ya que definen las diferentes casuísticas en las que nos encontramos a la hora de recorrer los ficheros de resultados. En esta fase, el flujo comienza con la importación de los CSVs extraídos por el último proceso de la **Fase 1**. Una vez importados los ficheros la herramienta desarrollada seguirá los pasos especificados en la **Ilustración 25**, el primero de ellos es la lectura de los datos de entrada. Esta lectura se llevará a cabo línea por

línea, comprobando el contenido de cada una de ellas y llevando por tanto el estado de ejecución a la necesidad de distinción sobre los diferentes tipos de línea y su significado.

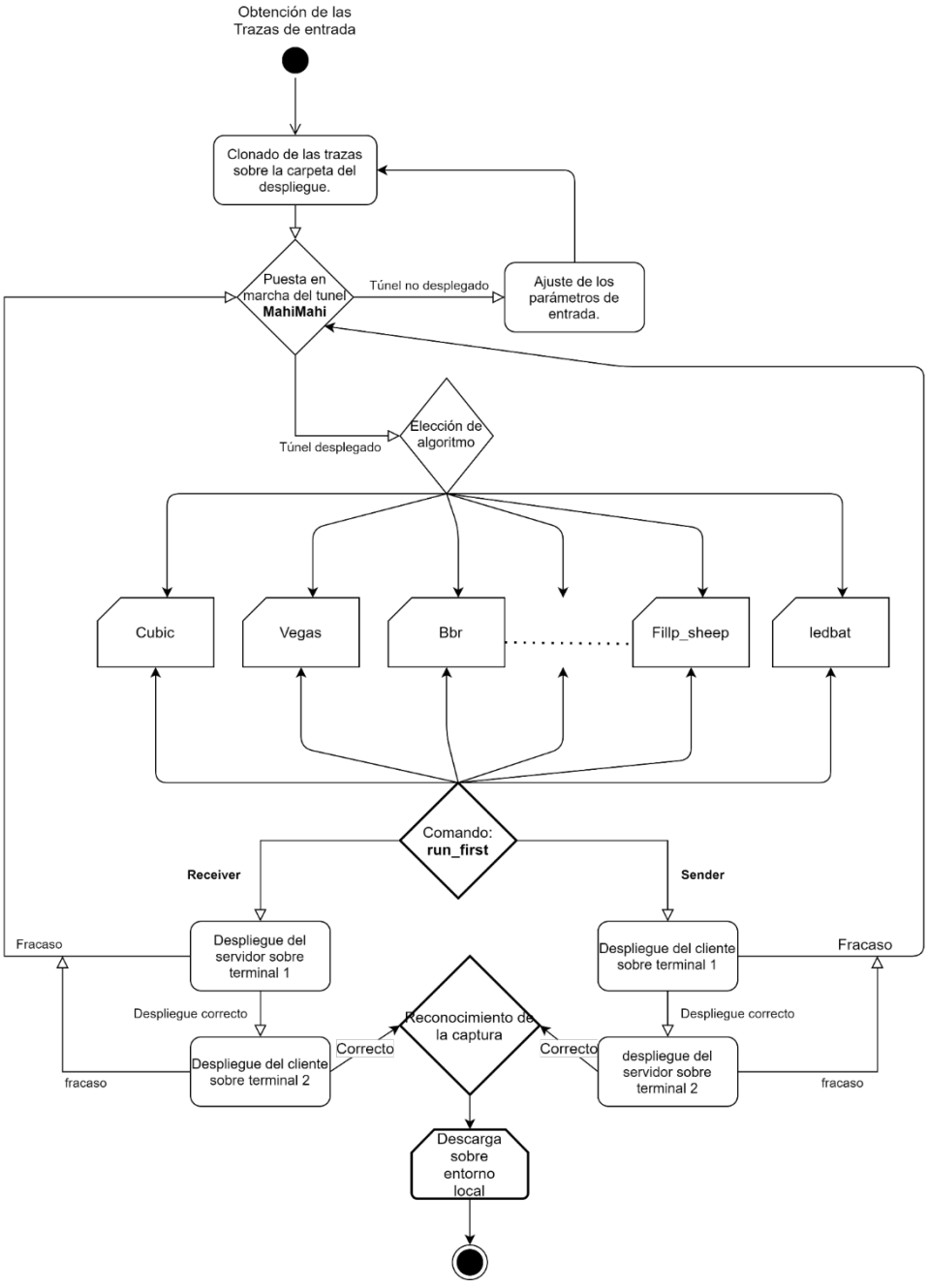

Ilustración 24: UML Fase 1

Como puede apreciarse en la imagen, cada línea será diferenciada en función del carácter que la defina, esto quiere decir que el programa ira documentando los resultados finales de forma dinámica e iterativa. Cualquiera de los procesos definidos puede llevar al inicio de la lectura en caso de percibir un error, o una línea que no contuviese ninguno de los caracteres remarcados en la **Tabla 2**, por lo que es de vital importancia realizar un depurado del fichero en caso de obtención de errores en la lectura. Una vez caracterizada una línea, el código

incrementará un contador que le llevará a la lectura de la siguiente línea del documento. Este proceso se repite continuamente hasta llegar al final del fichero, una vez llegados a este punto el código contrastará la información y en el caso de resultar según lo esperado tomará la decisión de incrementar un contador que llevará al programa a la lectura del siguiente CSV de resultados. Como es de esperar el programa realiza esta toma de decisiones en función de las cadenas definidas en la cabecera de este, y que le permiten saber cuál es la totalidad de los ficheros que han de ser analizados en el trabajo.

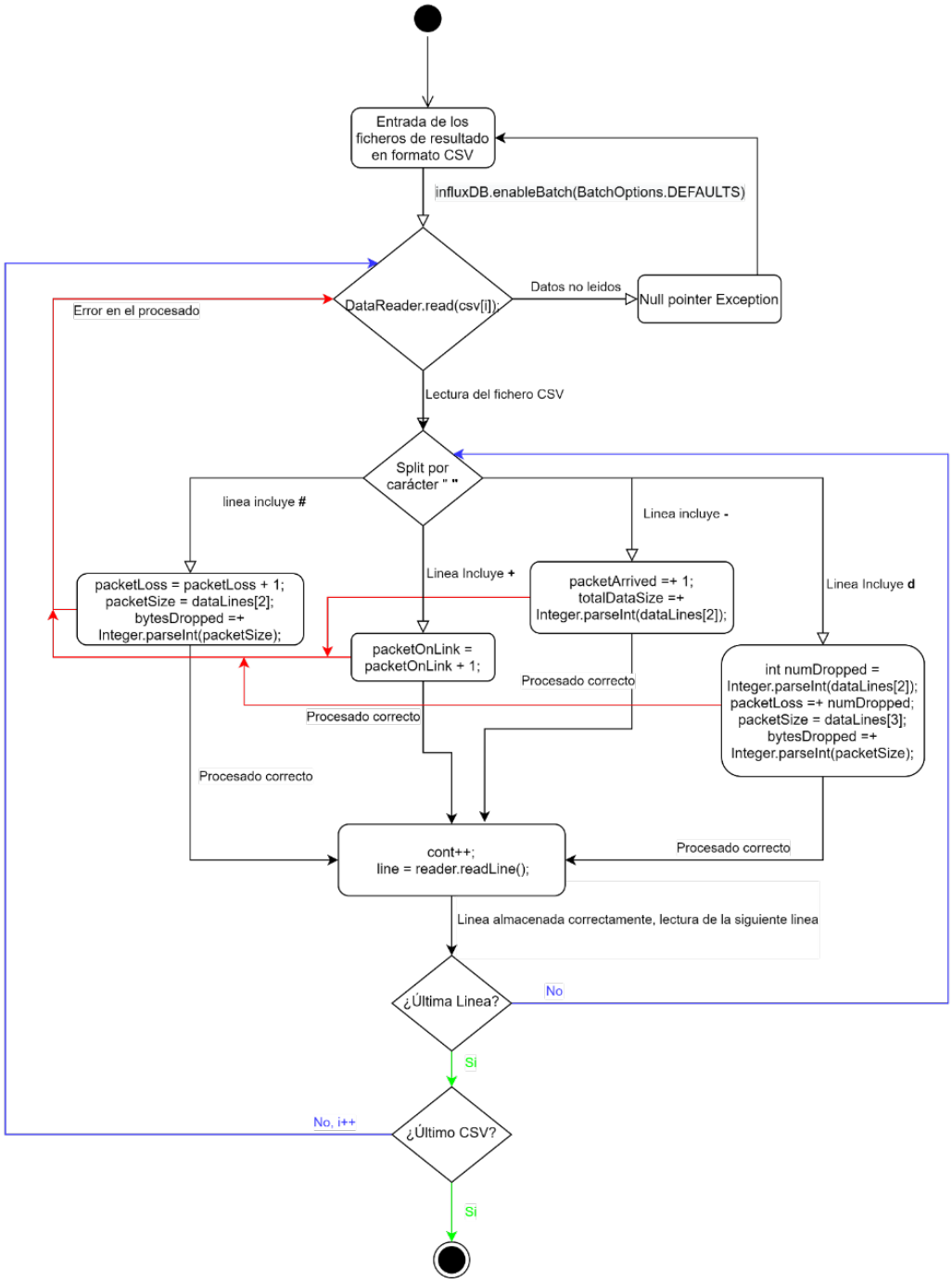

Ilustración 25: UML Fase 2

La herramienta almacenará los resultados obtenidos en el procesado de cada uno de los ficheros, como puntos de medida sobre la base de datos desplegada en **InfluxDB**. Para el desarrollo tanto de esta fase como de la anterior ha sido imprescindible la automatización debido a la inmensa cantidad de datos a procesar.

Por último, cabe destacar que si el flujo definido en la **Ilustración 24**, ha requerido de varios días para su extracción, en esta **Fase 2**, el tiempo de procesado es aún mayor,

La última fase definida en este apartado está constituida por los diversos procesos que han de ser desplegados, desde la obtención de todas las tablas sobre la base de datos, hasta su visualización final sobre el entorno grafico definido en capítulos anteriores y denominado **Grafana**.

En la **Ilustración 26**, se puede apreciar, a grandes rasgos, el flujo que seguirán los datos para su posible representación en la denominada **Fase 3**. El punto de partida en este caso son las diferentes tablas definidas gracias al uso de bases de datos no relacionales y orientadas a series temporales. La herramienta de representación nos permite el uso de **Querys**, en ellas se incorporarán diferentes **expresiones regulares** [31] gracias a las cuales la representación posterior toma un carácter dinámico y totalmente adaptable por el usuario, además facilita enormemente su visualización, sin necesidad de que el observador posea conocimientos en consultas sobre bases de datos. El entorno de representación gráfica se encarga de correr el código definido previamente en *background*, permitiendo al usuario moverse por los diferentes paneles.

El flujo representado en **Ilustración 26**, comienza con el despliegue de la herramienta **Grafana**, esta herramienta será desplegada en el entorno local, es decir sobre **localhost** (dirección IP 127.0.0.1), en el puerto **3000**. Una vez dentro del entorno gráfico proporcionado por la herramienta será necesario especificar la conexión que se encargará de realizar la inyección de los datos. En este caso y como puede apreciarse en la imagen, se ha asociado este campo con la base de datos desarrollada previamente sobre **InfluxDB.** Llegados a este punto, Grafana se encarga de mantener la conexión abierta con su proveedor de datos. La documentación aportada no pretende dar una explicación total del proceso de montaje, sin embargo, el aporte de esta breve descripción tiene por objeto clarificar el modo en el que las distintas fases descritas en el apartado se comunican.

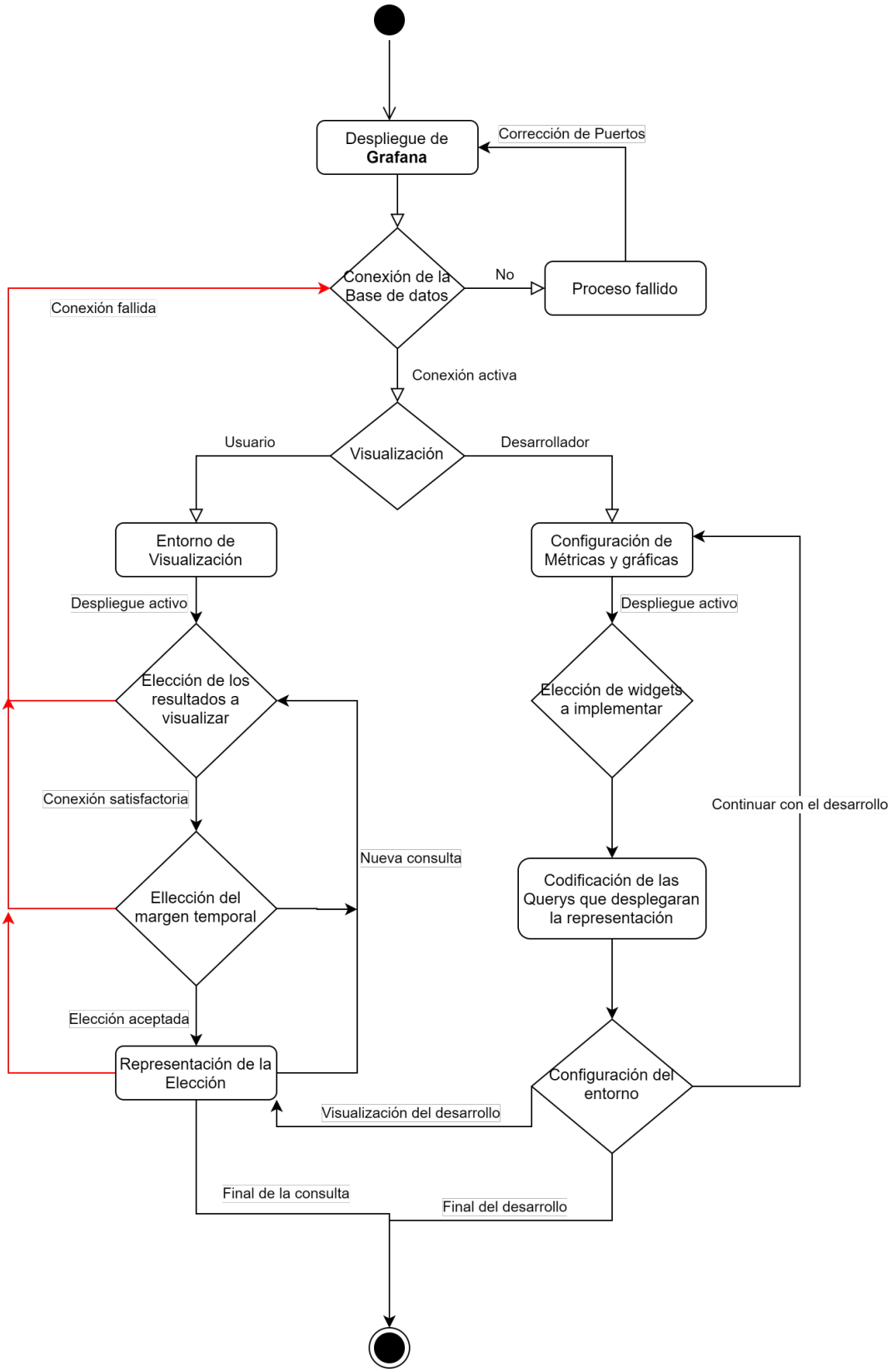

Ilustración 26: UML Fase 3

Evaluación de la implementación y análisis de Resultados

En capítulos anteriores se definieron tanto las herramientas utilizadas para el trabajo, incluyendo sus principales funcionalidades e información sobre sus repositorios, como la totalidad de la arquitectura que caracteriza el trabajo. En el capítulo anterior, se describen, tanto la arquitectura como los pasos necesarios para alcanzar la propuesta final del desarrollo. En este capítulo se presentarán los resultados obtenidos del desarrollo. El capítulo se subdivide en varias secciones en las cuales se aportan capturas del desarrollo, así como resultados del análisis sistemático de los algoritmos de control de congestión.

5.1 Métricas de rendimiento

Antes de presentar los resultados es necesario conocer el origen de los resultados que, más tarde arrojarán luz sobre los algoritmos de control de congestión definidos en el documento. Los datos aportados a continuación conforman los resultados aportados gracias al despliegue de la primera fase de desarrollo sobre el trabajo. Los datos que **Pantheon** provee resultan clarificadores, pero son escasos y no todos los algoritmos son capaces de hallar dichos resultados.

Pantheon, como ha sido definido en capítulos anteriores permite realizar una conexión Cliente-Servidor, que usa un canal de transmisión para enviar un flujo constante de datos, a la mayor velocidad posible que permite evaluar las condiciones, tanto del canal como de los *bufferes* en ambos sentidos de la comunicación, tanto enlace ascendente, como enlace descendente. Una vez desplegada la comunicación, con una terminal para cada uno de los dos procesos desplegados, los algoritmos mencionados proporcionan por pantalla una aproximación media del **throughput** obtenido en la comunicación. En una primera aproximación con el canal de comunicación, que sustenta ambos extremos, desplegado sobre sendas terminales sin el uso de una traza que manipule el canal para poner a prueba los distintos algoritmos de control de congestión, nos hallamos frente al escenario ideal, configurado para transmitir la máxima tasa posible. En la configuración definida, los resultados que **Pantheon** arroja sobre el estudio son los propios de una red de quinta generación, sobre ondas milimétricas. Los resultados son del orden de 1.5 Gigabit, algo mayor en ciertas ocasiones, llegando incluso a los 3 Gigabit/seg. Una tasa impresionante, que se verá reducida por la variabilidad del canal, que imposibilita la transmisión en muchos momentos.

Una vez introducidos los conceptos definidos en los párrafos precedentes, el documento proporciona resultados cuando se usan las trazas definidas en "**3.1 Trazas**" como canal. Dichas trazas pondrán a prueba las métricas de control de congestión, llevándolas en ciertos casos al límite y ayudando por lo tanto a evaluar cuál de ellas llega a obtener una conexión fiable en los casos más extremos, limitando la comunicación tanto en la distancia como en los obstáculos encontrados.

5.1.1 Herramienta desplegada

En capítulos anteriores se ha descrito el funcionamiento de **Pantheon**, en este caso se aporta ciertas capturas que facilitan la comprensión de las herramientas desplegadas únicamente para el desarrollo de este trabajo. Como objetivo, este documento, no pretende entrar al detalle sobre las herramientas de despliegue, sino aportar la suficiente información como para facilitar la comprensión a nivel técnico y de desarrollo sobre los diferentes módulos.

5.1.1.1 Tiempos de medición

En este subapartado se definen los tiempos de medición que han sido necesarios para que el canal tenga, al menos, una oportunidad de conexión. Esto se refiere a un tiempo de despliegue por algoritmo que no llene la memoria de almacenamiento del servidor encargado de alojar los repositorios. A su vez, el tiempo de despliegue debe ser suficiente como para que, en un caso límite, en el que la comunicación ha de realizarse sobre un canal angosto, las ejecuciones generen resultados.

En la siguiente imagen, **Ilustración 27**, se recoge un ejemplo de enlace. En este caso se trata de una comunicación libre de parametrización por traza, por lo tanto, las velocidades observadas, en este caso para el algoritmo **Cubic**, resultan de la configuración de un enlace ideal en el que la distancia entre los extremos de la comunicación es ínfima.

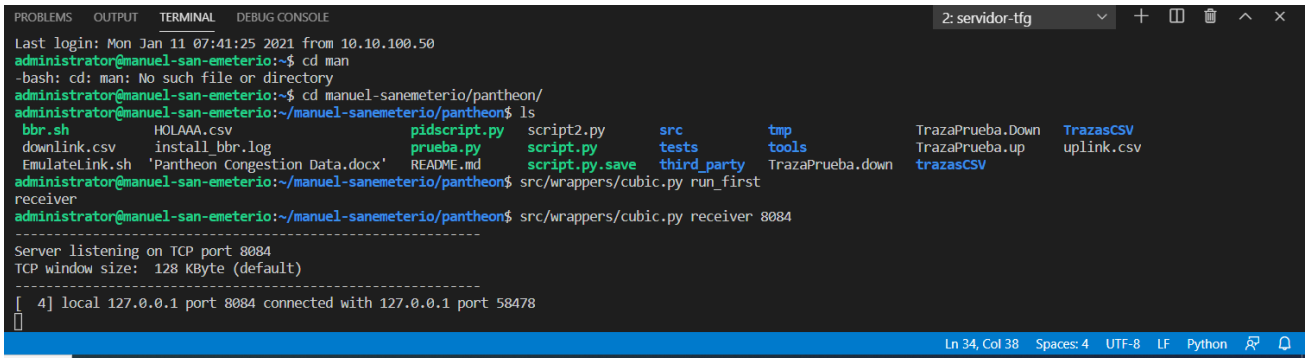

Ilustración 27: Enlace Cubic sin traza de entrada

Como se puede observar sobre cualquiera de las terminales o procesos que conforman la comunicación, **Ilustración 28**, el tiempo durante el cual se ha desplegado la medición, son exactamente **75.00 segundos**, a pesar del corto periodo en el que el enlace ha posibilitado la comunicación, gracias a cierto calculo previo, realizado sobre cada uno de los diversos algoritmos que componen el estudio, se ha conseguido que, ya en esta temprana fase del desarrollo, se obtengan unos resultados aproximados, que permiten hacerse una idea de cuál debe ser la magnitud de los resultados desplegados posteriormente. En la siguiente **Ilustración 28**, se muestran los resultados sobre el experimento propuesto, estos resultados clarifican la potencia de la red. Además, mediante este ejemplo se demuestra de una forma visual que los datos transferidos, y por lo tanto los resultados almacenados son de una magnitud considerable, por lo que, si los tiempos de testeo no hubieran sido especificados previamente, nuestro programa podría llenar la capacidad de un servidor en pocas horas, e incluso minutos. Como se puede observar en la figura, la **ventana TCP**, es de **2,50 MBytes**, además la tasa propia de la comunicación ha resultado de unos **7.20 Gbits/seg**, lo que implica que en el corto periodo durante el que se desplegó la comunicación, la cantidad de datos transferida ha resultado de unos **62,9 Gbytes**. Este cálculo no resulta tan preciso como el que se postulará en secciones posteriores del documento, pero ofrece una cota que resulta bastante realista.

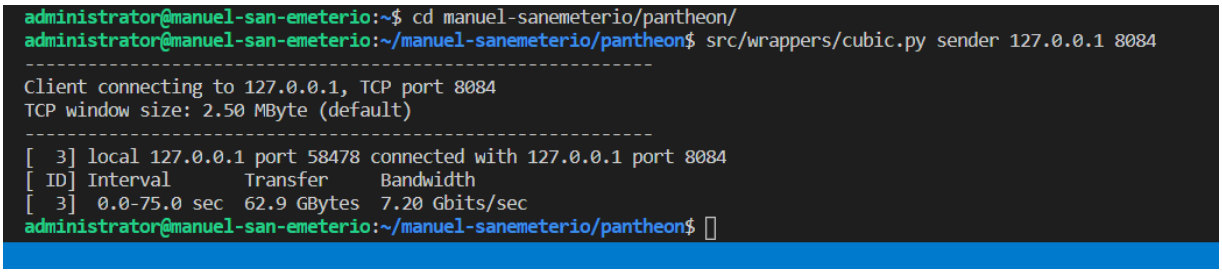

Ilustración 28: Cubic sin traza de entrada resultado

Una vez estudiada la configuración anterior, se aclara que todos los algoritmos han sido dotados de la misma oportunidad de transmisión, es decir, el tiempo elegido para las pruebas es el máximo permitido en el enlace de cada uno de ellos, ofertando, por lo tanto, la misma calidad de enlace que proporcionada mediante las trazas de entrada dotara al sistema de unos resultados homogéneos y de fácil estudio.

5.1.1.2 Datos portables en la configuración de Pantheon

En esta sección se aportan los datos obtenidos mediante la primera fase del despliegue, constituida por la herramienta **Panthen** [16], el túnel desplegado sobre **Mahimahi** [18], y los repositorios que conforman el estudio de despliegue. En secciones posteriores se hará una breve comparativa sobre la que toman parte, tanto los resultados obtenidos y definidos en este apartado, como los resultados obtenidos mediante el procesado de los ficheros de salida, gracias al esnifado del canal. Estos datos conformarán la entrada de la herramienta de representación gráfica. En primer lugar, sustentando la información especificada textualmente en este subcapítulo se añaden gráficos similares a los presentados en el apartado anterior.

Para comenzar en esta sección, en la siguiente **Ilustración 29**, se muestra como gracias a la herramienta **MahiMahi** y las trazas que conforman la entrada del trabajo, definidas en el capítulo "**3.1.1 Trazas**", desplegamos el canal sobre el cual la comunicación comenzara a fluir.

Ilustración 29: Cubic sobre mm-link

El comando que desencadena la comunicación, aportado por la herramienta **mm-link**, que define el canal mediante unas trazas de entrada, se muestra a continuación. En la **Ilustración 29**, se puede apreciar cómo una vez desplegado el comando que inicia el canal conforme a los datos de la traza de entrada, cambiará el símbolo del **prompt**, añadiendo los caracteres **[link]**:

$ mm-link /home/administrator/traces/Build5_1_UE_rx.txt /home/administrator/traces/Build5_1_UE_rx.txt --meter-all --uplink-log TrazasCSV/UpLink/UPBuild5_1_UE_rxverus.csv --downlink-log TrazasCSV/DownLink/DOWNBuild5_1_UE_rxverus.csv

Los caracteres que aparecen en negrita, sobre comando que precede este párrafo, son las instrucciones propias del enlace emulado mediante la herramienta **mm-link**. En este punto cabe destacar que la configuración inicial de la herramienta ha sido modificada previamente para la adaptación sobre el trabajo. Una vez definidos estos parámetros propios del comando en sí, es hora de introducir las trazas que simularán un canal real sobre el que previamente se midieron las características físicas, esta parte está expuesta con una fuente de color verde. Por último, tanto el texto que aparece en azul sobre el comando anterior como el que se representa mediante la fuente de color amarillo constituyen, por un lado, la métrica de salida del canal ascendente y por otro la del canal descendente respectivamente.

Una vez desplegado este comando solo será necesario repetir el proceso descrito en el apartado anterior. Tras la consecución de este proceso sobre todos y cada uno de los algoritmos, en la siguiente **Tabla 3**, se exponen

los resultados obtenidos mediante **Pantheon**, para todos los algoritmos del despliegue, sobre los primeros tres escenarios del desarrollo. En la siguiente tabla solo se muestra el dato de tasa binaria, ya que es el único que **Pantheon** da a conocer en el momento de ejecución, gracias a sencillas operaciones realizadas durante el proceso de comunicación. Por consiguiente, estos datos nos servirán como baremo, que permitirá intuir si las medidas registradas en fases posteriores del trabajo toman valores similares a los recogidos en este epígrafe o si por el contrario la herramienta de esnifado tiene algún problema.

Tabla 2: Análisis de resultados sobre Pantheon

	Throughput		
Algoritmos	**Build1_1_UE_rx**	**Build2_1_UE_rx**	**Build3_1_UE_rx**
Cubic	160 Mb/s	122 Mb/s	60 Mb/s
Vegas	5 Mb/s	6 Mb/s	10 Mb/s
Verus	33,5 Mb/s	26 Mb/s	23 Mb/s
Bbr	150 Mb/s	130 Mb/s	65 Mb/s
Sprout	100 kb/s	200 kb/s	150 kb/s
Scream	220 kb/s	300 kb/s	180 kb/s
Pcc	3 Mb/s	2 Mb/s	9 Mb/s
Vivace	8 Mb/s	7 Mb/s	70 kb/s
Pcc_experimental	4,5 Mb/s	7,5 Mb/s	25kb/s
Fillp	110 Mb/s	45 Mb/s	40 kb/s
Fillp_sheep	6 Mb/s	5 Mb/s	150 b/s
Ledbat	66 Mb/s	46,5 Mb/s	45,5 kb/s

En la **Tabla 3** se recoge el valor del *throughput*, para todos los algoritmos escogidos como parte del estudio final. En esta primera fase, por lo tanto, ya pueden apreciarse ciertos atisbos que confluyen en las conclusiones expuestas en el último capítulo de este documento, por ejemplo, se puede observar cuáles son los algoritmos que presentan un mejor rendimiento, remarcados con un color celeste como fuente de texto.

También podemos apreciar cómo, en función del algoritmo de control de congestión elegido, la conexión no solo varía enormemente en cuanto a la tasa binaria se refiere, sino que también indica si la conexión será finalmente posible o, por el contrario, como podemos observar sobre los resultados de las métricas aportadas por **Fillp_sheep** sobre el escenario 3, "**Build3_1_UE_rx**", la tasa de envió no permite ni siquiera el establecimiento de la conexión.

Una vez descrito este análisis previo, que servirá como pilar en la consecución de las siguientes secciones, el documento continúa con la evaluación de rendimiento, para ello se dotara al texto de ciertas imágenes que tratan de clarificar el porqué de las conclusiones que serán definidas en el episodio final del documento.

5.2 Evaluación de rendimiento

Este subcapítulo conforma un importante paso en el desarrollo de los resultados finales. Se recogen los resultados, incluyendo los obtenidos tras la ejecución del proceso descrito en la sección anterior, sumado a la extracción de los documentos **CSV**, su procesado y posterior inyección sobre la base de datos.

Previo al análisis de los datos expuestos en la parte final de este subcapítulo, sobre la sección "**5.2.1.2 Datos Inyectados, configuración de las tablas**", el documento aporta un pequeño manual de usuario que permite

comprender los diversos paneles que forman la herramienta desplegada, así como la información que en ellos se expone.

Antes de comenzar con este subcapítulo, cabe destacar que el despliegue de la herramienta de representación gráfica sobre **Grafana**, fue elegido por contemplar una relación basada en series temporales, dotando al trabajo de un valor añadido, la escalabilidad. En este proyecto se exponen datos dependientes del **TimeStamp**, lo que permite a estudios futuros sobre redes físicas, determinar el momento exacto en el que se produce un fallo o simplemente una medida importante del canal. Así mismo, **Grafana** aporta herramientas que permiten la consecución de alarmas en función del flujo de datos que va llegando a la herramienta de representación.

5.2.1 Herramienta desplegada

Como ya se ha definido en capítulos anteriores, específicamente sobre la sección "**4.1.4 inyección y tratamiento de los datos**", la herramienta desplegada, consta de varias sub-herramientas, que concuerdan en la consecución de un entorno de estudio, sobre el cual la investigación toma un carácter informativo, gracias al flujo de los datos, desde su inicio en la obtención de las trazas de entrada "**3.1.2 Trazas**", hasta la representación gráfica consecuencia del procesado de los numerosos datos esnifados sobre el canal de comunicación.

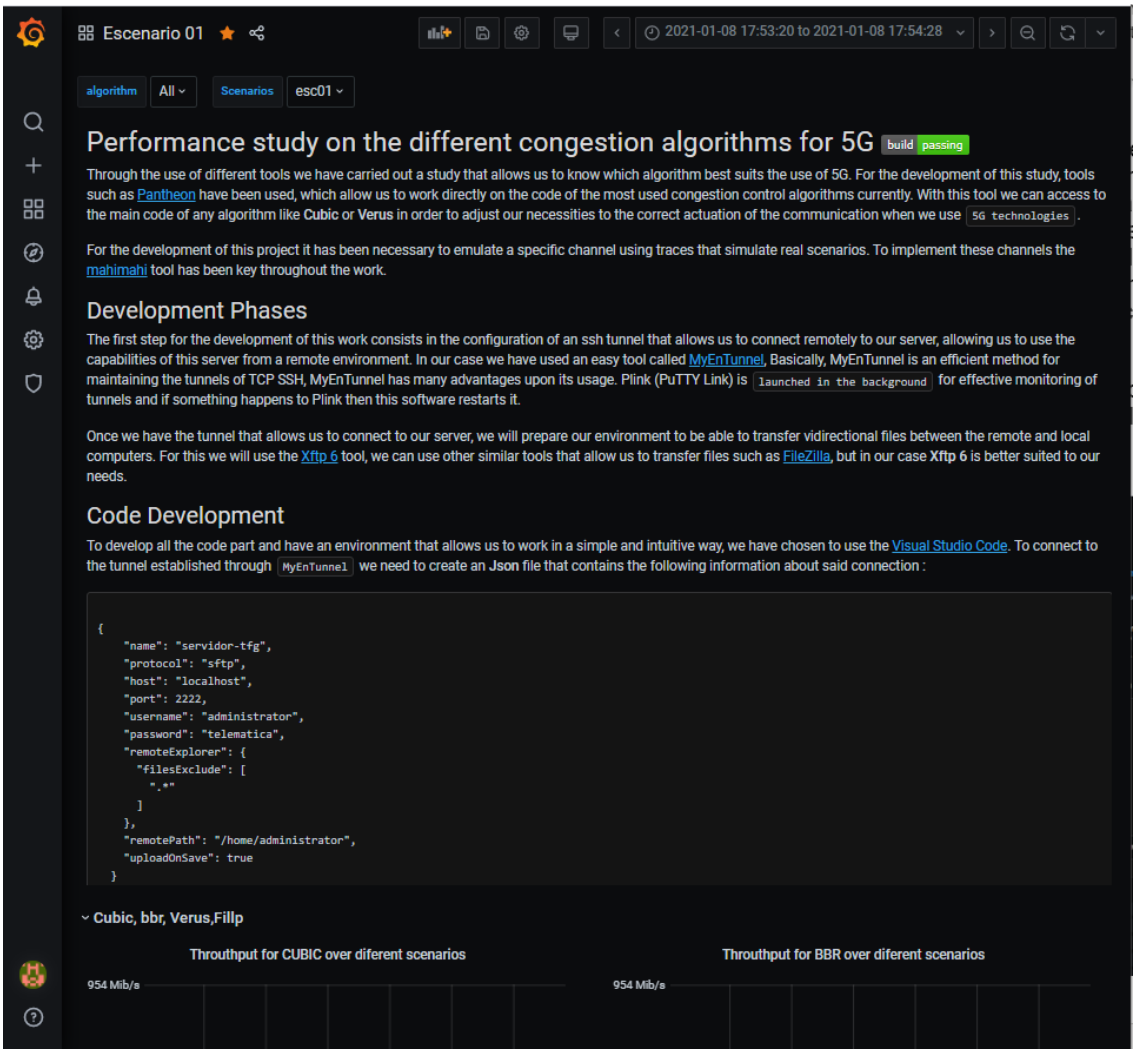

Ilustración 30: Mapa de Grafana

En esta sección se expone una pequeña introducción que pretende facilitar el uso de la herramienta final del despliegue, por ello en la **Ilustración 30**, se muestra la apariencia final del entorno gráfico.

Como puede apreciarse en la ilustración anterior, el entorno de representación gráfica va precedido de un documento **ReadMe** [17], descrito en lenguaje **MarkDown** [22], que trata de facilitar al usuario la comprensión del estudio realizado mediante pequeñas pautas, además este documento aporta trozos del código que fue generado para la consecución de los diversos pasos del desarrollo que han sido definidos sobre este documento, enlaces sobre las herramientas de reposición y demás links, que facilitarán al usuario la formación en los términos que permiten la comprensión del trabajo y de sus resultados.

En primer lugar, bajo el documento **ReadMe**, expuesto anteriormente, podemos encontrar varias pestañas que nos permitirán ir moviéndonos a lo largo del despliegue. En la **Ilustración 31**, se exponen sus nombres y además puede apreciarse cómo en el margen izquierdo, posterior al nombre de la sección aparece un valor en gris, este campo indica el número de paneles que conforman la sección. En esta ilustración puede apreciarse que el primero de los campos toma el nombre de cuatro de los algoritmos introducidos en la investigación, este dato es un adelanto de información, que permite saber cuáles son los algoritmos que mejor se han comportado en media sobre las distintas trazas de entrada. La siguiente sección remarcada en este documento son las tasas de llegada, esta sección ilustra las diversas tasas de llegada obtenidas sobre todos los escenarios, para cada uno de los algoritmos del estudio. Por último, para poder obtener todos los valores de forma meticulosa, las siguientes secciones están compuestas por diversos gráficos que ilustran las capacidades resultantes de cada enlace.

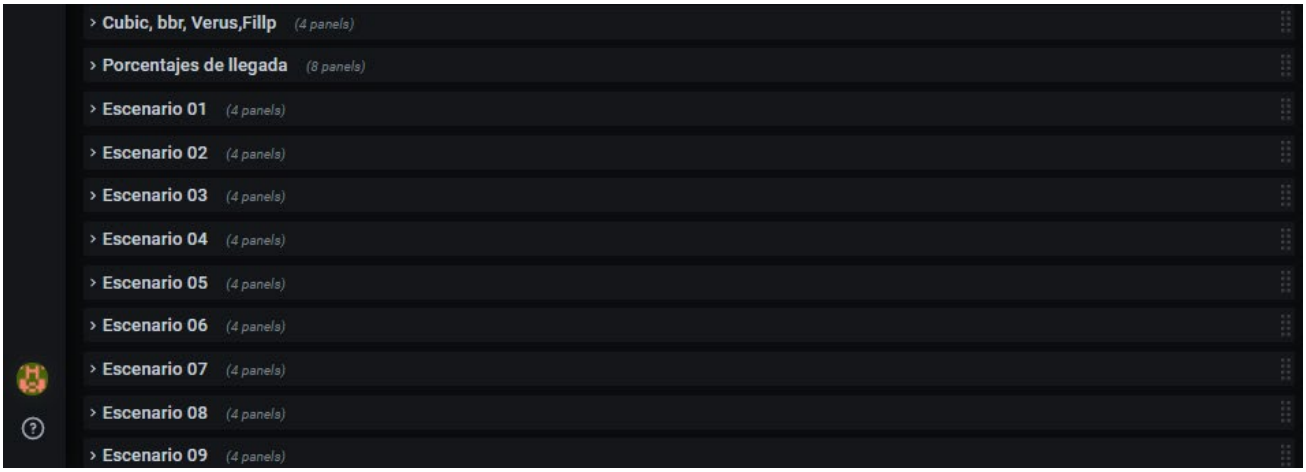

Ilustración 31: Secciones en Grafana

En la siguiente **Ilustración 32**, se recogen las distintas tablas que podemos encontrar sobre cada uno de los escenarios que componen el proyecto. Como se ha definido en el párrafo introductorio los resultados que conforman la salida son los datos representados por **Grafana** [7], gracias al uso de **Querys SQL** [31] sobre el lenguaje de base de datos de **InfluxDB** [29]. En concreto, la **Ilustración 32**, muestra los resultados recopilados sobre el escenario "**Build2_1_UE_rx**".

```
> select * from esc02
name: esc02
time                Canal      FirstTimeStamp PercentageOfArrived PercentageOfLoss Throughput         TotalDataSize algoritmo       firstTimeStamp percentageOfArrived
----                -----      -------------- ------------------- ---------------- ----------         ------------- ---------       -------------- -------------------
1610124801285000000 upStream   3393           100                 0                1817781.8256552261 21006514      cubic
1610124804222000000 upStream   3218           100                 0                59086.219096425586 688901        vegas
1610124807408000000 upStream   3260           100                 0                1169210.0068257907 13489322      bbr
1610124810296000000 upStream   3253           100                 0                24196898.178331736 252373648     verus
1610124810421000000 upStream   3234           100                 0                4606.658446362515  467           sprout
1610124813361000000 upStream   3055           100                 0                9060.758643840214  104594        scream
1610124816634000000 upStream   3674           100                 0                37584.1446737289   460340        pcc
1610124819800000000 upStream   4656           100                 0                255940.91678012954 3101620       vivace
1610124822946000000 upStream   4713           100                 0                201531.1372123589  2414998       pcc_experimental
1610124826087000000 upStream   5677           100                 0                473329.6761745842  5673152       fillp
1610124828372000000 upStream   5742           100                 0                55740.39968950245  457775        fillp_sheep
1610124830962000000 upStream   5281           100                 0                45702.945456370115 455407        ledbat
1610124834637000000 downStream                                     0                118749960.64393    1376653450    cubic           3341           100
1610124837603000000 downStream                                     0                4244331.354375774  49686265      vegas           3090           100
1610124841362000000 downStream                                     0                129827823.80425559 1510887528    bbr             3096           100
1610124844158000000 downStream                                     0                1128945.0041681554 11849689      verus           3232           100
1610124844288000000 downStream                                     0                132504.20168067227 11826         sprout          3331           100
1610124847327000000 downStream                                     0                202163.86044462593 2351848       scream          3059           100
1610124850487000000 downStream                                     0                1352795.8313205957 16452872      pcc             4385           100
1610124853894000000 downStream                                     0                5917122.644328357  71784313      vivace          4600           100
1610124857096000000 downStream                                     0                5709063.456119679  69647006      pcc_experimental 3961          100
1610124860740000000 downStream                                     0                39436098.22953551  473331769     fillp           5668           100
1610124862999000000 downStream                                     0                2566273.490387219  21323808      fillp_sheep     5049           100
1610124865879000000 downStream                                     0                45227853.096856736 450729477     ledbat          5290           100
>
>
```

Ilustración 32: Escenario 2 sobre InfluxDB

Los resultados obtenidos gracias al procesado llevado a cabo previamente son expuestos en forma gráfica facilitando su comprensión, y pasando ciertas medidas a unidades del Sistema Internacional, hasta finalmente, conformar las distintas tablas presentadas en la **Ilustración 33**. La primera de estas tablas muestra el momento en el que se realizó la comunicación, basándose en los *timeStamp* ofrecidos por la base de datos no relacional. Además, en el momento que el canal se encontraba en funcionamiento, esta tabla muestra tanto en la leyenda como, gracias a un diagrama de barras, el *throughout* obtenido por todos los algoritmos al ser desplegados sobre uno de los escenarios, todos ellos en escala logarítmica, lo que facilita la comparativa sobre todo para los algoritmos que han conseguido unas tasas binarias pobres en relación con los que mejor se adaptaron al enlace. Cabe destacar que, en esta tabla, se muestran tanto los datos correspondientes al enlace ascendente de la comunicación como al descendente, conservando el mismo tono para las medidas que pertenecen a un mismo algoritmo, sin embargo, sobre la leyenda aparecerá solo el valor máximo de *throughput*, que el algoritmo obtuvo, bien sobre el enlace ascendente o sobre el enlace descendente.

La segunda tabla desplegada sobre cada uno de los algoritmos y sobre cada uno de los enlaces, es un gráfico que muestra los valores del tamaño total de los datos que fueron transmitidos en la comunicación. En la leyenda inferior del grafico se adjunta el valor especifico que han tomado cada uno de ellos seguido del nombre del algoritmo que aporta el dato. Esta segunda tabla conserva el eje perpendicular logarítmico, lo que facilita una vez más la comparativa de todos los algoritmos entre sí.

Ilustración 33: tablas por escenario

La última tabla visible en la **Ilustración 33**, está formada por los valores de throughput obtenidos por todos los algoritmos de control de congestión definidos para este estudio. En este caso solo se contempla el enlace, o bien ascendente, o bien descendente, que se encarga de la transmisión de los datos; por lo tanto, el enlace encargado del envío de ACKs no se contempla en este gráfico. Una cuarta grafica que se expondrá en las siguientes secciones del documento conforma el porcentaje de llegada o perdida que ha sufrido el canal durante la conexión.

Concluido este apartado, el documento prosigue con una aproximación de los tiempos de procesado requeridos por la herramienta de procesado.

5.2.1.1 Tiempos de Procesado

Esta sección aporta una breve descripción del tiempo total que la herramienta de procesado empeña en la consecución del resultado alojado sobre la base de datos no relacional basada en sello de tiempo [34]. En esta parte el flujo de datos ha de ser procesado he inyectado sobre la base de datos. Para ello los ficheros de resultado, en formato **csv**, son transferidos al entorno local, donde se encuentra la herramienta que permite la consecución de los siguientes pasos del desarrollo.

Este paso del desarrollo confluye en la extracción de todos los ficheros de resultado, debido a su extensión el proceso descrito a continuación ha de realizarse por partes. La extensión de un solo escenario, entendiendo como tal a los ficheros de resultados propios de cada algoritmo de control de congestión, corriendo sobre una de las trazas integradas en el estudio, puede variar entre los **12 GBytes** a los **60 GBytes** de información. Es por esto por lo que el proceso de extracción, su posterior procesado y por último su inyección han sido resueltos de forma iterativa y automatizada, escenario por escenario hasta inyectar resultados en la base de datos.

Gracias a una nueva herramienta desarrollada para este trabajo, el procesado de datos y su posterior inyección sobre **Influx DB** [29], han sido realizados de forma autónoma. El ***script***, desarrollado para esta sección del desarrollo, es capaz de realizar en procesado de los datos basándose en el significado de las diferentes líneas de transmisión. Los puntos de medida son creados de la misma forma automáticamente por la herramienta lo que se traduce en la consecución del proceso de traspaso de resultados al entorno local un número de veces igual al de los escenarios que quieren desplegarse.

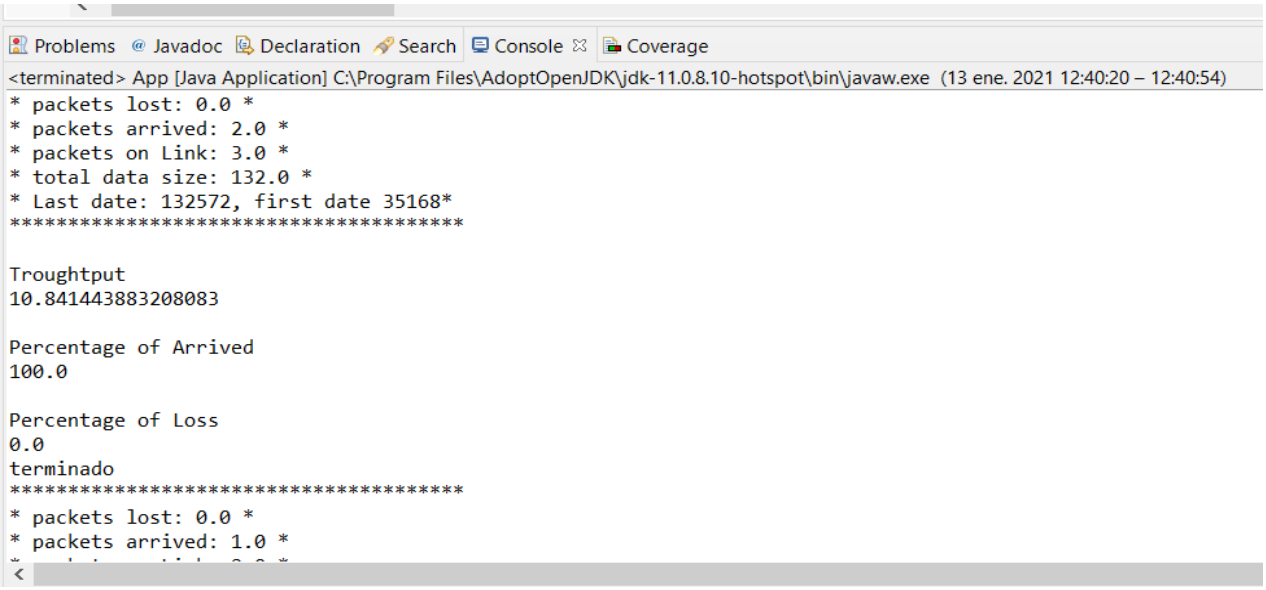

Ilustración 34: Terminal Trabajo Eclipse

En la **Ilustración 34** puede apreciarse como la terminal proporcionada por el entorno de desarrollo **Eclipse IDE** [25], continúa su procesado añadiendo los resultados calculados en base a la lectura de todas y cada una de las líneas que componen cada fichero csv. Esto puede arrojar claridad sobre el motivo por el cual el procesado puede ser una fase en la que es importante tener en cuenta el tiempo total de despliegue. Es imprescindible realizar cálculos que permitan prever el tiempo de desarrollo, calculando, en primer lugar, el número de líneas a procesar que en media tiene uno de los ficheros de resultados. El resultado es que, en media, cada uno de los ficheros procesados contiene unos 50 millones de líneas.

Este dato clarifica el por qué, en total, el número de horas requeridas entre la descarga de los ficheros de resultados su procesado y posterior inyección sobre la base de datos es de unas **46 horas**. A continuación, se expresa de forma sencilla cómo ha sido realizado este cálculo en base a los criterios medidos gracias a las primeras fases del despliegue en el que el flujo fue realizado en unidades pequeñas y automatizando solo el procesado e inyección de un único algoritmo sobre un único escenario. Además, cabe mencionar el significado de los acrónimos definidos como ***Td***, que identifica al tiempo de descarga del fichero de resultados alojado en el entorno remoto hacia el repositorio local. ***Tp***, implica el tiempo de procesado, es decir el tiempo que se tarda

en hacer un *Split*, línea por línea en el fichero que valla obteniendo resultados que en computo conformaran la salida del sistema, ***Tiny***, implica el tiempo que se tarda en realizar la inyección de los datos sobre **Influx DB** [29], por último, ***Ttotal***, representa el tiempo total que lleva el proceso completo referido a esta fase del despliegue, y expresado a continuación:

$$Ttotal = \sum_{i=0}^{9}(Td[Escenario[i]] + Tp[Escenario[i]] + Tiny[Escenario[i]])$$

$$Td[Escenario[i]] = \sum Td[resultado[algoritmo]]$$

$$Tp[Escenario[i]] = \sum Tp[resultado[algoritmo]]$$

$$Tiny[Escenario[i]] = \sum Tiny[resultado[algoritmo]]$$

Una vez llegados a este punto del documento, se indica la disposición de los datos sobre la base de datos no relacional basada en series temporales.

5.2.1.2 Datos Inyectados, configuración de las tablas

En esta sección se documentan los distintos elementos desplegados sobre la base de datos, cuáles son sus características y el porqué de su inclusión sobre el módulo final del desarrollo, para ello se aportan ilustraciones que tratan de clarificar el desarrollo.

En primer lugar, para aclarar el sentido que toman los diversos campos incluidos en la base de datos, en la **Ilustración 35**, se muestra cuáles son las propiedades de cada uno de los campos incluidos en el módulo de inyección. La herramienta desplegada para esta fase del desarrollo implica el uso de dos submódulos, el primero de ellos contempla la lectura de los ficheros **csv** y su posterior inyección sobre **InfluxDB** [29]. Este módulo es denominado como primario, pues se encarga de, en base a los recursos proporcionados anteriormente mediante **Xftp 6** [14], llamar al segundo submódulo, el cual se encarga de realizar la lectura línea a línea y su procesado obteniendo como resultado los datos representados en la **Ilustración 35**, y que conformarán la entrada del submódulo principal. En resumen, el código desarrollado, y que sustenta las propiedades de un trabajo **Maven**, se encarga de realizar un proceso iterativo de lectura y obtención de datos que desencadena en la nutrición de la base de datos.

```
63  if(DataReader.getFirstTimestamp() != null && !DataReader.getFirstTimestamp().isEmpty()) {
64
65      if(i >= 0 && i <= 11) {
66         // Write points to InfluxDB.
67          influxDB.write(Point.measurement("esc07")
68              .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
69              .tag("algoritmo", algoritmo[i])
70              .addField("Canal","upStream")
71              .addField("FirstTimeStamp", (DataReader.getFirstTimestamp()))
72              .addField("Throughput", DataReader.getThroughput())
73              .addField("PercentageOfArrived", DataReader.getPercetageOfArrived())
74              .addField("PercentageOfLoss", DataReader.getPercentageOfLoss())
75              .addField("TotalDataSize", DataReader.getTotalDataSize())
76              .build());
77
78      }else if(i >= 12 && i <= 23) {
79         // Write points to InfluxDB.
80          influxDB.write(Point.measurement("esc07")
81              .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
82              .tag("algoritmo", algoritmo[i])
83              .addField("Canal","downStream")
84              .addField("firstTimeStamp", (DataReader.getFirstTimestamp()))
85              .addField("Throughput", DataReader.getThroughput())
86              .addField("percentageOfArrived", DataReader.getPercetageOfArrived())
87              .addField("PercentageOfLoss", DataReader.getPercentageOfLoss())
88              .addField("TotalDataSize", DataReader.getTotalDataSize())
89              .build());
90
```

Ilustración 35: Código de Inyección

Como puede observarse en la **Ilustración 35**, cada punto de medida desplegado en el script consta de 7 campos esenciales para la obtención de los resultados buscados. En primer lugar, el nombre que caracteriza las tablas que conforman el estudio está compuesto por los caracteres "**esc0X**", donde X es un número del 1 al 9, directamente relacionado con el nombre de la traza del despliegue, **Tabla 4**, este proceso creara una tabla sobre la base de datos seleccionada previamente sobre la misma clase java que contiene la pieza de código mostrada en la **Ilustración 35**.

Tabla 3:Correspondencia Escenario-Tabla

Escenario	Tabla
Build1_1_UE_rx	esc01
Build2_1_UE_rx	esc02
Build3_1_UE_rx	esc03
Build4_1_UE_rx	esc04
Build5_1_UE_rx	esc05
InHM_1_UE_rx	esc06
InHO_1_UE_rx	esc07
RMa_1_UE_rx	esc08
UMa_1_UE_rx	esc09

Al tratarse de un despliegue sobre una base de datos no relacional basada en sello temporal, el código mostrado en la **Ilustración 35**, presenta un campo que no es visible, propio de este tipo de bases de datos, se trata del *TimeStamp* en el que el dato resulta inyectado sobre la base de datos. Ese tiempo toma el valor de **Tag** o

Etiqueta, proveyéndolo de las características de indexado relacionadas con este tipo de campos. Además, el código caracteriza al campo ***Algoritmo***, como una segunda etiqueta, esto se traduce en una búsqueda rápida y efectiva en la que el usuario, finalmente, podrá buscar no solo entre los resultados de un escenario, sino que tendrá la posibilidad de encontrar los datos seleccionando cierto orden en función de las dos etiquetas presentadas.

Los demás campos que conforman las tablas son simples ***Fields***, esto quiere decir que se trata de los datos que conforman los resultados. Estos toman menor importancia en cuanto al orden estructural de la base de datos se refiere, pero son el fruto del flujo completo de los procesos que se han ido desencadenando para hacer posible este trabajo, y por lo tanto conforman la base documental sobre la que se han obtenido los resultados presentados en las últimas secciones del documento. Gracias al campo **Canal** la tabla se divide en dos mitades, posibilitando el conocimiento no solo de las capacidades del enlace encargado de transmitir los datos, si no el resultado sobre el canal que las porta. Los campos **Throughput** y **TotalDataSize**, conforman el resultado de la comunicación, es decir, las capacidades medidas sobre el enlace; el primero se encuentra expresado en bits por segundo, y el segundo campo denominado **TotalDataSize** expresa el total de los datos sobre las tramas transferidas en bytes. El campo **PercentageOfLoss** y el campo **PercentageOfArrived**, introducen el número porcentual de los paquetes que se entregaron en la comunicación extremo a extremo sobre el canal remarcado en la línea, esto ofrece una visión global del número de paquetes retransmitidos o que simplemente han de ser rechazados probablemente debido a la congestión del canal.

5.3 Resultados gráficos y su configuración

Este es el apartado que conforma la parte final del documento, por ello en las siguientes secciones se documentan los resultados obtenidos sobre cada uno de los tres tipos de escenarios que han sido desplegados para la consecución del estudio.

En primer lugar, el documento expone la información obtenida sobre las trazas de tipo ***Building***, refiriéndose a las 5 primeras trazas de la **Tabla 4**, representada en el subcapítulo anterior. Además, esta sección aportará una breve descripción para cada uno de los protocolos que caracteriza tanto sus puntos fuertes como los que debilitan la conexión en determinados escenarios.

La segunda sección aporta la misma información, esta vez, referente a los datos obtenidos sobre enlaces de comunicación del tipo ***Indoor***. Estos escenarios pueden ser más exigentes para los algoritmos, ya que el número de objetos que pueden causar interferencias sobre el flujo de la señal es mucho mayor.

Por último, la tercera sección representa los escenarios obtenidos sobre las trazas del tipo ***RMa*** y ***UMa***. Esta sección conforma además los últimos dos escenarios propuestos para el trabajo. Cabe destacar que tanto los escenarios de tipo ***Indoor*** como los ***RMa*** y ***UMa***, presentan una descripción menos detallada de los resultados, mostrando únicamente los algoritmos que han sido capaces de establecer una conexión de mayor calidad.

Los datos aportados en esta sección son altamente variables lo que ha requerido de la repetición de los ensayos. Esto es debido a la variabilidad tanto del canal como de los escenarios propuestos y nos permite estimar qué algoritmo tendrá un mejor comportamiento cuando el medio posea unas características específicas.

5.3.1 Datos obtenidos sobre trazas del tipo Building

En este apartado se muestran los resultados obtenidos en las 5 trazas que se encargan de la emulación de un canal al aire libre en el que el número de edificios y la distancia con el **Hostpot** es variable, desplegando por lo tanto un entorno de conexión habitual entre los consumidores de redes móviles.

En primer lugar, la **Ilustración 36** muestra los diferentes resultados representadas sobre la herramienta desplegada en **Grafana** [7], para el escenario **esc01** o **Build1_1_UE_rx** descrito en el capítulo "**3.1.2 Trazas**". Esta sección aportará al menos un gráfico por cada escenario, en el texto que lo precede se describen tanto los resultados como las conclusiones perceptibles y una breve descripción de cómo la traza afecta al rendimiento de dichos algoritmos.

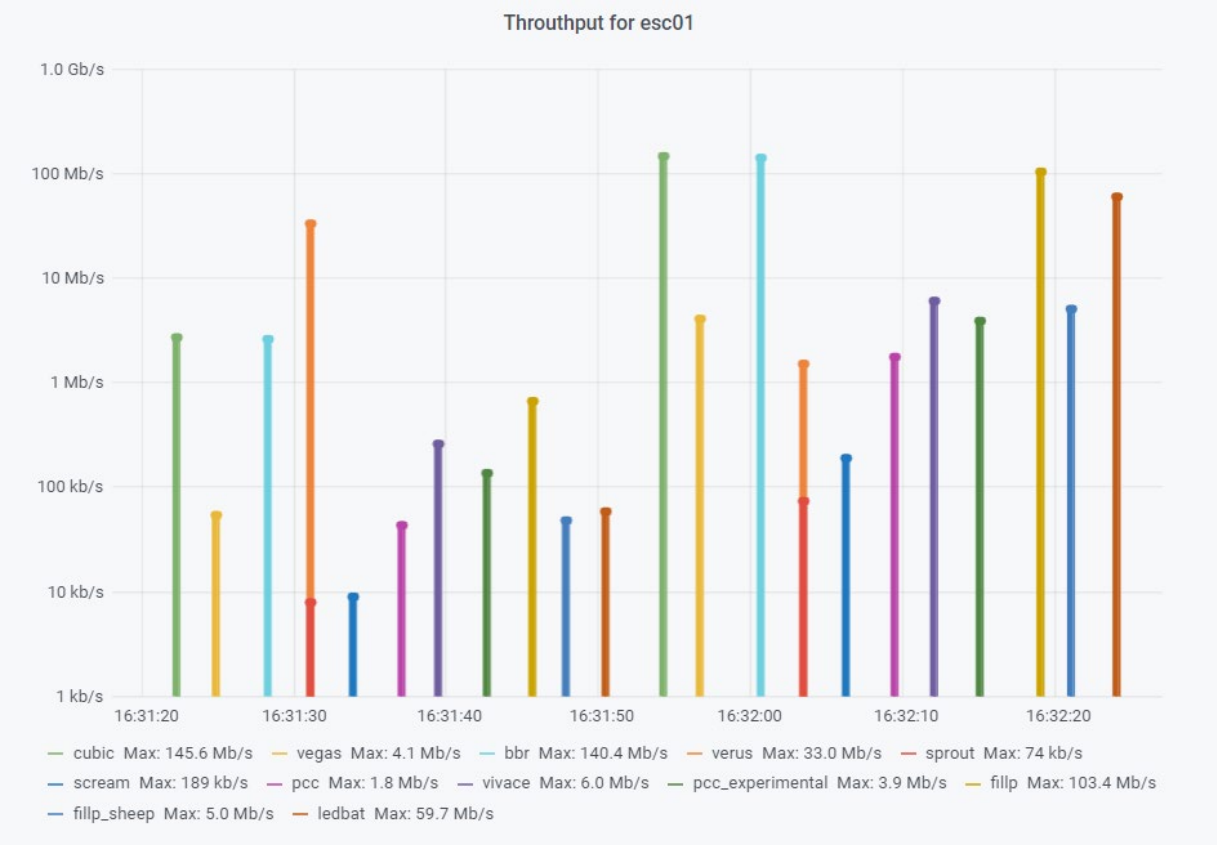

Ilustración 36: Esc01

Comenzando por el escenario denominado **Build1_1_UE_rx**, **Ilustración 36**, los resultados obtenidos en la comunicación sobre el enlace desplegado indican que los datos enviados no toman valores altos, lejanos a los obtenidos sobre un enlace sin traza de entrada; de hecho, muchos de los algoritmos desplegados sobre este enlace han obtenido valores de *throughput* muy bajos. Tanto **Sprout** como **Scream** han resultado totalmente dependientes de la calidad del canal en cuanto a opacidades se refiere, los resultados obtenidos por estos algoritmos son de entorno a los **Kb/s**. **PCC**, **PCC_experimental**, **Fillp_sheep Vivace** y **Vegas** continúan la lista obteniendo unos valores de *throughput* en el orden de los 5 Mb/s, resultado de una adaptación un poco superior, pero que sin duda sufre la ralentización de un canal altamente variable. En el medio de la lista los algoritmos **Verus** y **Ledbat** han obtenido valores bastante significativos. En concreto **Ledbat** ha demostrado una adaptabilidad superior y por lo tanto ha conseguido valores significativos de *throughput*. Finalmente, los

algoritmos que presentan un mejor comportamiento son **Cubic**, **BBR** y **Fillp**, que a diferencia de su variante **Fillp_sheep**, ha obtenido un valor de **103.4 Mb/s**. **BBR** ha obtenido un segundo puesto, gracias a un *throughput* de **140.4 Mb/s**. Por último, **Cubic** presenta el mejor comportamiento con una tasa binaria de **145.6 Mb/s**.

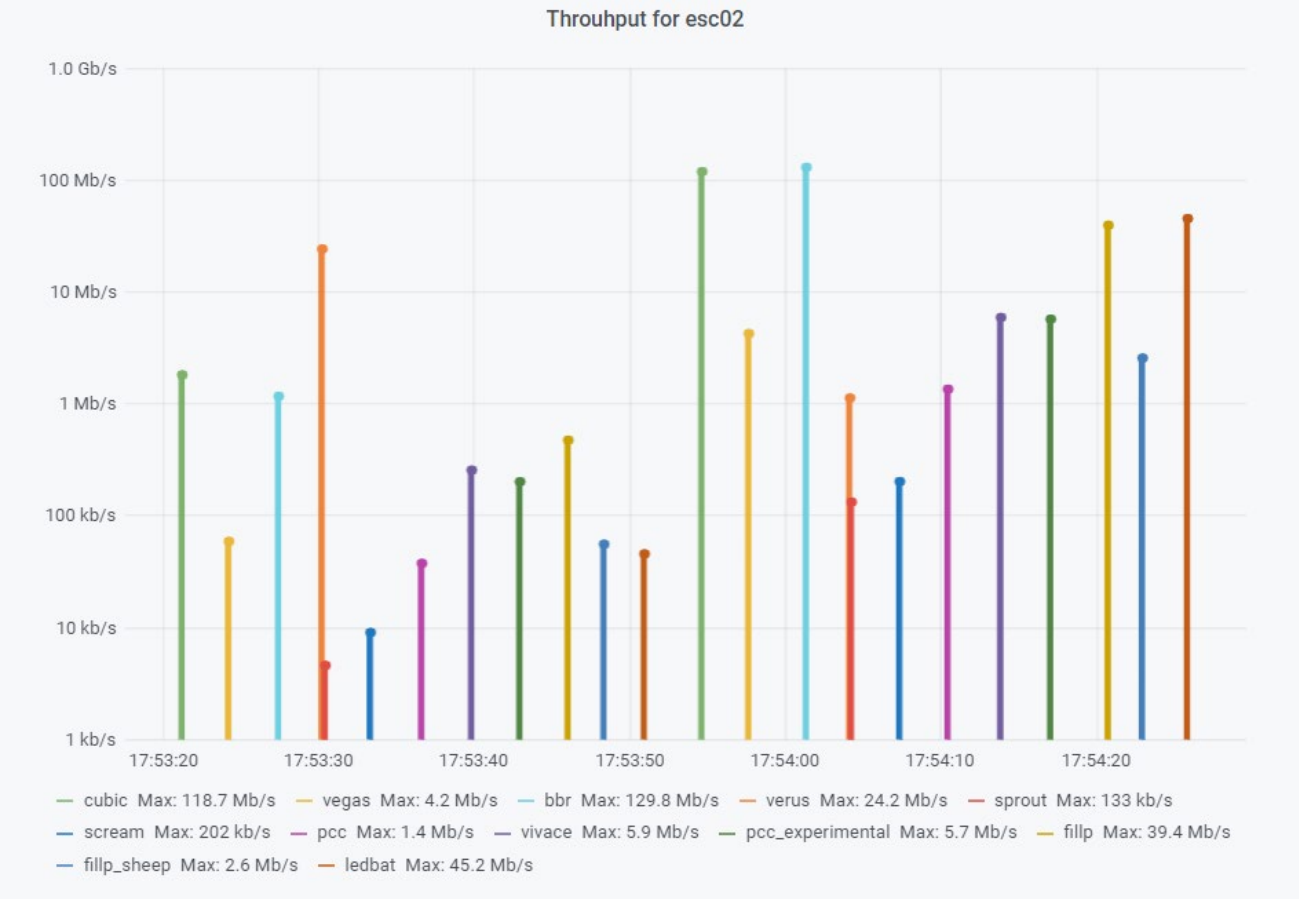

Ilustración 37: Esc02

En segundo lugar, el escenario denominado **Build2_1_UE_rx, Ilustración 37**, que conforma la segunda traza contemplada en este estudio, ha concluido en los resultados obtenidos en la comunicación sobre el enlace desplegado mostrando que los datos enviados no toman valores altos. De igual manera que sobre la traza anterior, tanto **Sprout** como **Scream** han resultado totalmente dependientes de la calidad del canal, los resultados obtenidos por estos algoritmos son de entorno a los **Kb/s**. **PCC**, **PCC_experimental**, **Fillp_sheep** **Vivace** y **Vegas** continúan la lista obteniendo unos valores de *throughput* en el orden de los 5 Mb/s. En el medio de la lista los algoritmos **Verus** y **Ledbat** han obtenido valores bastante significativos, destacando **Ledbat** que ha demostrado una adaptabilidad superior incluso a la percibida en el escenario anterior, con un valor de tasa binaria de **45, 2 Mb/s**. Nuevamente el mejor rendimiento lo proporcionan **Cubic** y **BBR**. En concreto **BBR** ha obtenido un valor de **129.8 Mb/s** mientras que **Cubic** ha obtenido una tasa de **118.7 Mb/s**. Los resultados expuestos muestran que este segundo escenario ha resultado positivo para ciertos algoritmos mientras que otros han decaído de forma significativa.

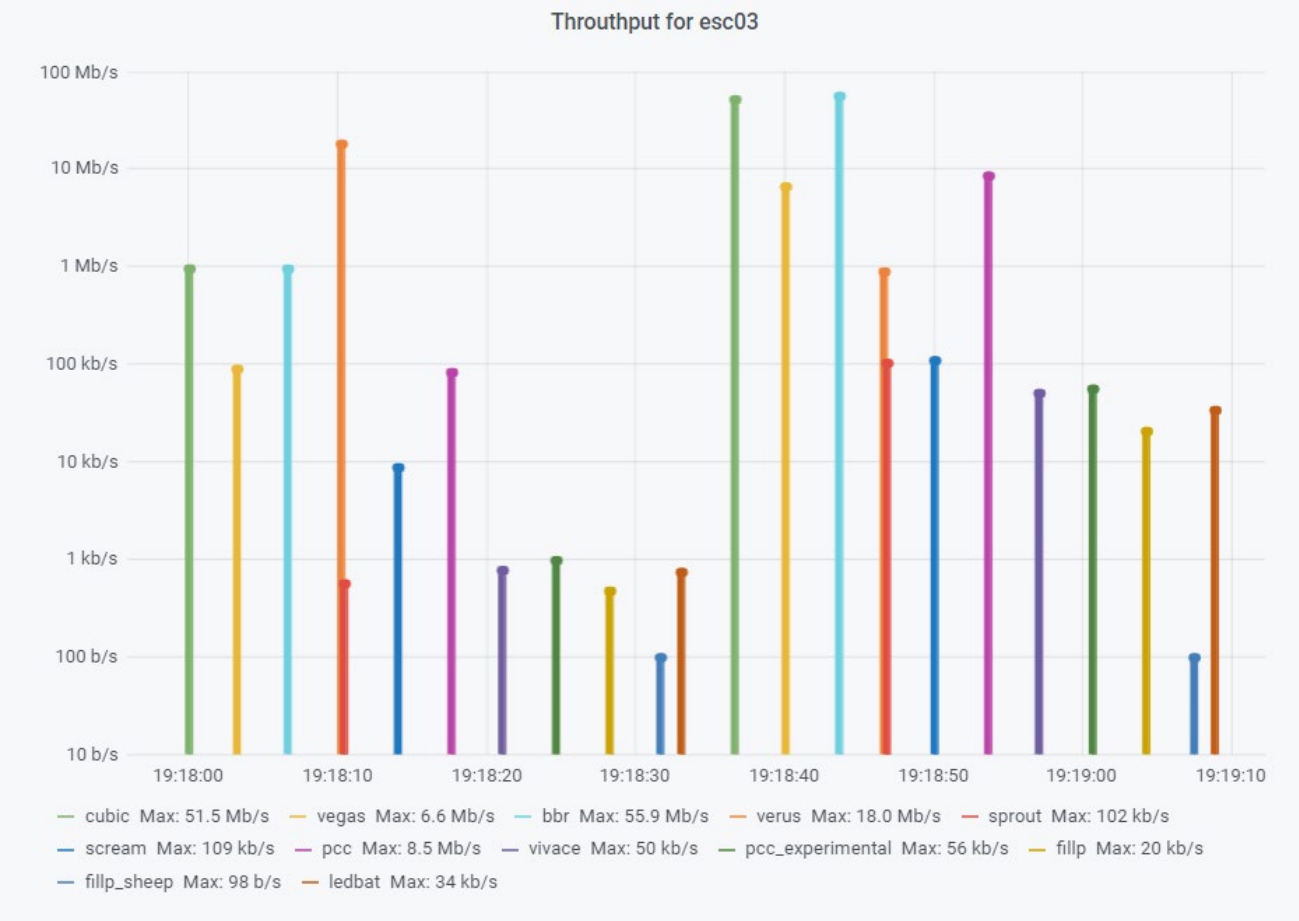

Ilustración 38: Esc03

En la **Ilustración 38** se describe el resultado obtenido en el despliegue caracterizado por la traza **Build3_1_UE_rx**, la mayoría de los algoritmos del estudio han perdido casi por completo la capacidad de comunicación, sin embargo, un nuevo patrón es descrito sobre esta traza. Al igual que en las trazas anteriores tanto **Cubic** como **BBR** muestran el mejor rendimiento, obteniendo **51,5 Mb/s** y **55,9 Mb/s** respectivamente.

En este caso en el escenario 3 algoritmos que habían conseguido un control sobre el canal, es el caso de **Ledbat** y **Fillp**, han decaído enormemente en sus valores quedando imposibilitados en este tipo de escenarios. Sin embargo, **PCC** alcanza una tasa de **8,5 Mb/s**. Por su lado **Vegas** aumenta la tasa conseguida en los dos escenarios anteriores lo que implica que su despliegue en redes variantes es una posibilidad que permite una baja tasa binaria que aporta fiabilidad al enlace a pesar de la distancia y los obstáculos definidos sobre esta métrica. Cabe mencionar que dentro de los algoritmos que han conseguido tasas aceptables, **Verus** ha resultado el algoritmo que mejor se ha adaptado a los inconvenientes del canal, manteniendo en este caso un *throughput* de **18,0 Mb/s**.

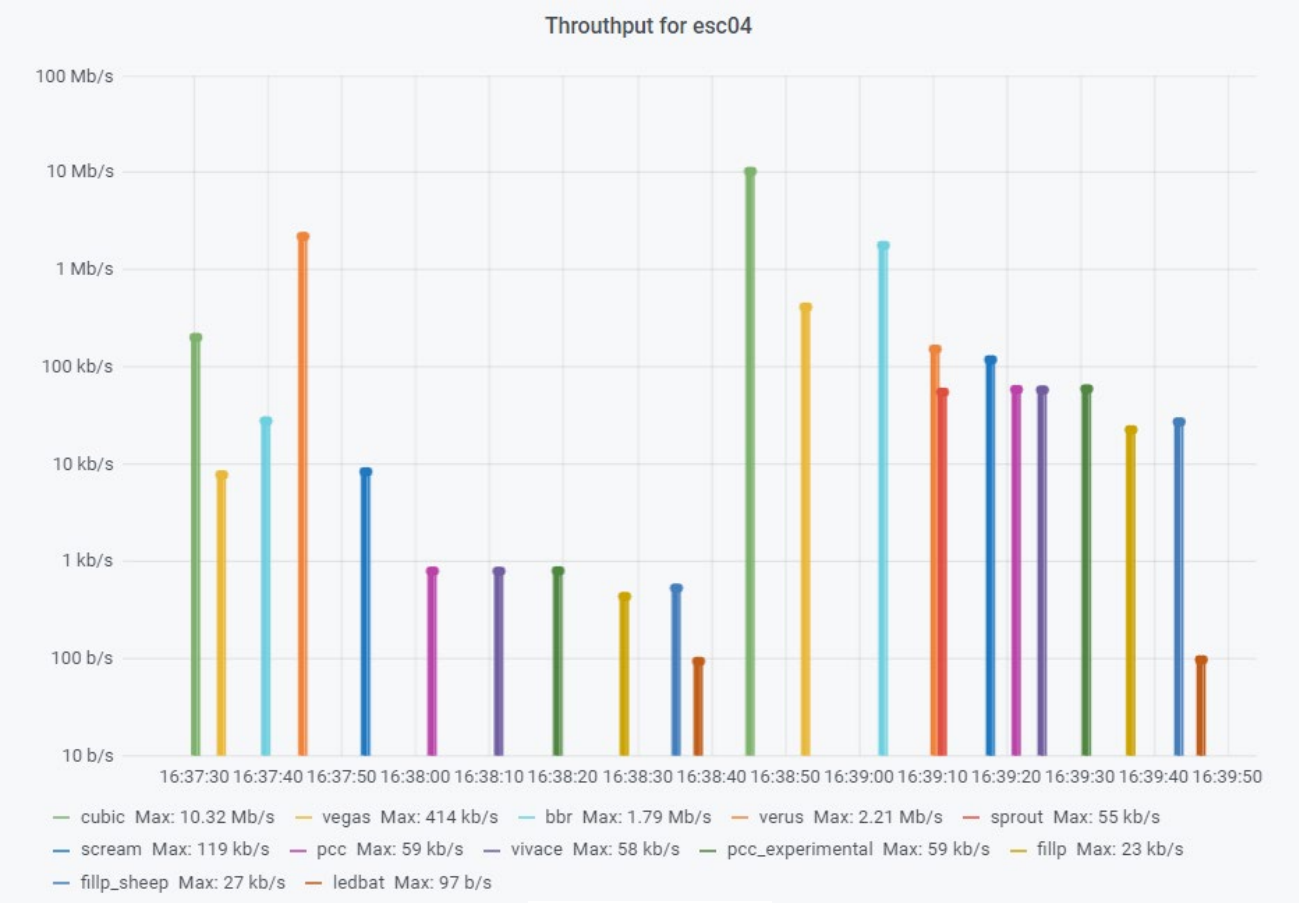

Ilustración 39: Esc04

La **Ilustración 39**, define los resultados obtenidos sobre la traza denominada **Build4_1_UE_rx**. En este escenario, es **Cubic** el protocolo que mejor se comporta, ofreciendo una tasa binaria de **10.32 Mb/s**, y cuatriplicando la tasa binaria del algoritmo que lo precede. Como ha sido descrito en el estudio anterior **Verus** ha sido el algoritmo que menos ha mermado porcentualmente, llegando en este caso a superar a **BBR**. Los demás algoritmos no logran mantener el establecimiento de la comunicación obteniendo unas tasas binarias pobres.

El último escenario del tipo **Building**, denominado **Build5_1_UE_rx**, que resulta además el más complejo de todos, el rendimiento de todos los algoritmos disminuye de forma muy significativa, no permitiendo su uso en servicios reales.

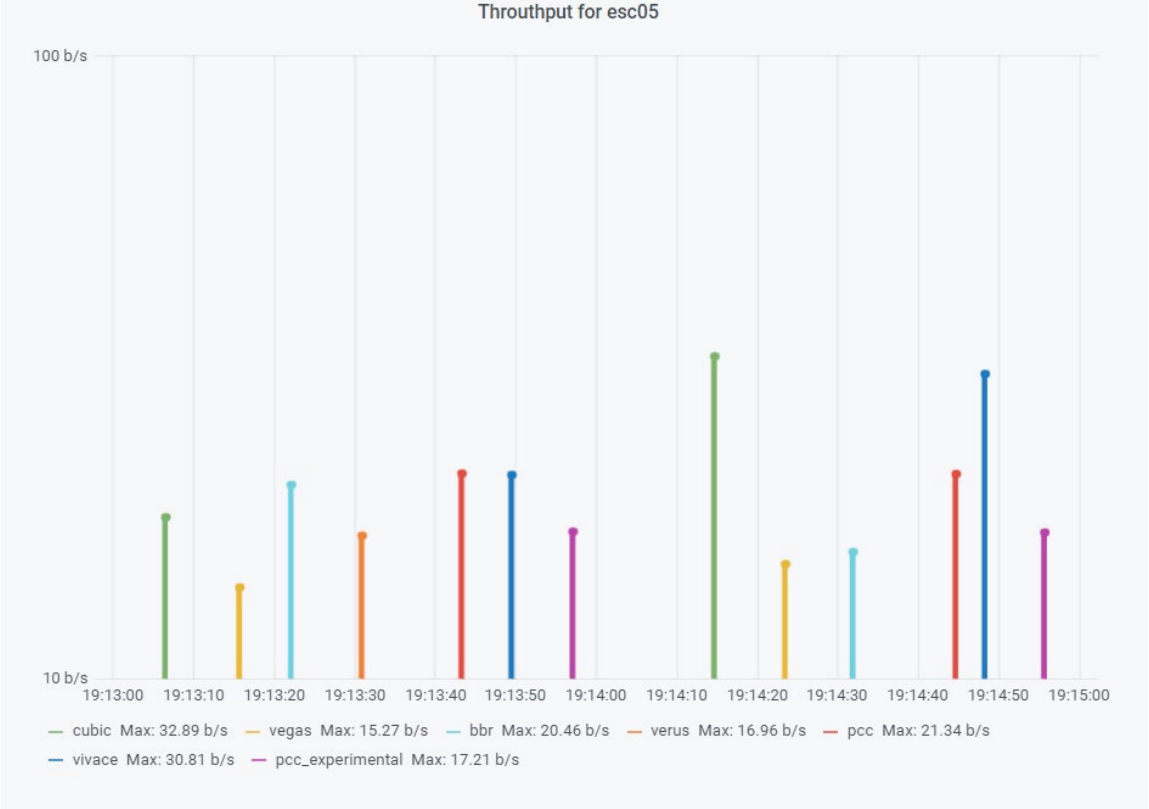

Ilustración 40: Esc05

En la **Ilustración 40**, se representan los resultados obtenidos en esta comunicación, las señales son tan débiles que apenas alcanzan tasas binarias de entorno a los bits por segundo, alguno de los protocolos elegidos ni siquiera ha sido capaz de establecer la conexión por lo que no aparece en el gráfico. **Cubic** queda en cabeza en la comunicación sobre este escenario, aun así, ningún algoritmo puede mantener una comunicación de datos continua.

5.3.1.1 Comportamiento de Cubic

Cubic ha demostrado un comportamiento ejemplar para todos los escenarios que componen esta sección, encabezando tres de los cinco escenarios desplegados. Este algoritmo de control de congestión plantea una solución capaz de adaptarse al canal y proporcionar tasas binarias altas sobre escenarios de despliegue con un canal de calidad media, y dota de comunicación a la infraestructura de redes móviles de quinta generación hasta en entornos limítrofes con la capacidad de envió de señal por ondas milimétricas **mmWave**. La tasa binaria proporcionada por este algoritmo va desde los **145,6 Mb/s** en el escenario 1, hasta los **10,32 Mb/s** para el escenario 4, límite de la comunicación en el estudio.

5.3.1.2 Comportamiento de BBR

Este algoritmo proporciona el mejor rendimiento en dos escenarios, **Build2_1_UE_rx** y **Build3_1_UE_rx** respectivamente. El comportamiento sobre el escenario uno, denominado **Build1_1_UE_rx**, ha resultado similar al de **Cubic**, por lo tanto, estos dos algoritmos crean la base de una buena comunicación sobre redes de quinta generación basadas en él envió de datos sobre canales **mmWave**. La tasa binaria ofrecida por este algoritmo varía entre los **140,4 Mb/s** a los **55,9 Mbps** desde el escenario uno hasta el tercer escenario que compone el estudio, decayendo enormemente en el cuarto escenario a **1,79 Mb/s**.

5.3.1.3 Comportamiento de Verus

Verus ha resultado ser una solución fiable sobre todo cuando la comunicación es variante en distancia e interrupciones u obstáculos. Las tasas binarias ofrecidas por este algoritmo, sobre la mayoría de los escenarios, son bastante robustas frente a las adversidades, siendo el protocolo que mejor se adapta al canal con una variación porcentual más baja que en el resto de las soluciones. La tasa binaria ofertada por este algoritmo varía entre los **33 Mb/s** a los **2,104 Mb/s** propios de la comunicación desplegada sobre el escenario cuatro.

5.3.1.12 Comportamiento de Fillp y Ledbat

Tanto **Fillp** como **Ledbat** son soluciones contemplables en un rango de obstaculización sobre el canal entre los rangos más bajos de dificultad, en el escenario uno definido como **Build1_1_UE_rx**, ambos escenarios consiguen tasas competitivas; en el caso de **Fillp** para el escenario uno consigue una tasa de **103 Mb/s**, por otro lado, **Ledbat** [33] queda por detrás con unos **59,7 Mb/s**. En la segunda fase del despliegue relacionada con el escenario 2, **Build2_1_UE_rx**, es **Ledbat**, el que muestra el mejor rendimiento con una tasa de **45,2 Mb/s**, mientras que **Fillp** reduce a más de la mitad la tasa binaria que ha sido registrada durante el experimento con unos **39,4 Mb/s**.

5.3.2 Datos obtenidos sobre trazas del tipo IndoorHM

El estudio realizado sobre las trazas de los escenarios **IndoorHM**, da lugar a los resultados representados sobre la **Ilustración 41**, donde se definen las tasas conseguidas por los diferentes algoritmos de control de congestión elegidos para el desarrollo del trabajo.

En este escenario los datos representados solo conforman los canales establecidos sobre la traza **InHM_1_UE_rx**, sobre la cual se obtuvieron datos aceptables en la conexión. En el caso del escenario que comparte la escenografía de despliegue definida sobre el capítulo "**3.1.2 Trazas**", y denominado **InHO_1_UE_rx**, el resultado del despliegue no ha sido posible para la mayoría de los algoritmos de control de congestión. En concreto, en el mejor de los casos se obtienen tasas del orden de decenas de bits por segundo, lo que indica que no hay una conexión que permita el envío de datos.

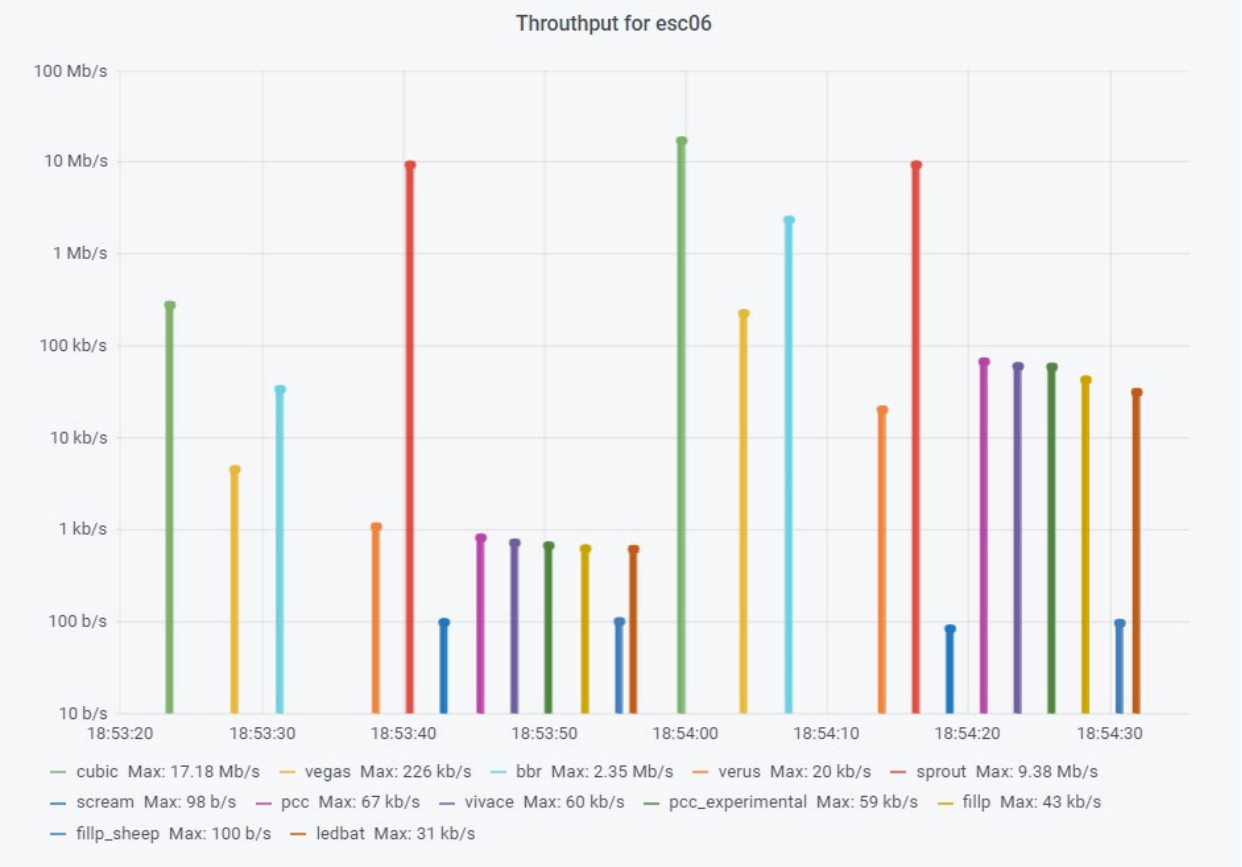

Ilustración 41: Esc06

Cabe mencionar que en el escenario **InHM_1_UE_rx,** varios algoritmos han logrado la consecución de los procesos de conexión. Los resultados que una vez más muestran que **Cubic** presenta el mejor rendimiento, con un *Throughput* de **17,18 Mb/s**. El envío total de datos representado en la **Ilustración 41**, representa una cantidad considerable capaz de sustentar numerosos servicios. Sobre este tipo de escenario se puede ver que **Sprout** presenta el segundo mejor rendimiento, con una tasa de **9,38 Mb/s**. **BBR** ha conseguido llegar a los **2,35 Mb/s**, por lo tanto, con ese rendimiento se sitúa el tercero en términos de rendimiento.

Los escenarios propuestos para formar parte de esta sección demuestran la alta dependencia y necesidad de adaptabilidad que la red sufre con el cambio de protocolo de control de congestión. Así mismo, se ha observado que el porcentaje de perdida de paquetes para la mayoría de los algoritmos sobre trazas **Indoor** ha resultado por debajo de la media observada para las trazas de tipo **Building**. Esto, como era de esperar, denota que la comunicación sobre ondas milimétricas, **mmWave**, es altamente dependiente del entorno, por lo tanto, su transmisión sobre escenarios como los descritos es este apartado ha dificultado enormemente la transmisión de los datagramas.

5.3.3 Datos obtenidos sobre trazas RMa y UMa

En este subcapítulo se referencian los resultados obtenidos tras aplicar las trazas de entrada **RMa_1_UE_rx** y **UMa_1_UE_rx** sobre los algoritmos, tal como se muestra en la **Ilustración 42**. Las trazas introducidas al sistema conforman una prueba de cómo se desenvuelven los distintos algoritmos de control de congestión sobre un entorno de macro celda rural y urbana.

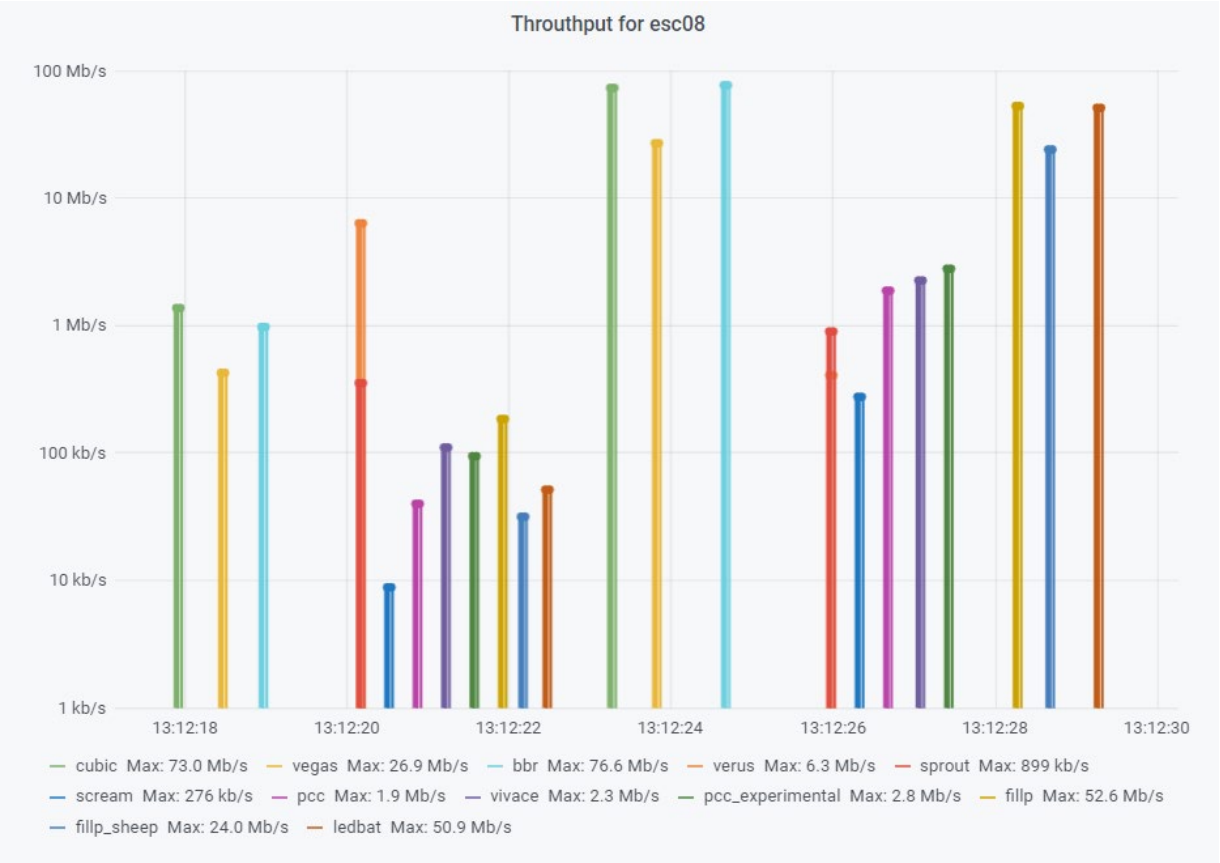

Ilustración 42: Esc08

Las dos configuraciones, tanto la macro celda urbana representada sobre la **Ilustración 42**, como la traza que configura el escenario de la celda rural, **Ilustración 43**, representan muestran que algunos algoritmos tienen un comportamiento muy superior al resto. En primer lugar, sobre el escenario **RMa_1_UE_rx**, se han obtenido los datos representados en la **Ilustración 42**; como puede apreciarse en los patrones obtenidos hasta este momento del estudio, **BBR** ha ofrecido grandes tasas binarias sobre los enlaces menos exigentes del trabajo, mientras que su rendimiento ha decrecido enormemente sobre los escenarios que contemplan más perdidas debidas a los obstáculos percibidos desde el punto de vista. Como cabe esperar en la **Ilustración 43**, se advierte una gran pérdida en la tasa binaria para **BBR**, esto supone un problema que resta puntos a la solución ofrecida.

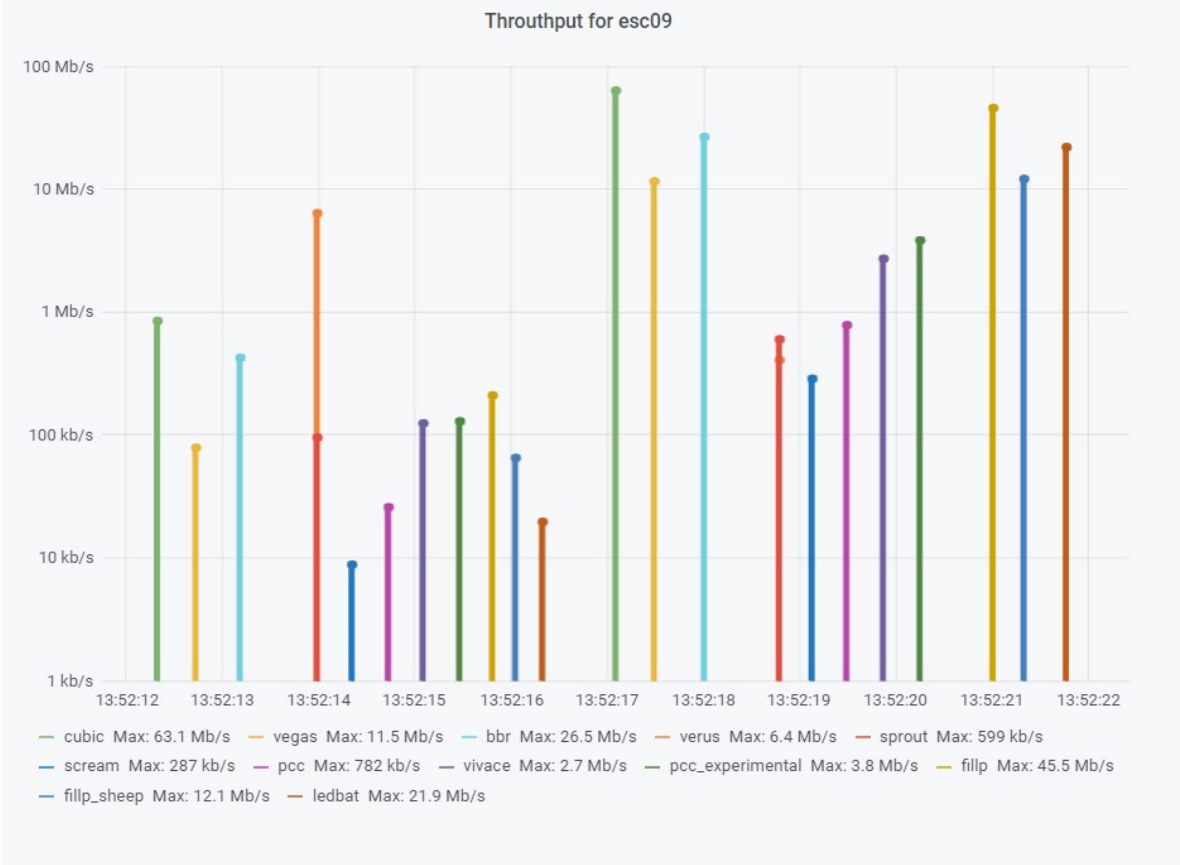

Ilustración 43: Esc09

Los datos obtenidos sobre el escenario representado por la traza **RMa_1_UE_rx**, muestran una velocidad de transmisión ínfima sobre los algoritmos **Scream** y **Sprout**; sin embargo, ha resultado ser un entorno satisfactorio para la mayoría de los algoritmos de control de congestión contemplados. Las tasas aceptables más bajas han descrito el comportamiento de los algoritmos, **Vivace**, **Pcc** y **Pcc_experimental**, que han obtenido un enlace capaz de transmitir datos con unas tasas binarias de entre los **2,8 Mb/s** y los **1,9 Mb/s**. Por encima de ellos se encuentra **Vegas,** con un throughput de **26,9 Mb/s,** y **Fillp_sheep,** con una tasa de **24 Mb/s**. A continuación, tal como se observa en la **Ilustración 42**, se encuentra el algoritmo **Fillp**, que una vez más ha superado a su segunda configuración ofreciendo una tasa binaria de **52,6 Mb/s**, un poco por debajo de la calidad ofrecida por esta conexión se encuentra **Ledbat** con unos **50,9 Mb/s**. Por último, cabe destacar el rendimiento de **Cubic**, con un throughput de **73 Mb/s**. Finalmente, el mejor rendimiento lo obtiene el algoritmo **BBR** con un rendimiento ligeramente superior a Cubic**.**

En cuanto al escenario **UMa_1_UE_rx**, representado sobre la **Ilustración 43**, sobre él se han obtenido unos resultados que una vez más confirman la adaptabilidad de ciertos algoritmos al entorno de despliegue, mientras que afecta negativamente sobre otros. Es el caso de **BBR** que, aunque ha obtenido una tasa binaria de **26,6 Mb/s** ha decrecido porcentualmente más que su principal competidor, **Cubic** que en este caso ha mantenido una tasa similar a la obtenida en experimento sobre el escenario **RMa_1_UE_rx**, en este caso el algoritmo ha obtenido un *throughput* de **63,1 Mb/s**.

Fillp ha conseguido el segundo puesto gracias a una tasa de **45,5 Mb/s**, lo que representa una disminución porcentual mucho menor que la del resto de soluciones.

Conclusiones

A lo largo de los últimos años se ha estudiado como los algoritmos de control de congestión actuales presentan serias carencias de rendimiento sobre canales inalámbricos, ya que interpretan las variaciones de capacidad del canal como perdidas. Si esto ha sido un problema con la mayoría de tecnologías inalámbricas, es de esperar que se acentúe con la llegada de las tecnologías sobre mmWave que aparecerán con los despliegues 5G.

A fin de mitigar esta problemática, se han propuesto multitud de soluciones de control de congestión. Sin embargo, estas son normalmente analizadas sobre escenarios concretos y de forma individual, lo que dificulta una comparativa justa. En este sentido, en este trabajo se ha desarrollado un entorno de evaluación sistemática de algoritmos de control de congestión, que permita la automatización de la misma y la obtención de unos resultados homogéneos.

El entorno se ha desarrollado explotando las herramientas, previamente existentes, MahiMahi y Pantheon y se ha adaptado su operativa para permitir la automatización de la evaluación. Sobre estas herramientas se ha desarrollado un entorno de visualización y análisis de resultados. Este desarrollo se ha llevado a cabo en Java, usando bases de datos basadas en series temporales (Influx) como persistencia y el entorno Grafana para la generación de paneles de control que permitan visualizar los resultados obtenidos.

El funcionamiento del entorno desarrollado se ha llevado a cabo mediante el análisis de rendimiento de 12 algoritmos sobre diversos escenarios de comunicaciones. En concreto se han usado trazas obtenidas con el simulador de red ns-3 para alimentar el entorno de evaluación. Estas trazas abarcan escenarios con obstáculos determinados (edificios), escenarios estadísticos rurales y urbanos, y escenarios en interiores.

A partir de esta evaluación se ha corroborado el correcto funcionamiento del entorno. Asimismo, se ha observado que la evaluación necesita gestionar grandes volúmenes de datos, por lo que se deben usar equipos que lo permitan. En cuanto a la comparativa entre algoritmos, se ha visto que en general los algoritmos evaluados no son capaces de explotar el potencial del canal. De entre ellos, destaca el comportamiento tanto de Cubic como BBR, que presentan, en general, rendimientos muy superiores al resto. También cabe mencionar Verus que, aunque no proporciona una tasa alta, muestra un comportamiento estable en los diferentes escenarios.

De cara al futuro, el entorno desarrollado podrá ser utilizado para extender el análisis presentado en el trabajo. En primer lugar, la herramienta permite analizar los algoritmos mencionados en el trabajo sobre otros escenarios sin realizar ningún cambio, siempre y cuando las trazas que representan los escenarios sigan el formato requerido. Asimismo, se podrá extender el conjunto de algoritmos, aunque esto requerirá cambios adicionales en el código de Pantheon. Finalmente, se podrá extender la funcionalidad del entorno gráfico, implementado sobre Grafana, para representar otros parámetros de rendimiento, tales como el *jitter*.

Bibliografía

[1] mmWave, «Springerlink,» [En línea]. Available: https://link.springer.com/article/10.1007/s11276-015-0942-z. [Último acceso: 2020 7 17].

[2] MahiMahi, «MahiMahi,» [En línea]. Available: http://mahimahi.mit.edu/#getting. [Último acceso: 2020 3 19].

[3] J. M. a. G. D. H. S. U. Francis Y. Yan, «Pantheon,» [En línea]. Available: https://www.usenix.org/system/files/conference/atc18/atc18-yan-francis.pdf. [Último acceso: 2020 3 13].

[4] Grafana, «Grafana,» Grafana Labs, [En línea]. Available: https://grafana.com/. [Último acceso: 11 5 2020].

[5] IEEE, «FDMA,» [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/975766?casa_token=qEiUVBpsPasAAAAA:KrGQwMtILolF3wNI99t5shfDDOsy9B58EBQpaPY8GWrkwfnPDww5pGGCJdeHMMHCErrYDdZAIg. [Último acceso: 7 3 2019].

[6] D. Library, «GSM,» [En línea]. Available: https://dl.acm.org/doi/book/10.5555/573838. [Último acceso: 21 7 2019].

[7] OnlineLibrary, «TDM,» [En línea]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4460110110. [Último acceso: 12 3 2019].

[8] OnlineLibrary, «GPRS,» [En línea]. Available: https://dl.acm.org/doi/book/10.5555/540485. [Último acceso: 23 6 2019].

[9] IEEE, «EDGE,» [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/772978?casa_token=LRhvKVDPz-

oAAAAA:euBWp-fc56CBR5d1vE4hlp2URhiPjEh9kWiIuwW1sWH8IM_2T-ToWWn8jwwnwOFfBl0xBJUNFg. [Último acceso: 23 4 2020].

[10] A. A. L. L. S. N. V. N. Heikki Kaaranen, «UMTS,» [En línea]. Available: https://books.google.es/books?hl=es&lr=&id=kX3Z9ss_5P0C&oi=fnd&pg=PR11&dq=UMTS&ots=D2BeXj_dFn&sig=MNSsNlldx1mylGicRG_MMI3fUuc#v=onepage&q=UMTS&f=false.

[11] IEEE, «CDMA,» [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/163765?casa_token=PwaP7ggRTGwAAAAA:GmOZeIlKyH_DJCsSvkTiTO5w0woj2ap94peAzcIGcBzzR_ad9EECKPmeByDsWGP7OZzD04y1cw.

[12] arXiv, «LTE,» [En línea]. Available: https://arxiv.org/abs/1110.1519. [Último acceso: 23 3 2019].

[13] Trhoughput, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Tasa_de_transferencia_efectiva. [Último acceso: 2016 6 23].

[14] BrainKart, «CC Bucles A C,» [En línea]. Available: http://www.brainkart.com/article/Congestion-Control--Open-Loop-and-Closed-Loop_13488/. [Último acceso: 25 4 2019].

[15] S. X. Serman L.Gavette, «RTT,» [En línea]. Available: https://patents.google.com/patent/US20080031136A1/en. [Último acceso: 21 6 2020].

[16] «machine learning aproach,» [En línea]. Available: https://books.google.es/books?hl=es&lr=&id=pxSM7R1sdeQC&oi=fnd&pg=PR9&dq=machine+learning+approach&ots=SNECOdV_Zn&sig=8-r6kLwi4qfJKwc71nRxFs7ixH4&redir_esc=y#v=onepage&q=machine%20learning%20approach&f=false. [Último acceso: 14 4 2020].

[17] M. L. G. F. R. Thomas R. Henderson, «NS-3,» [En línea]. Available: http://conferences.sigcomm.org/sigcomm/2008/papers/p527-hendersonA.pdf. [Último acceso: 4 5 2020].

[18] MathWorks, «Matlab,» [En línea]. Available: https://www.mathworks.com/products/matlab.html. [Último acceso: 23 4 2017].

[19] IEEE, «3GPP,» [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/5876496?casa_token=D89eJd0e8jYAAAAA:R5gH1yim8hFJqs_gzGnQISD4IDtNImhTE166JEC0Hia2yO0hXjmPacLen77smL21niX-RbwNyw.

[20] Wikipedia, «Timestamp,» [En línea]. Available: https://es.wikipedia.org/wiki/Marca_temporal. [Último acceso: 12 7 2019].

[21] R. B. A. S. Arthur Goldberg, «HTTP vs HTTPS,» [En línea]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3692&rep=rep1&type=pdf. [Último acceso: 23 11 2019].

[22] ACM, «CUBIC,» [En línea]. Available: https://dl.acm.org/doi/abs/10.1145/1400097.1400105. [Último acceso: 15 5 2019].

[23] ACM, «VEGAS,» [En línea]. Available: https://dl.acm.org/doi/abs/10.1145/190314.190317.

[24] Networks, «BBR,» [En línea]. Available: https://dl.acm.org/doi/pdf/10.1145/3012426.3022184. [Último acceso: 16 3 2020].

[25] SpringerLink, «SPROUT,» [En línea]. Available: https://link.springer.com/chapter/10.1007/978-3-540-30192-9_42. [Último acceso: 8 6 2020].

[26] ACM, «VERUS,» [En línea]. Available: https://dl.acm.org/doi/abs/10.1145/263932.264023. [Último acceso: 7 6 2020].

[27] AMC, «SCREAM,» [En línea]. Available: https://www.hindawi.com/journals/wcmc/2018/3142496/. [Último acceso: 15 5 2019].

[28] AMC, «WebRTC,» [En línea]. Available: https://dl.acm.org/doi/abs/10.1145/2910017.2910605. [Último acceso: 3 4 2020].

[29] UseNix, «PCC,» [En línea]. Available: https://www.usenix.org/conference/nsdi18/presentation/dong. [Último acceso: 8 7 2020].

[30] U. Mo Dong and Tong Meng y Doron Zarchy, «VIVACE,» [En línea]. Available: https://www.usenix.org/conference/nsdi18/presentation/dong. [Último acceso: 26 8 2019].

[31] M. D. a. Q. Li, «PCC_experimental,» [En línea]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong. [Último acceso: 2 9 2019].

[32] A. l. o. o. p. Turck, «FILLP,» [En línea]. Available: https://www.sciencedirect.com/science/article/pii/S108480451730231X?casa_token=NNGw30JXAw8AAAAA:3A82kKLs4kB_c_SUhi0fFVTLfq7f9INnz0ELi6ehxCSxmfYY5gzGsLJTWv70EkbI2EOAJWa5dQ. [Último acceso: 23 9 2020].

[33] IEEE, «LEDBAT,» [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/5560080?casa_token=QSvukmw25ogAAAAA:IJqVSWWDB_qnHRNx4LboQ-9a_qgwDQvXwvUm6qmMxTWZpcDLps_-qkwBsJY0kjS2vv0JSjXZNQ. [Último acceso: 14 6 2020].

[34] Influx, «Influx Data,» InfluxDB, [En línea]. Available: https://www.influxdata.com/. [Último acceso: 21 4 2020].

[35] ISO, «ISO.org,» [En línea]. Available: https://www.iso.org/ics/35.100/x/. [Último acceso: 2017 5 11].

[36] GuitHub. [En línea]. Available: https://github.com/.

[37] BitBucket, «BitBucket Repository,» [En línea]. Available: https://bitbucket.org/product/. [Último acceso: 9 11 2020].

[38] Bash, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Bash.

[39] «Python,» [En línea]. Available: https://es.python.org/. [Último acceso: 9 11 2020].

[40] MyEnTunnel, «MyEnTunnel,» [En línea]. Available: https://myentunnel.informer.com/3.4/. [Último acceso: 2020 3 3].

[41] C. T. F. T. T. A. &. D. L. Gede Artha Azriadi Prana, «Springerlink- ReadMe for Software Developers,» [En línea]. Available:

https://link.springer.com/article/10.1007/s10664-018-9660-3. [Último acceso: 2016 7 10].

[42] D. C. Felix Fuentes, «Ethereal vs Tcpdump,» [En línea]. Available: https://dl.acm.org/doi/abs/10.5555/1047846.1047873. [Último acceso: 2020 5 23].

[43] J. p. Lenguaje, «Google schoolar,» [En línea]. Available: https://thereaderwiki.com/en/Java_(programming_language). [Último acceso: 14 3 2017].

[44] E. f. developers, «Eclipse,» [En línea]. Available: https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers. [Último acceso: 2 13 2017].

[45] J. Casey, «Trabajo Maven,» [En línea]. Available: http://garbuz.com/download/books/better-builds-with-maven.pdf. [Último acceso: 23 7 2020].

[46] A. Spark, «Apache,» [En línea]. Available: http://acme.byu.edu/wp-content/uploads/2020/01/Spark.pdf. [Último acceso: 10 26 2019].

[47] D. J. Barrrett, «Google Schoolar,» [En línea]. Available: https://books.google.es/books?hl=es&lr=&id=JFa5aLIII6oC&oi=fnd&pg=PP11&dq=TCP+SSH&ots=gK4ZKCY-uR&sig=wqQmaj22H6eVo76aGlMj2xewhZY#v=onepage&q=TCP%20SSH&f=false. [Último acceso: 2020 7 13].

[48] I. Jacobson, «UML Papper,» [En línea]. Available: http://repositoriokoha.uner.edu.ar/fing/pdf/5933.pdf. [Último acceso: 23 2 2018].

[49] SFTP, «Google Scholar,» [En línea]. Available: https://patents.google.com/patent/US10469533B2/en. [Último acceso: 2020 4 17].

[50] A. SQL, «Expresiones regulares para Bases de Datos,» [En línea]. Available: https://solutioncenter.apexsql.com/es/como-usar-expresiones-regulares-regex-en-sql-server-para-generar-datos-de-prueba-aleatorios/. [Último acceso: 8 7 2018].

[51] J. M. K. Z. H. A. M. W. C. B. L. M. Weging, «SIOX - SpringerLink,» [En línea]. Available: https://link.springer.com/chapter/10.1007/978-3-319-07518-1_16. [Último acceso: 23 5 2020].

[52] S. K. G. Manda Sai Divya, «ElasticSearch- CompuSoft,» [En línea]. Available: https://d1wqtxts1xzle7.cloudfront.net/42044915/COMPUSOFT__26__171-175.pdf?1454594532=&response-content-disposition=inline%3B+filename%3DCOMPUSOFT_2_6_171_175.pdf&Expires=1609852464&Signature=bcS6xbYYj4kkX0Ag5w-1bhEErkgIUm5DqpV2L-wyNdql85j~T3NyzgqDX1748IC7. [Último acceso: 23 5 2020].

[53] V. J.-M. Désert, «FUSE Database,» [En línea]. Available: http://adsabs.harvard.edu/full/2006ASPC..348..547D. [Último acceso: 24 5 2020].

[54] V. Studio, «VisualStudioCode,» [En línea]. Available: https://code.visualstudio.com/. [Último acceso: 2020 2 3].

[55] R. Hat, «Linux,» [En línea]. Available: https://www.redhat.com/es/topics/linux. [Último acceso: 23 12 2018].

[56] J. A. G. G. Yihui Xie, «MarkDown,» [En línea]. Available: https://books.google.es/books?hl=es&lr=&id=octmDwAAQBAJ&oi=fnd&pg=PT17&dq=Markdown&ots=TvJ9r-ImeW&sig=ANevY_nCzFi0oelm-twVGEKEl_I#v=onepage&q=Markdown&f=false. [Último acceso: 2 7 2019].

[57] [En línea].

[58] NetSarang, «NETSARANG,» [En línea]. Available: https://www.netsarang.com/en/xftp/. [Último acceso: 2020 3 23].

[59] FilleZila, «FilleZila,» [En línea]. Available: https://filezilla-project.org/. [Último acceso: 2018 4 12].

[60] Json, «Google scholar,» [En línea]. Available: https://books.google.es/books?hl=es&lr=&id=ZYYnCgAAQBAJ&oi=fnd&pg=PP3&dq=json+format&ots=ahA0kWX_Bt&sig=Rq-cYOkiFm299xLV_1EPmZwKx38#v=onepage&q=json%20format&f=false. [Último acceso: 2020 4 13].

[61] R. Love, «Linux Kernel Development,» [En línea]. Available: https://books.google.es/books?hl=es&lr=&id=5BwdBAAAQBAJ&oi=fnd&pg=PR7&dq=linux&ots=rxI5Tqp3nh&sig=Ram7wp6xwCzCsRPF_2pkb0v-ryQ#v=onepage&q=linux&f=false. [Último acceso: 13 3 2016].

[62] RevistaDigital, «Parser all you must to know,» [En línea]. Available: https://revistadigital.inesem.es/gestion-empresarial/el-parseo-de-datos-en-internet-nuevas-oportunidades-comerciales-al-filo-de-la-legalidad/. [Último acceso: 21 12 2019].

[63] IEEE, «Non-standalone,» [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/9269939?casa_token=lwJNf6qexTcAAAAA:szgsEBAjK0rsj1eieXbEEyYd3Y3fPNt-Z-20mX_o-IYw3wWjvzc7KIGPxUgq1cd5MC8ZfRDGnQ.

[64] SpringerLink, «Java FX,» [En línea]. Available: https://link.springer.com/book/10.1007%2F978-1-4302-6461-3. [Último acceso: 3 2 2020].

Anexo I – Manual de Usuario

Este anexo está compuesto por dos subcapítulos, sendos subcapítulos describen tanto el entorno de trabajo remoto como el Entorno de desarrollo. Sobre ellos se expone la visión del desarrollador del despliegue, ayudando a la comprensión de los distintos procedimientos desplegados.

9.1 Entorno remoto de trabajo

Esta sección conforma el primer paso del desarrollo, pues toda la estructura desplegada funciona sobre un entorno remoto, formado por un servidor que desplegado en la Universidad de Cantabria aloja tanto los repositorios clonados para el uso de herramientas externas como el material desarrollado para automatizar los procesos y documentación digital sobre todos los pasos seguidos por este estudio. Cierta información fue clonada a un entorno local para asegurar un sistema distribuido tolerante a fallos o perdidas de la información.

Para establecer las comunicaciones entre el entorno de desarrollo y el computador local se escogió **MyEnTunnel** [8], esta herramienta permite al usuario básicamente, un método eficiente para establecer un túnel de comunicación TCP **SSH** [9], en el mercado actual existen multitud de herramientas que ofrecen este mismo servicio, pero esta herramienta en especial posee esenciales ventajas para el desarrollo del proyecto. En primer lugar, nos ofrece Plink (PuTTY Link), que se inicia en background para ofrecer un monitoreo efectivo de los túneles y si algún fallo ocurriera con Plink, el software lo reiniciaría automáticamente. El proyecto requiere de varios procesos corriendo en paralelo por lo tanto es indispensable que la herramienta encargada de desplegar el túnel no ocupe muchos recursos, MyEnTunnel ofrece como uno de sus sellos de identidad el bajo consumo de recursos, garantizando un uso mínimo de la CPU. La herramienta incluye una opción denominada sondeo lento, que solo cambia un contexto por segundo, además, para garantizar el cifrado la herramienta interna Plink se encarga del cifrado. En la **Ilustración 44**, se puede observar la ventana principal que nos da acceso al despliegue del túnel.

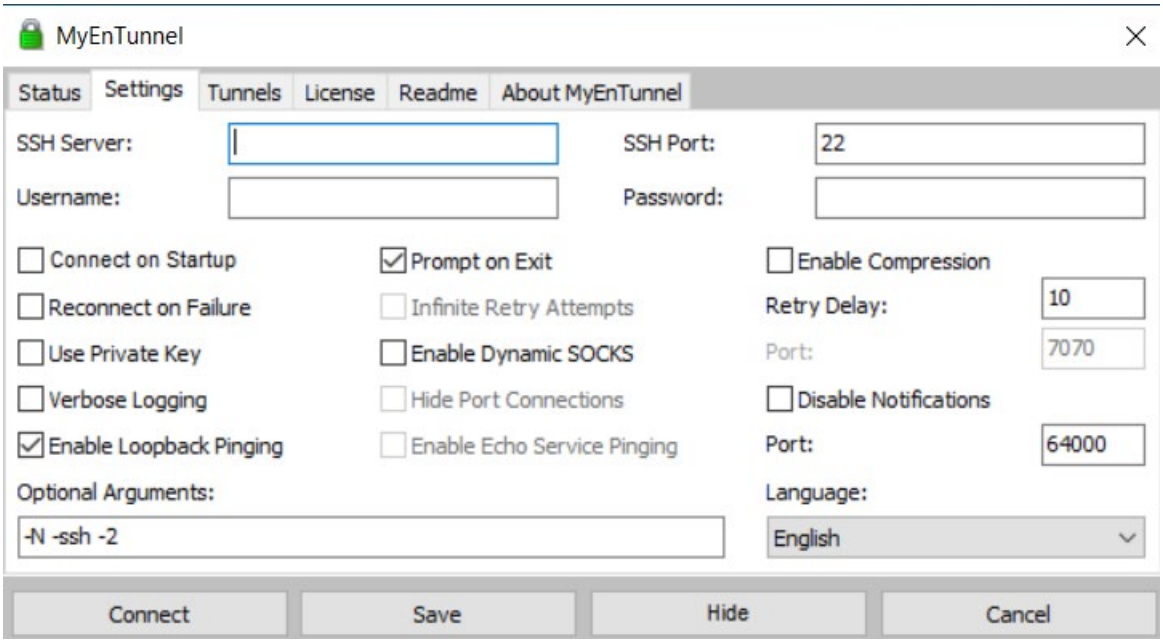

Ilustración 44: MyEnTunnel

Como se puede apreciar en la ilustración anterior la herramienta permite configurar el túnel aportando la información del destino, así como el usuario en caso de poseer acceso a varias particiones del mismo equipo

servidor. En logo representado en la **Ilustración 47** se muestra cuando la conexión se encuentra en el proceso de establecimiento de la ruta, en este paso el usuario introduce las credenciales correspondientes al equipo remoto; En segundo lugar, el logo mostrado en la **Ilustración 46**, representa el proceso de búsqueda del destino y establecimiento de la tubería. Por último, en la **Ilustración 45**, se muestra el logo que muestra que la conexión SSH ha sido finalmente establecida, siendo sustituido por el mismo candado en color verde en este momento.

Ilustración 45: Definiendo conexión

Ilustración 46: Estableciendo Comunicación

Ilustración 47: Conexión Establecida

Una vez desplegada la tubería que nos dará acceso remoto tanto de escritura como lectura es momento de desplegar otra herramienta que nos permitirá traer servicios remotos a nuestro entorno local, bien sean ficheros de resultados o una copia de seguridad que dote a la arquitectura de tolerancia ante fallos. La tecnología escogida en este caso para la transferencia de ficheros entre el entorno remoto y el local es **Xftp 6** [10], en la primera fase del desarrollo esta herramienta compitió con otros servicios de compartición de ficheros con entornos remotos, su principal competidor fue **FilleZilla** [11], ambas soluciones son FTP de código abierto, sin embargo las herramientas aportadas por Xftp 6 se ajustaban mejor a los requerimientos del sistema. Nuevas versiones de este servicio han sido implementadas recientemente por el equipo de desarrolladores software de la empresa, sin embargo, la versión utilizada en este proyecto sigue teniendo soporte actual, **Ilustración 48**.

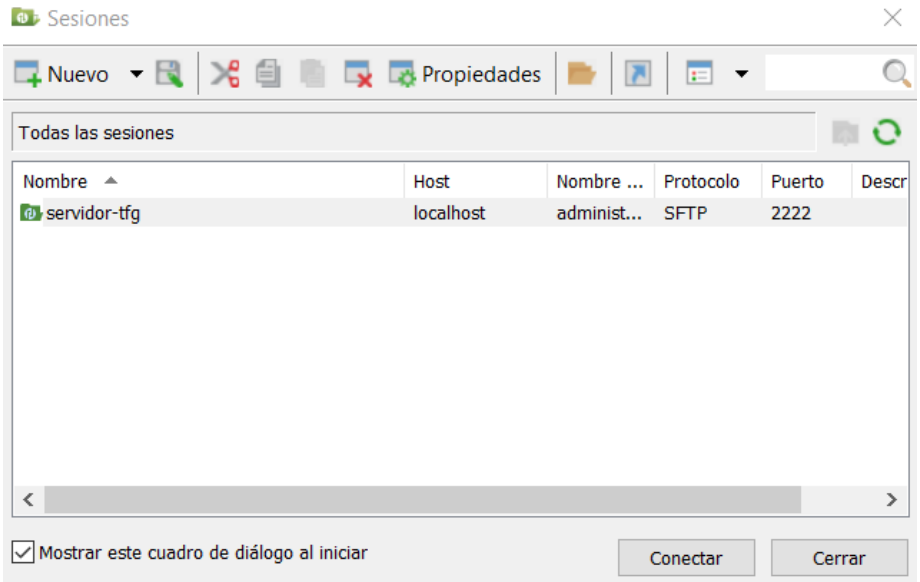

Ilustración 48: Xftp 6 - Agregar tunel

El primer paso en la configuración de este servicio es crear un túnel para la extracción y/o inserción de documentos sobre el entorno remoto, para ello en la **Ilustración 48**, se muestra la ventana que permite al usuario establecer el túnel, el ejemplo mostrado en la ilustración representa la ventana de configuración donde se

establece la ruta, este módulo está integrado en el túnel representado en la **Ilustración 22** descrita en la introducción del capítulo "**4. descripción de la solución Desarrollada**". Para esclarecer el proceso de conexión bajo las indicaciones de la herramienta se adjunta la **Ilustración 49**, en ella se representan los parámetros de configuración aportados para hacer posible la interacción con los ficheros remotos; cómo se puede observar la conexión se realiza por SSH como bien remarca el puerto escogido "2222".

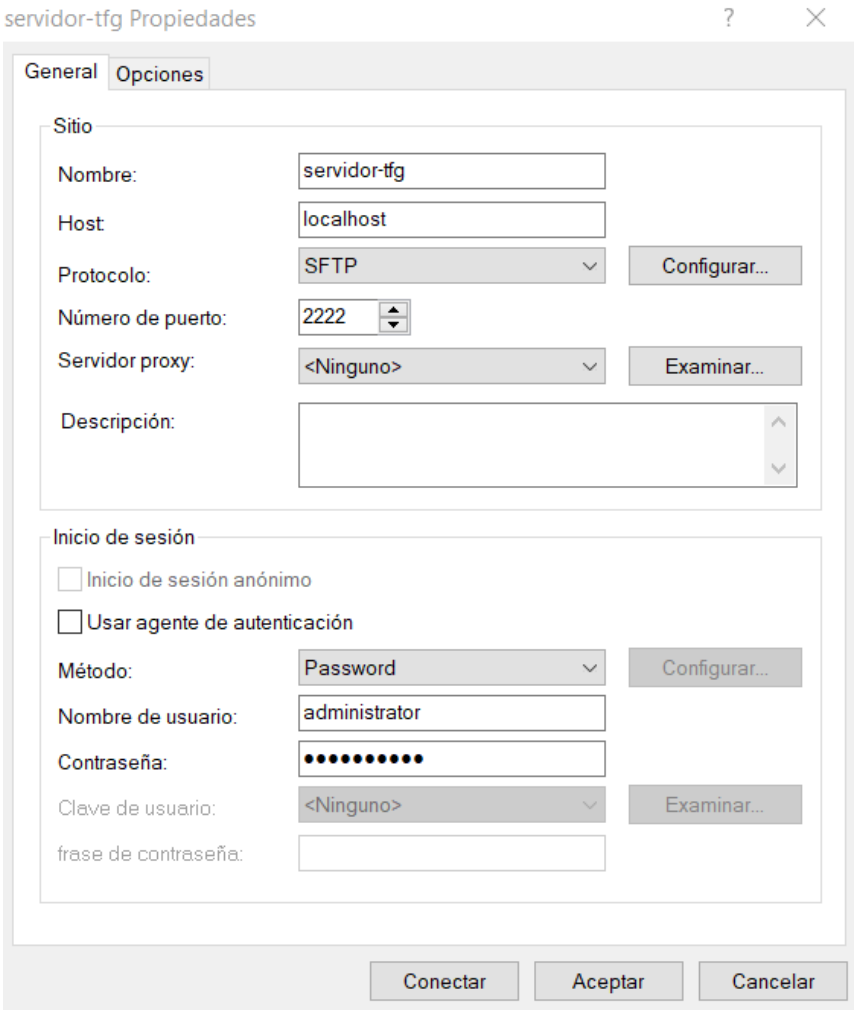

Ilustración 49: Configuración de la conexión Xftp 6

Una vez descritas las dos herramientas anteriores y deducida por tanto la arquitectura de trabajo remoto y como los componentes pertenecientes al sistema se interconectan, este subcapítulo requiere de descripciones adicionales. A continuación, se representan la estructura de datos y repositorios clonados dentro del servidor en la UC, para ello es indispensable introducir una tercera herramienta que permite cierra el círculo correspondiente al trabajo remoto; se trata de **Visual Studio Code** [12], esta herramienta permite la visualización del entorno remoto y facilita enormemente la programación de scripts y protocolos desde el entorno local.

Para permitir la conexión y poder obtener un entorno de acceso remoto, en el cual se pueda hacer acceso tanto a ficheros alojados en el servidor UC como la posibilidad de servirse de un control de root, que nos permitirá

tanto la escritura y lectura de los ficheros como la posibilidad de operar dentro de las terminales del repositorio del proyecto. Esta herramienta resulta totalmente indispensable para el despliegue de la aplicación; En el módulo nombrado entorno local en la **Ilustración 50**, se puede apreciar como el entorno de desarrollo software ha sido implementado sobre la herramienta **Visual Studio Code**, también podemos observar como en el módulo del equipo remoto aloja todos los repositorios utilizados para el proyecto, por lo tanto una vez configuradas las herramientas anteriores es hora de establecer la conexión con el equipo remoto y permitir el uso de comandos como administrador.

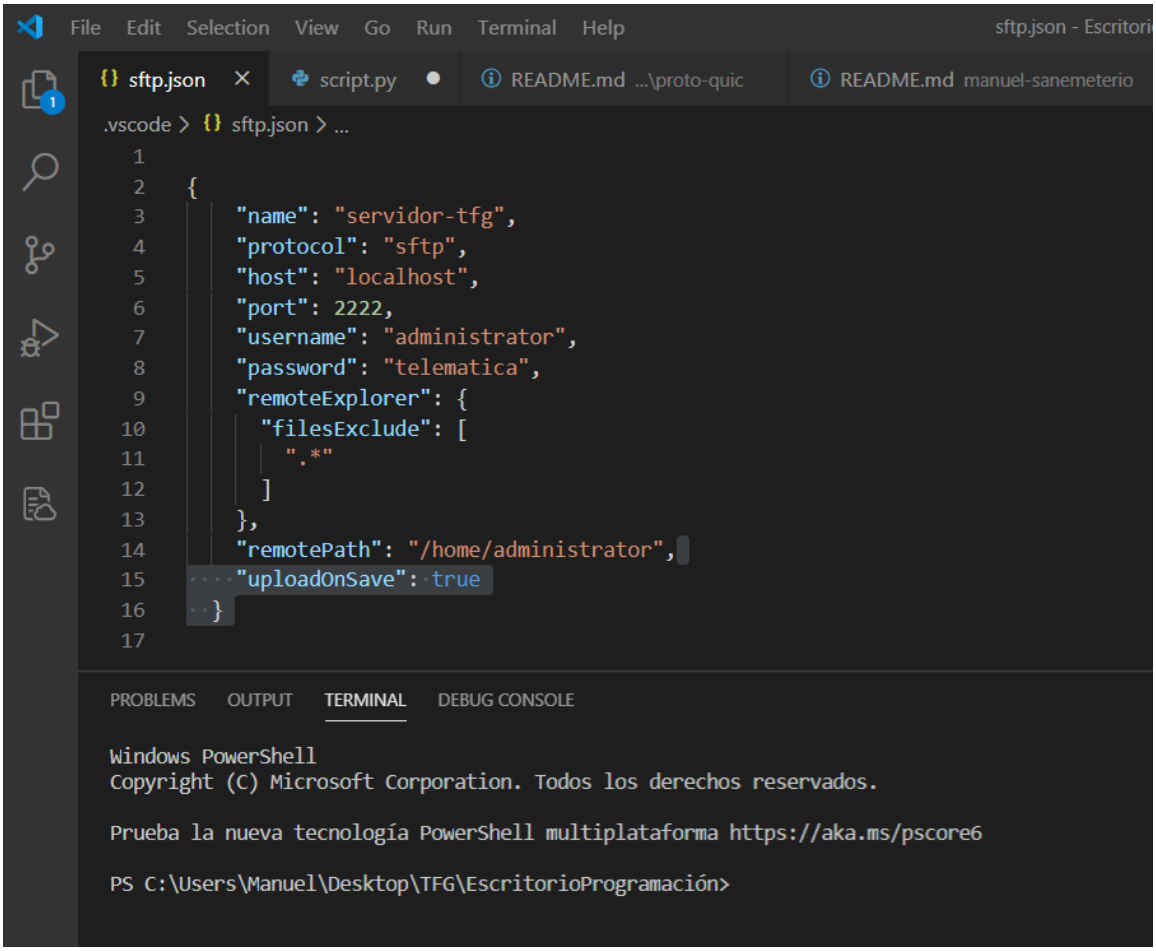

Ilustración 50: Json conexión sftp

Para habilitar la conexión entre la herramienta de **Visual Studio Code** y el servidor remoto será necesario crear un fichero **Json** [13], que enlace el contenido de la maquina Linux alojada en el servidor.

En la **Ilustración 50**, se muestra como fue definido el fichero de datos **Json**, que permitirá realizar una conexión segura **SFTP** [14], dicho fichero está compuesto por los campos reflejados en la ilustración nombrada, entre ellos como puede apreciarse se encuentran las credenciales que referencian a nuestro entorno remoto de trabajo, junto con el puerto "2222", que referencia a la conexión SSH, además las etiquetas **remote path** hace referencia a la dirección en a la que se accederá al hacer el acceso OTA (On The Air).

Para entender la funcionalidad de este submódulo integrado en el entorno local y que permite el despliegue de los servicios que se alojaran sobre la maquina remota, es necesario adjuntar una breve definición de las

funcionalidades de **SFTP**: esta tecnología permite la transferencia de archivos cifrados, por ese motivo el acrónimo añade la **S** al protocolo FTP, denotando que añade la capa de seguridad. Otros tipos de transferencia están también permitidos pudiendo auditarse, pero todos estos archivos pueden ser controlados en un firewall u cualquier otro tipo de puerta de enlace. Los archivos que, por lo tanto, hayan sido transferidos pueden estar sujetos al análisis de prevención de perdida de datos y/o controles de virus. En la siguiente figura: **Ilustración 20** aportada por el Paper: **Controlling and auditing SFTP file transfers**, redactado por **Tatu J. Ylonen y Samuel Douglas Lavitt**, se puede apreciar las capas de protocolos a los que se encuentra sujeto esta conexión:

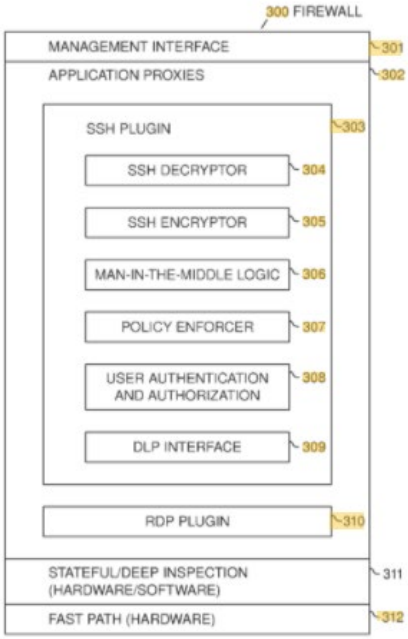

Ilustración 51: Capeado SSH conexión

Este documento además aporta datos relevantes acerca de los posibles escenarios en los que podemos desplegar el servicio SFTP [14], estas posibles configuraciones son las albergadas dentro de la conexión establecida, todas ellas poseen soluciones diferenciadas que permiten desplegar el servicio en función de la topología red en la que nos encontremos desplegando la aplicación. En la siguiente **Ilustración 51**, se muestran diferentes topologías de red, la mayoría representan casos de uso habituales y como se puede apreciar en la imagen todas ellas han de lidiar con el cortafuegos, por ello la herramienta centra sus esfuerzos en mantener una conexión fiable y dinámica capaz de recuperar la conexión en caso de ser perdida en mitad del proceso de desarrollo remoto.

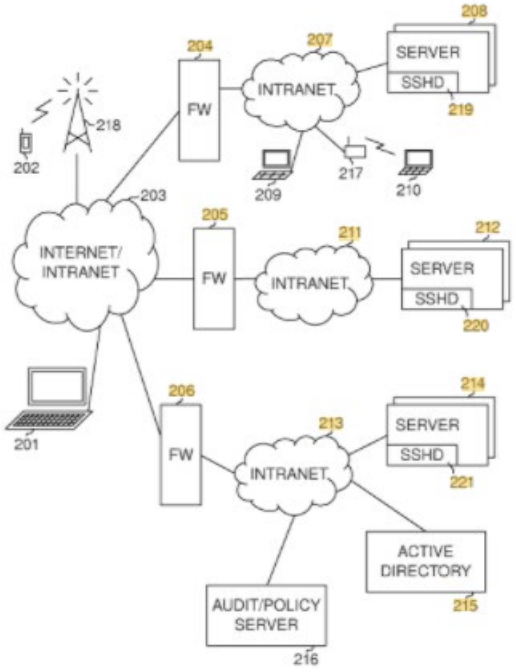

Ilustración 52: topologías de conexión SFTP

Una vez desarrollado e incluido el fichero **Json** [13] mostrado en la **Ilustración 52**, el siguiente paso para poder acceder al equipo remoto y permitir tanto el intercambio de ficheros, como su lectura, escritura y la posibilidad de operar en remoto sobre la terminal del servidor.

Para terminar con esta sección es imprescindible describir la forma de interacción con la herramienta descrita: **Visual Studio Code** [12]. Las herramientas descritas anteriormente proveen servicios que han resultado indispensables en el desarrollo de este proyecto, a pesar ello si este módulo ha de tener un núcleo este es claramente el entorno de trabajo soportado en **Visual Studio**; tanto la descarga como la subida de ficheros en el entorno remoto conformado por el servidor UC, quedan cubiertas por la herramienta **Xftp 6** [10], además el túnel que transportara toda la información se encuentra configurado gracias a la herramienta **MyEnTunnel** [8].

Una vez desplegadas estas herramientas el entorno de trabajo estaría completo, al menos desde el punto de vista de las capacidades requeridas, pero estas herramientas convierten el proceso de desarrollo software en una tarea ardua y compleja, es por ello que para facilitar la tarea más importante dentro del proyecto será necesario incorporar una herramienta que posea un poderoso entorno grafico que permita al usuario desarrollar las acciones remotas de una forma simplificada que se asemeje lo máximo posible a la situación convencional, en la que el desarrollador introducirá comandos u ficheros con código, scripts, etc; en un entorno local.

Para alcanzar el modelo anterior en este estudio se ha elegido la herramienta presentada al comienzo de esta sección, se trata de **Visual Studio Code** [12]. Una vez introducido el fichero Json que permitirá realizar la conexión con el servidor, podremos acceder con facilidad al entorno de trabajo remoto, en la siguiente **Ilustración 53** se puede apreciar como en el panel vertical desplegado en el marco izquierdo del programa se ofrecen ciertos servicios internos entre los cuales se encuentra en último lugar, remarcado en amarillo, el módulo

de conexión SFTP, al introducirnos en este panel, clicando sobre él, puede apreciarse como la opción remarcada con una estrella naranja representa el path desde el que accedemos al repositorio remoto, esto indica el root utilizado para la conexión. Al clicar con el botón izquierdo del ratón sobre esta opción se desplegará debajo el árbol de rutas que aloja todos los repositorios, scripts, ficheros, etc, utilizados para este proyecto. Si por el contrario clicamos sobre esta opción con el clic derecho del ratón se desplegará la opción subrayada en azul en la ilustración, al pinchar sobre ella podremos acceder a la terminal del servidor remoto permitiendo el uso de comandos, que en este caso como se define anteriormente serán nativos de **Bash** [5], pues el servidor remoto posee un sistema operativo de código abierto basado en Linux [15].

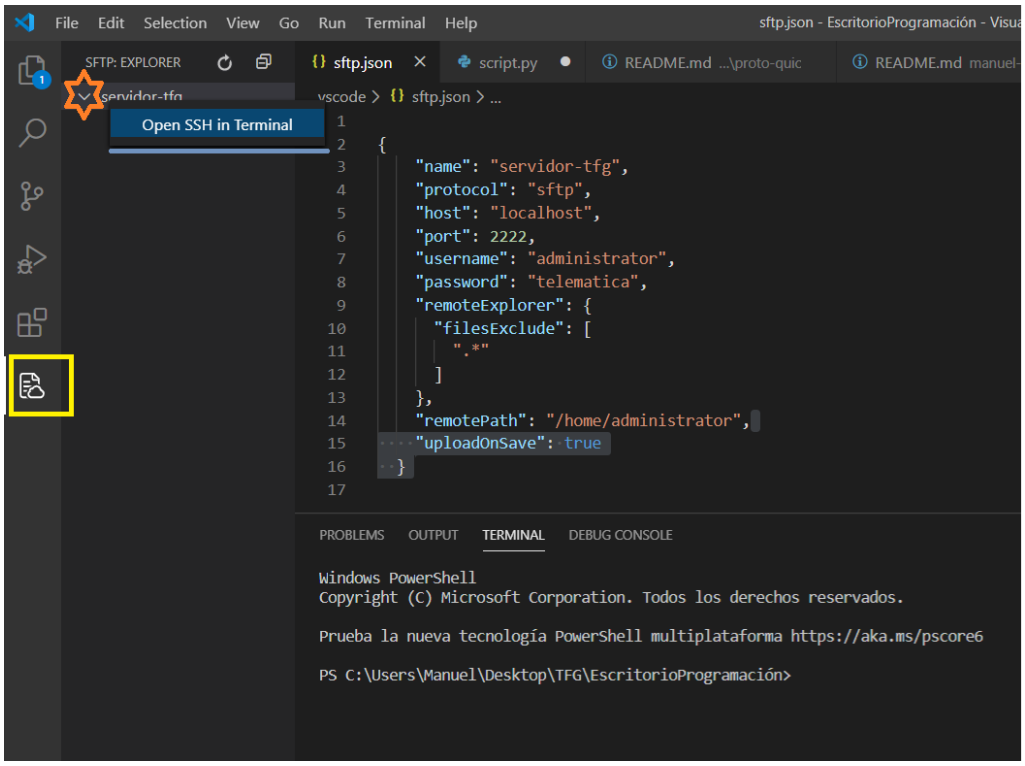

Ilustración 53: Abrir Terminal SSH

Una vez descrito, tanto el despliegue como las funcionalidades aportadas por las herramientas, que en conjunto aportan una solución de desarrollo remoto muy ajustada al propósito final del proyecto, este epígrafe llega a su fin y por lo tanto es el momento de pasar a la siguiente sección que conforma este subcapítulo, en ella se pretende aclarar tanto las funcionalidades como el desarrollo del entorno virtualizado de trabajo.

9.2 Entorno de Desarrollo

En este apartado se recogen los componentes que han permitido el despliegue del módulo denominado: Entorno de desarrollo, además, en él, se definen las herramientas requeridas para el desarrollo del estudio, tanto las herramientas adaptadas o customizadas, como las desarrolladas, implementadas y desplegadas únicamente para este proyecto, ambas han ido evolucionando a lo largo de la etapa de desarrollo e implementación, como en todo circulo de desarrollo software hasta alcanzar la versión final presentada en este documento.

Para comenzar es imprescindible entender que el entorno de desarrollo no conforma únicamente componentes locales como si de un laboratorio se tratase, sino que, por el contrario, las diferentes herramientas que lo conforman pertenecen tanto al entorno local como al remoto, y muchas de ellas enfocan sus esfuerzos en la mera sincronización o comunicación. El despliegue de todo el "**Repositorio del Proyecto**", apreciable en la **Ilustración 54**, ha sido posible gracias a la herramienta **Visual Studio Code** [12], es por ello por lo que en el proyecto ha sido nombrado "*el corazón del despliegue*". Generalmente, el entorno de desarrollo cuenta con capacidades superiores, requeridas para el buen funcionamiento del sistema global, en el caso de este proyecto no podía ser diferente, y entre otros parámetros la memoria en disco ha sido aumentada varias veces para poder dar cabida al gran número de paquetes, records, y ficheros que lo conforman, la escalabilidad por tanto ha sido un parámetro indispensable para su desarrollo; Dado el elevado número de escenarios, trazas, y algoritmos fue previsible desde el inicio que la cantidad de métricas o paquetes recogidos en cada uno de los conjuntos de cada estudio crecería exponencialmente, más todavía al encontrarnos muestreando redes móviles de quinta generación cullos valores de **Throughput** [20] son capaces de llenar *buffers* completos en cuestión de segundos.

El proyecto está distribuido en forma de árbol en un servidor Linux [21], tanto para la programación como para el acceso a ficheros se ha hecho uso de la herramienta **Visual Studio Code** [12], que facilita sustancialmente el trabajo sobre equipos remotos, ofertando un entorno grafico sencillo y altamente configurable, que sumado al túnel desplegado previamente mediante el uso de **MyEnTunnel**, permite al desarrollador trabajar sobre entornos remotos como si se tratase del propio equipo local.

En este apartado se define la estructura en la que se ha desarrollado el proyecto para así entender y facilitar un posible uso futuro en el que la herramienta desplegada conforme los cimientos de nuevos análisis y sus respectivos desarrollos en el campo de la telecomunicación de nueva generación. En primer lugar, el proyecto se desarrolló desplegando tres subdirectorios en los que se encuentran distribuido el desarrollo en totalidad. El primero de ellos se denomina ***mahimahi***, y como su propio nombre indica en él se alojan tanto las librerías como archivos de configuración **Python** [6], documentos *Readme* y demás ficheros que conforman el repositorio; para el posterior uso de este proyecto es importante no deshacerse de esta carpeta y sustituirla por el repositorio original pues en ella se han realizado algunos cambios que han permitido el engrane de todas las herramientas. El segundo directorio que conforma el proyecto toma el nombre del desarrollador y está a su vez dividido en dos documentos de vital importancia, el primero de ellos se trata de la carpeta que contiene tanto el repositorio de **Pantheon**, como los scrypts, ficheros de configuración y demás documentos que conforman el proyecto, por lo tanto esta segunda directiva conforma la mayoría de la materia del estudio, es por ello que el segundo ítem, introducido en el directorio de desarrollador, es un documento de texto definido en lenguaje **MarkDown** [22], que trata de clarificar el tanto el objetivo del desarrollo como aportar fragmentos de código, consejos y directivas que permitirán al lector aclarar sus conceptos y manejarse de una forma sencilla e intuitiva sobre este. Por último, la tercera raíz del proyecto está formada por una carpeta que contiene documentos de texto plano, estos ficheros son las trazas definidas en el apartado "**3.1.2 Trazas**", y conforman la entrada y parametrización del canal **MahiMahi**, en el capítulo siguiente se definirán las líneas que conforman estos documentos pues en ellas se aporta la información del canal de manera simple y concisa.

Para facilitar la comprensión del despliegue en la siguiente **Ilustración 54**, se muestra la representación de directorios:

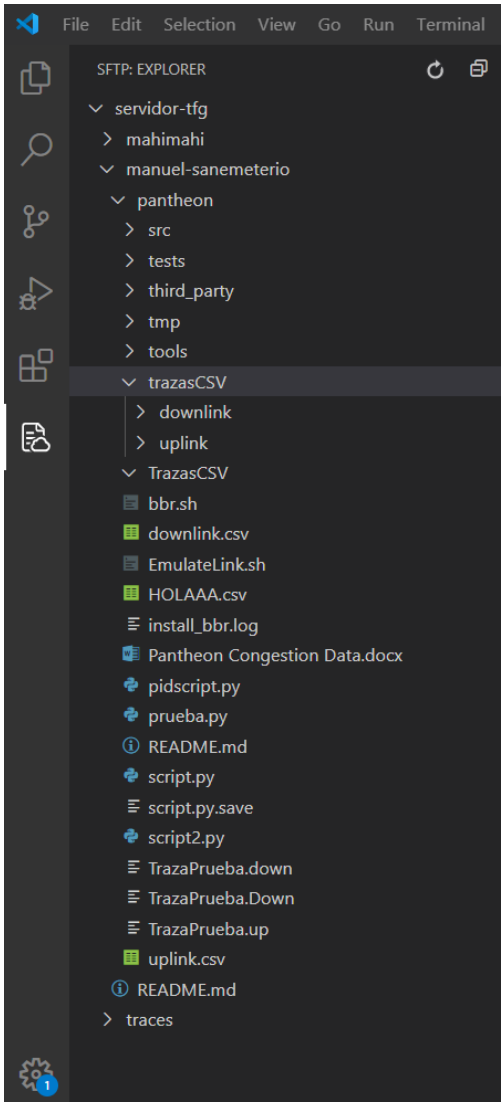

Ilustración 54: Estructura de directorios

Como se puede apreciar en la imagen anterior el directorio denominado "**pantheon**", engloba la mayoría de las herramientas, tanto utilizadas como desarrolladas para el proyecto, entre ellas cabe destacar que la carpeta remarcada en la imagen y nombrada "**trazasCSV**", contiene los resultados o capturas desempeñadas por **MahiMahi** y *parseadas* [23] en fichero CSV para su posterior análisis y tratamiento. La carpera como se puede apreciar en la imagen esta subdividida en dos subcarpetas que conforman los ficheros obtenidos mediante el esnifado del canal descendente o ascendente respectivamente. Otro documento que resulta de vital importancia y que se encuentra en este nivel de profundidad es el documento denominado **script.py**, se trata de un código Python [6] que se encarga de automatizar el proceso de recolección de datos, desarrollado específicamente para este estudio este código permite mediante el uso de multiprocesos, subprocesos y tuberías el despliegue total del entorno de desarrollo mostrado en la **Ilustración 54** anterior; cabe destacar que este código fue precedido por un autómata desarrollado en **Bash** [5], y aunque ambos lenguajes aportan la sencillez y versatilidad de poder ser ejecutados desde cualquier terminal por línea de comandos sin necesidad de un compilador externo, Python ofrece numerosas herramienta con respecto al uso de hilos y procesos, términos imprescindibles en la

programación orientada a objetos y más aún cuando tratamos de automatizar una serie de comandos que no son más que procesos que pueden requerir de más de un terminal, o incluso de comandos que los hagan trabajar en **backgound** y así permitan al desarrollador el despliegue de otra parte de la arquitectura.

Para concluir con esta breve explicación modular del "***Entorno de desarrollo***", se aporta otra imagen en la que se puede apreciar el alojamiento de otra de las principales herramientas desplegadas para el desarrollo del proyecto, se trata de la carpeta fuente o "**src**", mostrada en la siguiente **Ilustración 55**. En la ilustración se aprecia como al desplegar esta carpeta varios subdirectorios serán presentados, cada uno de ellos contiene valiosos archivos de configuración y codificado, que guardan relación con las herramientas ofertadas por **Pantheon**. En este apartado no se especificará el contenido de cada una de ellas, pero la información acerca de las misma viene definida de nuevo en un documento **ReadMe**, que al igual que el presentado anteriormente se encuentra descrito mediante el lenguaje de etiquetado **MarkDown**, y que presenta simples descripciones acerca del contenido desplegado en cada uno de los subdirectorios del repositorio de la herramienta.

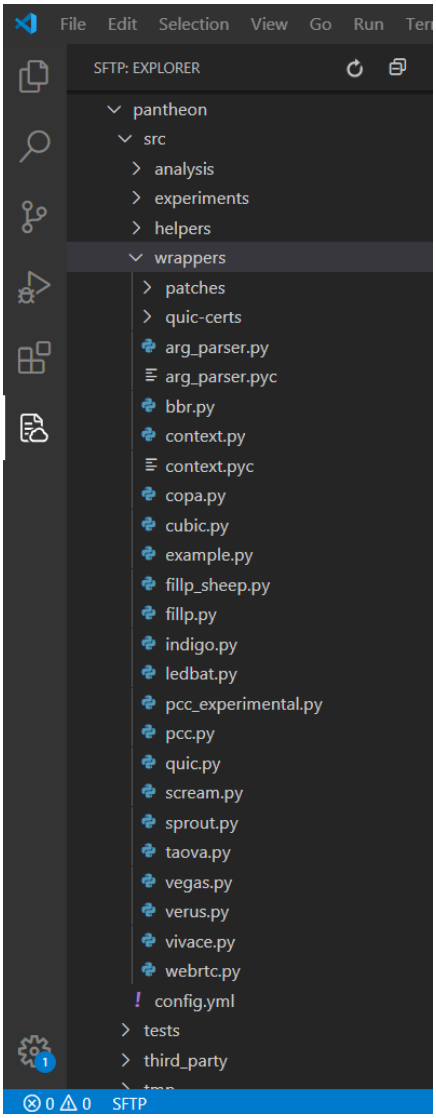

Ilustración 55: codificación de las métricas

En la **Ilustración 55** por lo tanto se puede apreciar que en la carpeta **src** especificada anteriormente aparece un subdirectorio remarcado en gris y denominado "***Wrappers***"; este subdirectorio se compone de diversos códigos descritos en lenguaje **Python** y que aportan cada una de las diferentes configuraciones y posibles conexiones para cada algoritmo de control de congestión contemplado en el estudio.

Como se puede apreciar en la imagen todos los algoritmos poseen un documento "**.py**" que lo define como un objeto compilable, en este caso todos estos documentos albergan los mecanismos de conexión, por lo tanto, mediante el uso de los comandos que se describirán en mayor detalle en el siguiente capítulo "**5. Resultados**", el repositorio desencadenara los diferentes procesos descritos en estos documentos mediante los cuales se establecerá una nueva conexión, en la que el protocolo introducido por comando desempeñara la labor de control de congestión.

Para concluir con este apartado es fundamental la mención de otro de los documentos relevantes aportados por **Pantheon**, se trata de la última carpeta legible en la **Ilustración 55**, y denominada "***thrird_party***", en ella se encuentra el conjunto de algoritmos utilizados en el desarrollo, pero esta vez los documentos aportados son el servidor y cliente a desplegar; mediante el uso de los comandos mencionados anteriormente se desplegara el servicio que interactuara con estos documentos y mediante dos sendos comandos se especificara el extremo que se quiere desplegar, en esta carpeta se encuentra el código descrito, en lenguaje **Python**, con el que se desplegara dicho *peer* en la comunicación.

Una vez concluida esta sección, el siguiente punto que se trata en este proyecto es la inyección y tratamiento de los datos, se podría referir a estas dos secciones como etapas a la vez que módulos, el orden elegido permite clarificar el flujo del proceso de investigación realizado.