Implementación de un mecanismo de asignación de recursos en dispositivos WiFi

Ignacio Prandi Reyes
Juan Francisco Pérez Pereira

Tesis presentada en cumplimiento de los requerimientos para la obtención del título de Ingeniero en Computación

TUTORES TESIS:
Dr. Javier Baliosian, Universidad de la República
Dr. Matías Richart, Universidad de la República

Montevideo, Uruguay
27 de mayo de 2021

Resumen

La segmentación o partición de la red (network slicing) es una de las tecnologías claves para la adopción de las nuevas redes 5G. Esto permite a los proveedores de infraestructura asignar recursos a diversos proveedores de servicio (arrendadores de la red) los cuales usarán estos recursos para satisfacer las demandas de sus usuarios finales.

Este paradigma, que cambia la manera en que las redes han sido tradicionalmente gestionadas, fue propuesto inicialmente para las redes cableadas (núcleo de la red). Recientemente, la comunidad científica ha puesto su atención en la integración del network slicing a las redes inalámbricas. Obtener una solución de slicing para redes inalámbricas y en particular para redes WiFi no es trivial. En este proyecto se propone implementar el algoritmo ATERR, de asignación de recursos en puntos de acceso (APs) WiFi para lograr slicing.

Adaptive Time-Excess Round Robin (ATERR) es un algoritmo de scheduling de paquetes cuyo objetivo es garantizar una determinada asignación del tiempo de transmisión (airtime) disponible en un AP WiFi. Este algoritmo ha sido recientemente propuesto, ha sido estudiado analíticamente y en simulaciones pero no existe una implementación en hardware del mismo. En particular se propone implementar el algoritmo para tarjetas de red Atheros y utilizando el driver ATH9K el cual está totalmente implementado en software como parte del kernel del sistema operativo Linux.

El objetivo principal del proyecto es implementar el algoritmo ATERR en el driver ATH9K de tarjetas de red inalámbricas Atheros.

Índice general

Índice de figuras

Glosario

5G Redes de celulares de 5ta generación.

Access Point (AP) Dispositivo que permite conectar nodos inalámbricos a la red.

Adaptive Time-Excess Round Robin (ATERR) Algoritmo de asignación de recursos Round Robin.

Airtime Tiempo de uso de la red.

ATH9K Driver de red inalámbrica para tarjetas Atheros.

Handshake Protocolo de conexión para la comunicación.

Internet Of Things (IoT) Práctica consistente en la interconexión de objetos cotidianos.

iperf3 Herramienta para pruebas de rendimiento en redes.

Kernel Programa que implementa las operaciones de sistema en Linux.

Mega Bit Per Second (Mbps o Mbit/s) Unidad de transmisión de mega bit por segundo.

Nodo (cliente) Equipos conectados al AP.

Quantum Valor de tiempo de red asignado por el planificador.

Ratio Proporción de quantum asignado.

Round Robin (RR) Algoritmo de planificación por turnos.

Secure Shell Protocol (SSH) Protocolo de acceso remoto seguro.

Slice Hace referencia a una parte o rebanada de la red.

SSID (Service Set Identifier) Nombre de la red inalámbrica.

Traffic Identifier (TID) Identificador de tráfico en redes inalámbricas.

Transmission Control Protocol (TCP) Protocolo de comunicación orientado a la conexión.

Type Of Service (TOS) Tipos de servicio de red.

User Datagram Protocol (UDP) Protocolo de comunicación sin conexión.

VAP (Virtual Access Point) Punto de acceso virtual.

Wireless Fidelity (WiFi) Nombre con el que es normalmente conocido el estándar IEEE 802.11.

Wireless Local Area Network (WLAN) Sistema inalámbrico de interconexión entre dispositivos.

XMIT Abreviatura de transmisión.

Capítulo 1

Introducción

El network slicing (o rebanado de la red) define un paradigma donde se particiona la infraestructura física de la red en múltiples partes o rebanadas lógicas (slices) que cumplan ciertas características de red. Dichas slices componen una arquitectura de red virtual que permite asignar recursos de forma dinámica y a demanda para satisfacer requerimientos específicos de servicio.

El concepto de network slicing se puede asociar directamente a la implementación de redes virtuales sobre una infraestructura física común. La virtualización de los APs (Access Point) es transparente para los distintos clientes, cada uno se podría conectar a un VAP (Virtual Access Point) con su propio SSID. Los recursos del AP se pueden dividir en distintas slices en base a la capacidad de transmisión o el ancho de banda del mismo. Todos los VAP comparten el mismo canal de transmisión ya que se encuentran en el mismo dispositivo, por lo que la suma de los recursos de los VAPs será igual a los recursos del AP al que pertenecen. A su vez, para un mismo cliente pueden existir diferentes slices según los flujos de tráfico, los cuales se pueden discriminar según las direcciones IP o los puertos de origen y destino, o el tipo de tráfico, por ejemplo.

Este trabajo se basa principalmente en implementar el algoritmo Adaptive Time-Excess Round Robin (ATERR) presentado por Matías Richart, Javier Baliosian y colaboradores, en el artículo "Slicing in WiFi Networks Through Airtime-based Resource Allocation" [1]. ATERR es un algoritmo de planificación de tipo Round Robin (RR), donde el quantum asignado a cada slice en la ejecución del algoritmo determina el porcentaje de recursos asignado para las slices del AP. Los quantums deben adaptarse dinámicamente dependiendo de las slices activas que haya, los clientes conectados y el flujo de transmisión requerido por cada uno. Se pretende que el porcentaje de recursos asignado para las distintas slices

de un AP se pueda gestionar por un administrador de red mediante una aplicación.

1.1. Objetivos

En primer lugar el objetivo es obtener una implementación funcional del algoritmo ATERR sobre una tarjeta de red Atheros que utilice el driver ATH9K. El hecho de que este driver se encuentre implementado en software en su totalidad y dentro del kernel de Linux, avala su elección porque facilita el proceso de trabajo, además de ser comúnmente utilizado en gran cantidad de Access Points. En segundo lugar se busca crear una aplicación de usuario que facilite la gestión del slicing sobre los APs. Para esto se deberá implementar la comunicación del modo usuario con el modo kernel en el AP, para que las slices generadas y los valores de transmisión asignados a los mismos puedan verse reflejados en los quantums asignados a cada slice en la implementación del algoritmo. Esto se logra mediante el envío de parámetros configurables al kernel del AP.

1.2. Contribución

Como resultado de este trabajo se presenta una implementación de ATERR en el kernel de Linux y una aplicación de usuario para poder realizar pruebas en un ambiente real. Tanto el código resultante como todo el material relacionado al proyecto, se encuentran disponible de forma libre y accesible a través de GitLab [2]. Allí se encontrarán guías de ayuda para facilitar el trabajo, que explica paso a paso como compilar el kernel de Linux, o configurar un equipo en modo AP, por ejemplo. En el anexo A.1 se describen las distintas secciones que se pueden encontrar en el repositorio GitLab.

1.3. Estructura

El presente documento se encuentra estructurado de la siguiente manera. El Capítulo 2 presenta un contexto global del trabajo, con el fin de entender el problema que busca resolver el algoritmo ATERR. Incluye descripción y antecedentes del trabajo, así como algunos conocimientos necesarios de redes WiFi. En el Capítulo 3 se presenta el algoritmo ATERR en profundidad, destacando el algoritmo base que sustenta el trabajo. El Capítulo 4 presenta información referente al desarrollo propiamente dicho, tanto del algoritmo como de la interfaz de usuario. En el Capítulo 5 se presenta toda la información referente a las

pruebas realizadas. Se incluyen los resultados obtenidos para un conjunto de pruebas que se consideran interesantes, y se destacan pruebas a realizar en un futuro proyecto. Finalmente los capítulos 6 y 7 incluyen las conclusiones que arrojó el trabajo y las lecciones aprendidas.

1.4. Evolución del proyecto

Para llevar a cabo el presente proyecto, se definieron distintas etapas y se planteó una modalidad de desarrollo iterativa incremental. Las etapas principales que se definieron son:

- Estudios previos
- Definición de ambiente de trabajo
- Análisis
- Diseño
- Implementación
- Pruebas
- Documentación

En la modalidad planteada las etapas se encuentran solapadas ya que se se entiende que la naturaleza del proyecto requiere mantener presentes etapas anteriores mientras se trabaja en las siguientes. Esto ocurre en particular con las etapas de diseño e implementación, donde definir un diseño robusto sin involucrarse en la propia implementación es poco factible. Dentro de cada etapa se definen tareas y objetivos con distintas fechas de comienzo y fin aproximadas.

El seguimiento del cronograma se vio afectado a partir de los primeros meses, donde fue necesario replantear nuevas fechas para las etapas de diseño y posteriores. Dificultades planteadas fundamentalmente al momento de definir y configurar un ambiente de trabajo apto para el desarrollo del proyecto, fueron las causantes de no llegar a las fechas inicialmente definidas para las etapas. A esto se le suma una estimación errónea en la carga horaria para algunas de las tareas planteadas, por ejemplo el estudio y el análisis de los trabajos e implementaciones ya existentes, en los que se basa el desarrollo del proyecto actual.

Capítulo 2

Contexto del problema

2.1. Descripción

Las redes WiFi 802.11 se han convertido en las más utilizadas debido a su practicidad, bajo costo y alto rendimiento. Cada vez son más los equipos que se conectan de forma inalámbrica y la inminente llegada de las redes 5G, con un gran incremento en la velocidad de transferencia, propicia la necesidad de controlar la asignación de recursos en la red para un mejor aprovechamiento de la misma. La segmentación de la red o slicing, permitirá a los proveedores del servicio, gestionar el negocio de forma análoga al manejo de las redes cableadas. [3]

Una de las posibles formas de implementar slicing es gestionando el tiempo de transmisión (airtime) de los clientes. Esta forma de implementación es la estudiada en el presente trabajo, ya que es la elegida por el artículo en que se basa el proyecto. Es en este contexto que se busca implementar un algoritmo que permita asociar un valor de airtime a cada una de sus slices, de forma que cada slice obtenga un porcentaje del total de los recursos disponibles del AP, para la transmisión de paquetes.

El algoritmo de planificación de paquetes ATERR busca que el punto de acceso (AP) garantice el tiempo de transmisión asignado a cada cliente conectado, según corresponda. La implementación de dicho alogritmo se llevará a cabo íntegramente a nivel de software, dentro del kernel de Linux y específicamente en el driver ATH9K, encargado del funcionamiento de las tarjetas de red inalámbicas Atheros [4]. La asignación de recursos a los clientes se busca que pueda ser controlada por el administrador de la red de forma directa desde una interfaz. Para lograr acoplar este requerimiento, junto a la implementación del

algoritmo en la estructura de desarrollo ya existente, se deberán estudiar las distintas capas que componen la implementación del kernel Linux y sus drivers.

2.2. Conceptos

2.2.1. Estándar IEEE 802.11

Actualmente el estándar IEEE 802.11 (mas conocido como WiFi) es el estándar de facto en las redes de tipo WLAN, permitiendo interconectar diversos dispositivos mediante la transmisión de ondas de radio. Dicho estándar especifica las capas de enlaces y físicas para redes WLAN.

Las redes de tipo IEEE 802.11 pueden trabajar en alguno de los siguientes modos:

- Infraestructura: se cuenta con un nodo central, comúnmente llamado AP (Access Point), el cual recibe las tramas de los nodos y es el encargado de enviar las tramas al destinatario.
- Ad-hoc: en este tipo de redes no existe un nodo central como el AP y los terminales se comunican entre ellos de forma directa.

Dentro del alcance del presente proyecto, se encuentran únicamente las redes de tipo IEEE 802.11 que trabajan en el modo infraestructura, ya que ATERR fue desarrollado para implementarse sobre el nodo central AP [5].

2.2.2. Redes 5G

Se espera que las redes 5G aumenten no sólo la velocidad de transferencia de las redes móviles, sino también la escalabilidad, conectividad y eficiencia del consumo de energía de las mismas. Para 2023 se proyecta que cerca de 30 mil millones de dispositivos se encontrarán conectados a Internet, permitiendo que IoT se vuelva masivo en todo el mundo [6]. La baja latencia provista por 5G es fundamental para soportar tal aumento en la cantidad de dispositivos conectados, además de proveer interactividad en tiempo real para los servicios que utilizan la nube, como pueden ser los vehículos autónomos, por ejemplo. Además, el bajo consumo de energía va a permitir que los objetos conectados funcionen por más tiempo sin intervención humana.

La gran variedad de servicios disponibles para millones de dispositivos, podrá ser aprovechada de mejor forma gracias a la posibilidad de utilizar subredes, con el fin de proporcionar conectividad más ajustada, en términos de ancho de banda, velocidad y latencia, a las distintas necesidades concretas. Como caso particular interesante, desarrollar una subred exclusiva dedicada a los servicios de emergencia, podrá asegurar la disponibilidad a estos servicios en todo momento. Por último, también se abren nuevas puertas para distintas vías de negocio para los operadores de red. En este contexto, WiFi se plantea como una opción viable para ofrecer servicios cercanos al cliente y descomprimir la carga de las redes celulares. Por esto se considera que la tecnología WiFi es parte del mundo 5G [7].

2.2.3. Slicing

Uno de los pilares fundamentales para construir las redes 5G es el denominado network slicing, que refiere a una segmentación de la red, adecuando el tamaño de la conectividad para los distintos casos y permitiendo la implementación de subredes. Este concepto permite proporcionar diferentes comportamientos de red, referidos como network slices, en términos de funciones y prestaciones sobre una misma red 5G. El diseño de una network slice puede obedecer al soporte de una aplicación específica que demande un comportamiento concreto de la red, o la necesidad de proporcionar un servicio que cumpla ciertas garantías para un grupo de usuarios [8].

Concretamente, una slice (o segmento de red) es una red lógica de extremo a extremo independiente, que se ejecuta en una infraestructura física compartida. Cada slice comprende recursos dedicados y/o compartidos, por ejemplo, en términos de potencia de procesamiento, almacenamiento y ancho de banda, y puede abarcar varias partes de la red, que requieran implementarse a través de múltiples operadores. Es importante que la división de la red sea transparente para los usuarios y que cada segmento de red tenga aislamiento del resto. La Figura 2.1 muestra el uso de network slicing en redes 5G para gestionar distintos servicios [7].

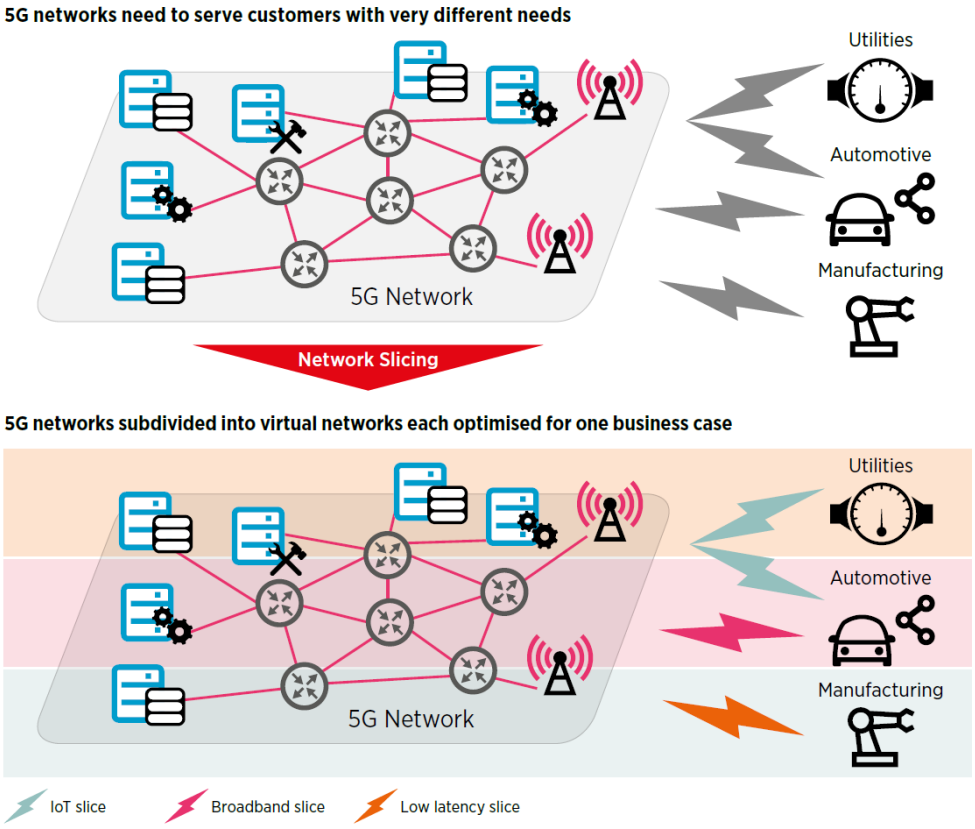


Figura 2.1: Diagrama de network slicing. Fuente [7]

2.2.4. ATH9K

ATH9K es un driver para controladores WiFi de una gran variedad de dispositivos desarrollados por la empresa Qualcomm Atheros. Dicho software se encuentra bajo una licencia de código abierto y fue incluido por primera vez en el kernel de Linux en agosto del 2008 [9].

Una de las características principales por la cual se decidió utilizar este driver en el transcurso del proyecto, es la dependencia que tiene con mac80211, que permite, entre otras cosas, realizar modificaciones a los módulos contenidos en el kernel de Linux, sin necesidad de modificar la lógica incluida en los controladores.

2.2.5. OpenWRT

El sistema operativo OpenWRT [10] para dispositivos integrados es un sistema operativo especialmente diseñado para enrutadores WLAN. Aunque OpenWRT se creó para enrutadores WLAN, también puede ejecutarse en varios tipos de dispositivos, por ejemplo, teléfonos inteligentes o computadoras portátiles. Mientras que la mayoría de los otros sistemas operativos de enrutadores WLAN consisten en un único firmware estático, OpenWRT proporciona un sistema de archivos totalmente grabable con su propio sistema de administración de paquetes. Uno de los principales objetivos de OpenWRT era producir un sistema que se ajustara a la memoria flash incorporada de aproximadamente cuatro a ocho megabytes que los enrutadores WLAN suelen proporcionar.

El proyecto OpenWRT comenzó en enero de 2004 y se basó primero en las fuentes de Linksys GPL20 para WRT54G. En 2005, se publicó una nueva versión que usaba fuentes oficiales del kernel GNU/Linux y solo agregaba parches para el sistema en el chip y los controladores de la interfaz de red. Como OpenWRT se basa en el kernel de Linux, también se basa en los controladores de la comunidad inalámbrica de Linux. En el pasado, los controladores inalámbricos usaban las extensiones inalámbricas (WEXT) para comunicación de espacio de usuario/kernel/hardware. Pero hoy WEXT está deprecado y fue reemplazado por cfg80211 junto con mac80211. Además del kernel de Linux y en conexión con cfg80211, el demonio de espacio de usuario hostapd se está ejecutando para manejar todas las funciones de administración del AP IEEE 802.11 [11].

2.2.6. MAC80211

En el pasado, el hardware de la interfaz de red era responsable de manejar la capa MAC y otros protocolos de capa inferior. Algunos fabricantes también utilizaron un modelo de dos partes de hardware y firmware de dispositivo de red. Este firmware era propietario y no era accesible para desarrolladores. Pero los fabricantes cambiaron su implementación a un modelo más flexible donde la funcionalidad MAC se implementa en software (este enfoque se llama SoftMAC), que luego se carga como un módulo del kernel de Linux.

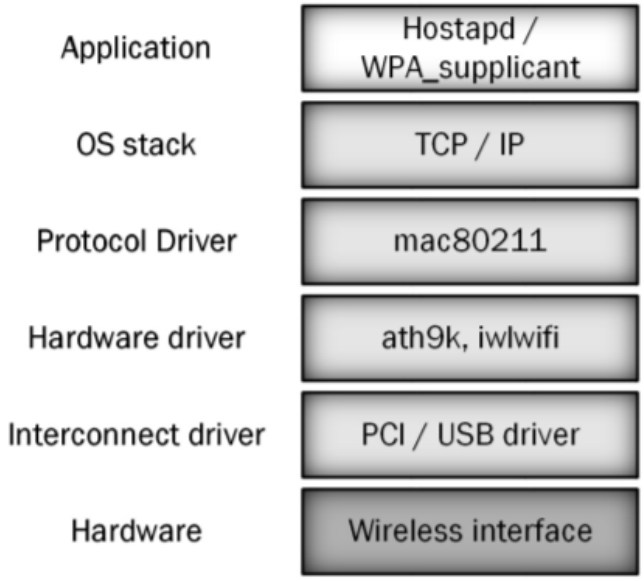


Figura 2.2: Diagrama del stack WLAN de Linux. Fuente [12]

La Figura 2.2 muestra el diagrama de capas del stack de red de Linux con el enfoque SoftMAC. El controlador de WLAN se divide en dos módulos, un módulo dependiente del hardware y un módulo de protocolo más genérico. Si bien el módulo dependiente del hardware es diferente para cada proveedor, el SoftMAC (módulo de protocolo) es utilizado por diferentes proveedores. En el caso de los controladores de WLAN , el módulo SoftMAC se llama mac80211. Para el controlador de hardware, el controlador ATH9K (un controlador para chipsets PCI 802.11n basados en Atheros) es uno de los más populares. Otros controladores de hardware son por ejemplo ath5k para chipsets AR5xxx basados en Atheros o iwlwifi para hardware basado en Intel.

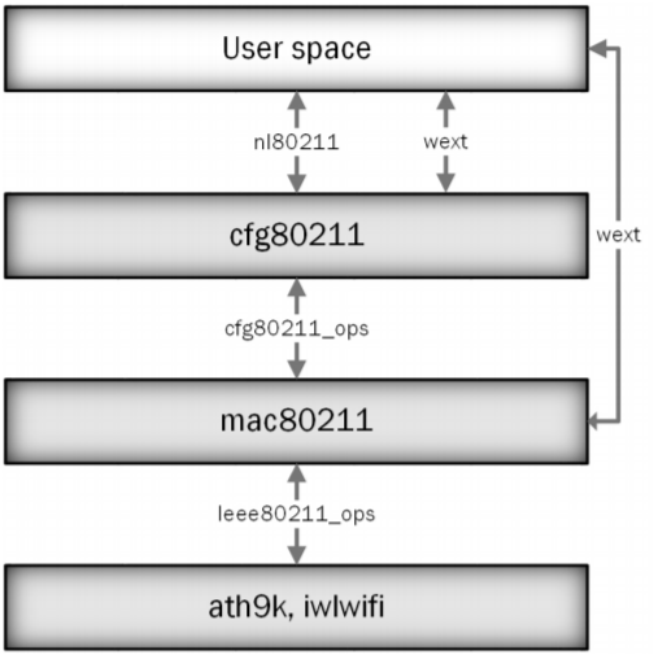


Figura 2.3: Estructura del módulo mac80211. Fuente [12]

La Figura 2.3 muestra los módulos particulares que se comunican con mac80211. Además, en la misma figura también se muestra la forma en que se realiza la comunicación entre los diferentes módulos. Mientras que cfg80211, mac80211 y el controlador ATH9K están dentro del espacio del kernel, las aplicaciones de control y administración residen en el espacio de usuario. Las aplicaciones de espacio de usuario populares son, por ejemplo, el hostapd ya mencionado para operaciones de AP, la herramienta iw para configurar el dispositivo inalámbrico, o wpa_supplicant que es la contraparte de hostapd y es utilizado por los dispositivos cliente para conectarse a un AP. Lo que todos ellos tienen en común es la forma en que se comunican con el módulo cfg80211. Como muestra la Figura 2.3, la comunicación se realiza mediante nl80211. Por supuesto, también es posible la antigua forma de comunicación utilizando las extensiones inalámbricas (wext), pero solo para proporcionar compatibilidad con versiones anteriores. Por lo tanto, solo se utiliza la comunicación que utiliza nl80211 [13].

2.2.7. NL80211

Nl80211 es una interfaz Netlink [14] IEEE 802.11 específica presentada en un archivo de encabezado C público, donde la comunicación se implementa utilizando sockets. Netlink es la interfaz preferida para la comunicación de espacio de usuario/espacio de kernel destinada a la configuración de red. Con los sockets Netlink es posible utilizar las API de socket estándar para abrir, cerrar, transmitir o recibir desde un socket, mientras que un socket se encuentra dentro del espacio del usuario y un socket se encuentra dentro del espacio del kernel [11].

2.2.8. CFG80211

El módulo del kernel cfg80211 recibe la comunicación Netlink desde el espacio del usuario, realiza las validaciones y la traducción del protocolo para comunicarse con el subyacente mac80211. Además, cfg80211 admite funciones para el registro de dispositivos, cumplimiento normativo, administración de estaciones, administración de claves, administración de malla, interfaz de gestión virtual y escaneo. La API del módulo mac80211 se realiza mediante devoluciones de llamada. Mac80211 luego convierte los datos en el formato de trama IEEE 802.11 correcto e inicializa todos los búferes y encabezados necesarios. Se elige una velocidad de transmisión adecuada mediante un algoritmo de control de velocidad y se realiza el cifrado y la fragmentación. Si el hardware admite un alto rendimiento (IEEE 802.11n), mac80211 también es responsable del manejo de ACK de bloques y la agregación de tramas. Por supuesto, en la dirección de recepción se ejecutan las mismas funcionalidades en orden inverso [15].

2.3. Antecedentes

El mecanismo a implementar en el presente proyecto, fue presentado en el artículo titulado "Slicing in WiFi Networks Through Airtime-based Resource Allocation" [1]. En este artículo se presenta el algoritmo ATERR y se realiza una prueba teórica que ratifica que efectivamente la cuota de transmisión asignada a cada slice se cumple, y que la asignación a cada cliente dentro un un mismo slice se hace de forma justa. A su vez se realiza un estudio analítico del funcionamiento del algoritmo, sumado a una prueba experimental en un ambiente simulado, donde los resultados muestran que el modelo analítico predice el comportamiento del algoritmo de forma correcta.

La implementación del algoritmo ATERR tiene como base el artículo recién mencionado. Para el desarrollo del proyecto y el cumplimiento de los objetivos planteados se hace uso principalmente de otros dos proyectos de código libre. Ambos trabajos presentan similitudes en algunos aspectos con el presente proyecto, lo que los hacen valiosos para contribuir al desarrollo del mismo. Para el primer objetivo se utiliza como base la implementación existente de un algoritmo, donde se asemeja al caso que se quiere implementar, pero el quantum asignado a cada cliente es fijo y no modificable. Este algoritmo es presentado en el artículo "Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi" [16] y se describe en el Capítulo 3. La implementación del segundo objetivo parte desde un trabajo realizado por Sven Zehl en la universidad de Berlín, bajo el título "hMAC: Enabling Hybrid TDMA/CSMA on IEEE 802.11 Hardware" [12], donde se trabaja en una interfaz de comunicación entre los modos usuario y kernel, para el pasaje de parámetros utilizables en la implementación del driver ATH9K [17]. El mismo autor junto con otros colaboradores, presentó también otro trabajo de interés para este proyecto, un artículo llamado "Hotspot Slicer: Slicing virtualized Home Wi-Fi Networks for Air-Time Guarantee and Traffic Isolation" donde habla sobre una posible implementación de slices y pruebas a ejecutar [18].

Para llevar a cabo la instalación física del código implementado, se evaluaron principalmente dos opciones. La idea original fue utilizar OpenWRT, ya que es el sistema operativo que se encuentra embebido en los APs que utilizan el driver ATH9K. Luego de estudiar otras posibilidades, se llegó a la alternativa de utilizar una computadora con sistema Linux, que cuente con la tarjeta de red apropiada, y configurando la misma en modo AP. De esta forma se obtiene la misma funcionalidad provista por un router o AP natural, pero se tienen facilidades en el proceso de trabajo, como son la compilación y depuración del código, y sobre todo en la ejecución de las pruebas. De todas maneras, a los efectos del código fuente del driver modificado, en ambos casos es exactamente el mismo. Sí cambia la forma en que se llega a ejecutar la nueva versión en ambos equipos físicos. Como en este

caso se trabajó con una versión de Linux de escritorio, esta es la versión que se detalla en el proyecto para ejecutar el código modificado.

Capítulo 3

Algoritmos

Este trabajo se basó en dos algoritmos de planificación de paquetes de red, que ejecutan en el AP encargado de controlar y suministrar la conexión a sus clientes. En primer lugar se debe presentar el algoritmo expuesto en el trabajo "Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi", definido bajo el título de "Airtime fairness scheduling" [16]. Este algoritmo "Fairness" ya se encuentra previamente implementado en el kernel de Linux y recibe este nombre porque justamente intenta ser equitativo con todos sus clientes, con respecto al ancho de banda que promete a cada uno de ellos. En el presente trabajo se referirá a este algoritmo como *algoritmo base*, ya que es a partir de donde nace la implementación de ATERR, que es el segundo algoritmo en cuestión, y sobre el que está centrado este proyecto. Ambos algoritmos son presentados en las siguientes secciones, haciendo énfasis en las estructuras y cálculos de la implementación del algoritmo base que son útiles para obtener la solución de ATERR, pero también marcando las diferencias entre ambos algoritmos, y la forma en que se espera llegar a implementar ATERR a partir del algoritmo base.

3.1. Algoritmo base

El algoritmo base o "Fairness" expuesto en [16], es un algoritmo de Round Robin , donde se introduce el concepto de Traffic Identifier (TID) y donde cada paquete será mapeado a un TID de acuerdo a su tipo de servicio. Este concepto será muy útil para representar las slices, como se verá más adelante. Si se toma en cuenta un único TID, por ejemplo el 0 (que es el que se utiliza en la mayoría de los casos para el tráfico regularmente), cada nodo (cliente) conectado al AP, cuenta con una cola individual en la que se procesan los paquetes

a transmitir. Este uso de un único TID, es la forma en que se interpreta el algoritmo fairness, para enfocarlo como punto de partida en la implementación de ATERR. Cada una de las colas cuenta con un quantum que va a ir disminuyendo a medida que transmite paquetes, y el algoritmo de Round Robin implementado por el planificador, rota entre las colas de forma justa de acuerdo al quantum disponible y el orden en que han ido transmitiendo las mismas. En el caso de esta implementación, todas las colas comienzan con el mismo quantum asignado, de 300 microsegundos. El código correspondiente a esta solución, se encuentra disponible de forma libre y pública en el kernel de Linux, a partir de la versión 4.15. El commit donde se introducen estos cambios es el punto de partida que se utilizó para la implementación del algoritmo ATERR [19].

3.2. ATERR

ATERR también es un algoritmo de planificación de tipo Round Robin, que agrega la complejidad de las slices para distintos tipos de tráfico, implementadas haciendo uso del airtime. En este caso el planificador mantiene una cola diferente por cada par de usuario y slice. Esto significa que, si un usuario participa en 3 slices distintas, se crearán 3 colas para ese usuario, una por cada slice.

Una gran ventaja de partir del algoritmo base, es aprovechar la estructura ya implementada para las distintos TIDs, para así utilizar un mapeo directo de TIDs con slices. Para esto es necesario quitar el uso de TIDs con el propósito original de diferenciar tipos de servicio. En la implementación de ATERR que se verá más adelante, los TIDS se utilizan para indicar la slice que desea utilizarse.

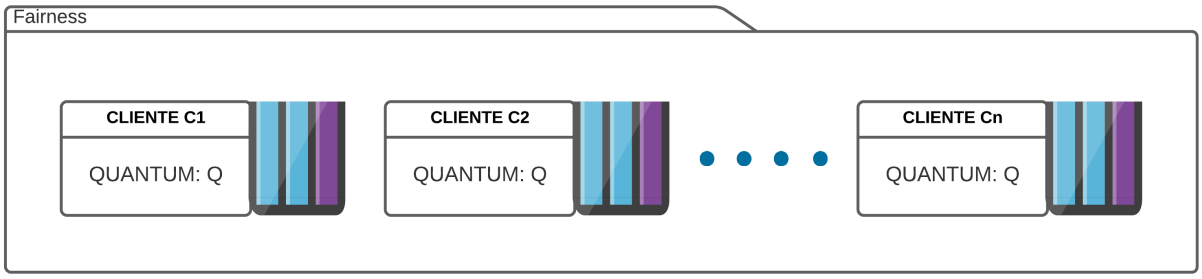
El diseño de ATERR tiene dos objetivos principales, asignar el airtime correspondiente a cada slice y asignar el airtime correspondiente a cada usuario dentro de una slice, de manera justa. El algoritmo entra en acción cada vez que se desea realizar una nueva transmisión de paquetes, y promete asignar los recursos de forma justa a las slices, independientemente de las características del canal que utilice cada usuario. Mediante un sistema de Round Robin, se irán intercalando las colas que deben transmitir de acuerdo al quantum que tengan asignado. Dicho quantum se corresponderá con el ratio asignado a la slice y con la cantidad de usuarios que estén utilizando la misma.

Las diferencias principales entre el algoritmo base y ATERR, con respecto a sus implementaciones, radican en que en ATERR, el planificador ya no deberá basar su decisión en el quantum restante para cada nodo, sino que debe haber una estructura diferente que permite controlar el quantum de acuerdo al nodo y el TID (slice) utilizado. En la Figura

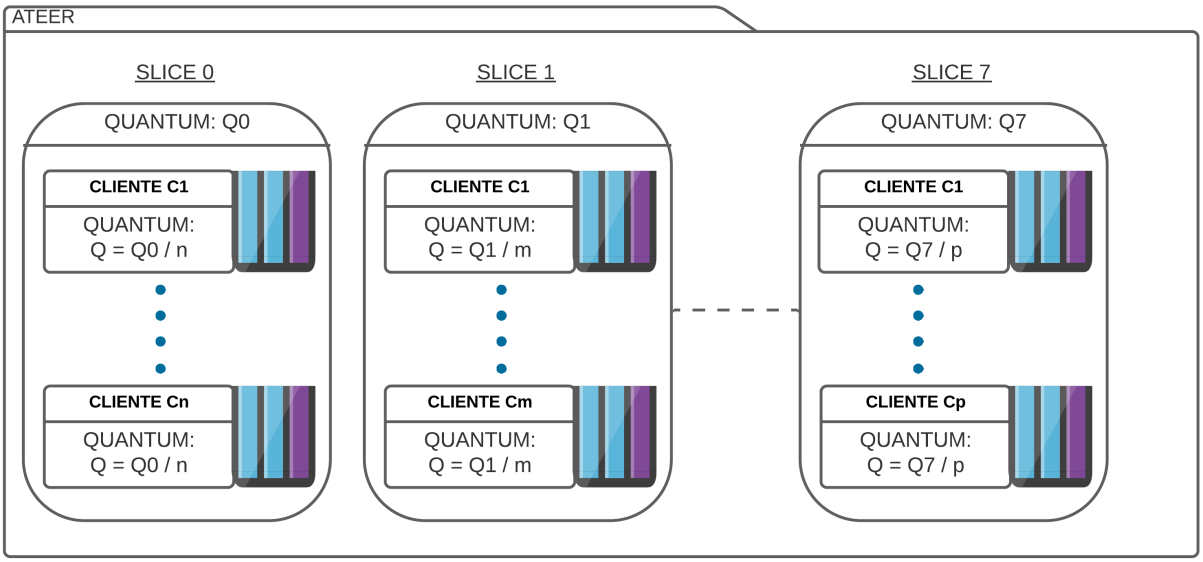
3.1 se muestra una comparación de la estructura de colas que utilizan ambos algoritmos, y como la adición de slices influye en el quantum asignado a los clientes.

A continuación se enumeran los principales cambios que deben aplicarse para obtener la solución de ATERR:

- El trabajo previo cuenta con una cola individual por cliente, mientras que ATERR mantiene una cola por cada par cliente/slice.
- El trabajo previo asigna el mismo quantum a todos sus clientes, previamente definido. Por otro lado, ATERR cuenta con quantums diferentes y calculados dinámicamente para cada una de sus colas. Donde lo que sí debe respetarse es que el quantum sea el mismo para cada una de las colas correspondientes a una slice (es decir, las colas de distintos clientes pero para la misma slice tendrán el mismo quantum). ATERR implementa dentro del contexto de una slice, un algoritmo fairness para los clientes pertenecientes a esa slice.
- Por lo tanto, ATERR debe especificar las slices y redefinir la estructura de colas para considerar las mismas, además de recalcular los quantums siempre que haya una variación en el conjunto de clientes o sus slices asignadas.



(a) Representación de la estructura de colas manejadas por el algoritmo base para sus clientes



(b) Representación de la estructura de slices y colas manejadas por ATERR para sus clientes

Figura 3.1: Diagramas comparativos de los algoritmos base y ATERR y la diferente forma en que emplean los quantums para sus clientes

3.2.1. Cálculo del quantum

El algoritmo redefine el quantum asignado a cada cola, cada vez que hay un cambio en el ratio de una slice, o se agrega/quita un nodo a una slice. Esto deriva en cambiar el quantum de las colas que representan los nodos de esa slice (ya que cambia la cantidad de nodos pertenecientes a la misma) y esto a su vez puede derivar en modificar el quantum del resto de las colas. Todo el cálculo de quantums se hace a partir de un quantum mínimo de-

finido MIN_AIRTIME_QUANTUM (100 microsegundos en la implementación de ATERR utilizada) que es utilizado para asignar a las colas de la slice de menor ratio, por lo que todos los nodos en esa slice tendrán el mismo quantum mínimo. A partir de este valor se calcula el resto de los quantums, siempre teniendo en cuenta el ratio asignado a cada slice y la cantidad de nodos pertenecientes a las mismas, manteniendo la relación correcta con respecto a los ratios.

El siguiente es un pseudocódigo del cálculo de los quantums por parte del algoritmo:

1. Encontrar la slice de menor quantum (minSlice):
   - Para cada slice i calcular: x=ratio(i)/cantidadClientes(i)
   - Definir minSlice como la slice con el mínimo valor de x
2. Asignar a minSlice el quantum mínimo: quantum(minSlice) = MIN_AIRTIME_QUANTUM
3. Definir minQ = MIN_AIRTIME_QUANTUM * cantidadClientes(minSlice)
4. Para cada slice i (distinta a minSlice):
   - Calcular el quantum de cada slice con la siguiente operación:
     quantum(i) = ((ratio(i)/ratio(minSlice))*minQ)/ cantidadClientes (i)

Capítulo 4

Solución propuesta

Una vez comprendido el algoritmo que se busca implementar, se procedió a investigar los módulos del kernel de Linux encargados del driver ATH9K, con el objetivo de diseñar la solución [20-23]. Durante las primeras etapas de la investigación se determinaron los procedimientos y funciones involucradas en la asignación del quantum por parte del sistema, y se comenzó con el diseño de las modificaciones necesarias para implementar la separación del tráfico en slices. De forma paralela comenzó el estudio y se realizaron las primeras pruebas sobre la compilación personalizada del kernel completo y de módulos individuales [24]. Esto dio como resultado que era necesario compilar una sola vez el kernel completo y luego se podía compilar por separado el módulo encargado del driver ATH9K para cada cambio que se introduzca. De esta forma se agiliza el proceso de implementación debido a la larga duración que puede producirse al compilar el kernel de forma completa. Con respecto a la forma de permitir que el usuario administrador impacte el driver implementado directamente de forma dinámica, se estudiaron algunas variantes al comienzo del proyecto, pero al ser un componente independiente a la implementación del algoritmo propiamente dicho, se retrasó un poco el estudio concreto de la solución a utilizar en las etapas tempranas de análisis y diseño.

4.1. Arquitectura

La arquitectura general de la solución propuesta se presenta en el diagrama de la Figura 4.1. Los componentes resaltados con color verde son aquellos que fueron creados o modificados para implementar esta solución de slicing. Se puede ver como interaccionan dentro del AP los distintos componentes que son partícipes de la solución, diferenciando los

espacios de usuario y kernel. Además se incluyen los actores externos al AP que interactúan con este, estos son los clientes y el administrador de la red, este último es el encargado de gestionar y configurar las slices y los clientes pertenecientes a las mismas.

Dentro del módulo de transmisión xmit se crearon estructuras auxiliares para dar soporte al uso de slices. Se agregó un arreglo para guardar la información de cada una de las slices, que indica si una slice esta activa, el ratio que tiene asignado y la cantidad de usuarios utilizando la misma. Además se añadió una lista donde se guardan los nodos conectados, indicando a qué slices pertenecen. Esta lista es diferente a la que maneja ATH9K para gestionar los nodos conectados al AP, con el objetivo práctico de mantener cierta independencia en la implementación, que evite introducir posibles problemas en la estructura ya existente. De todos modos, en ambos casos los nodos representados son los mismos.

Para la comunicación se debió agregar un nuevo módulo al kernel, llamado NetLink-Kernel, que interactúa con el módulo modificado xmit. También para la comunicación es utilizada una aplicación de usuario, esto se explica con más detalle en la Sección 4.3.2.

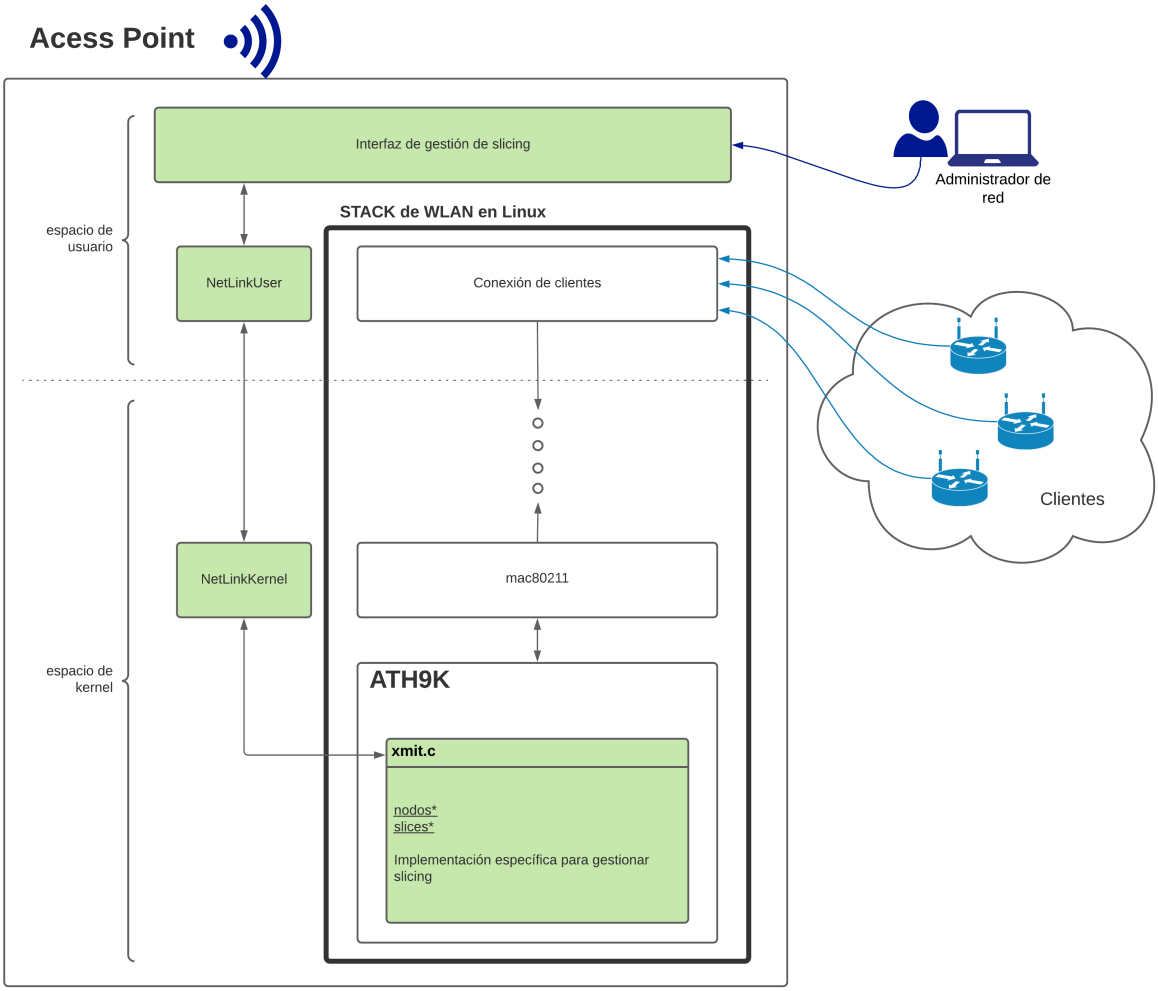


Figura 4.1: Diagrama de arquitectura para la solución propuesta

4.2. Análisis y diseño

Una de las decisiones más críticas para desarrollo de todo el proyecto fue la forma de representar el concepto de slice. La idea que se impuso fue utilizar la estructura TID ya implementada en la versión del driver desde la que parte el presente trabajo.

Por cada nodo existen 16 TIDs distintos para manejar diferentes tipos de tráfico, según

el nivel de servicio, que se puede diferenciar en el campo TOS (Type of Service) del protocolo IPv4. El objetivo es utilizar estos TIDs para que cada uno maneje el tráfico correspondiente a una slice dentro de un nodo, aprovechando que ya existe la implementación de una estructura de una cola por TID para cada nodo.

A pesar de que se pueden representar 16 TIDs con la estructura elegida, con el campo TOS se permite diferenciar solamente 8. De todas formas, se decidió utilizar este campo principalmente porque se encontró especialmente útil que con la herramienta iPerf3 [25] es posible indicar dicho campo para el tráfico a generar, y de esta forma se puede utilizar este campo al momento de procesar los datagramas para identificar qué slice se encuentra marcada. Además, parece una cantidad más que suficiente para probar el funcionamiento de las slices en este prototipo, siempre teniendo en cuenta que el objetivo del presente trabajo es conseguir un prototipo funcional del algoritmo para sustentar que su utilización es realmente posible en un ambiente práctico. Para indicar el TOS en las instrucciones iPerf3, se utiliza la bandera –S con el valor de TOS en notación hexadecimal con prefijo 0x, por ejemplo, para indicar que el tráfico generado se envía por la slice 0 se agrega el parámetro –S 0x10 [26].

El cambio principal que esto genera, es que ahora el scheduler decidirá cómo mover a los TIDs de las colas de transmisión, de acuerdo al déficit de airtime que tenga cada slice particular, en vez del nodo en general.

4.2.1. Enfoque del problema

El objetivo principal del proyecto involucra en primer lugar obtener una implementación del algoritmo ATERR para el driver ATH9K, y en segundo lugar implementar una interfaz de comunicación entre el espacio de usuario y el kernel, para poder definir distintas slices y modificar sus quantums.

Para encarar el problema se diseñó una solución incremental que se puede dividir en 4 fases principales:

- **Fase 0:** Compilar e instalar una versión modificada del kernel. Ver anexo A.2.
  - 0.1: Ejecutar una versión modificada del kernel compilando el mismo íntegramente
  - 0.2: Ejecutar una versión modificada del kernel compilando e instalando únicamente un módulo, en este caso el módulo ATH9K

- **Fase 1:** Versión funcional del algoritmo con parámetros definidos en el módulo de forma estática.
  - 1.1: En principio se asume que cada cliente pertenece a una única slice de quantum fijo, siendo el caso base del algoritmo
  - 1.2: Luego se agrega la posibilidad de tener más de una slice por cliente, según tipo de tráfico por ejemplo
- **Fase 2:** Versión funcional del algoritmo con parámetros dinámicos

  El objetivo de esta fase es que el módulo sea capaz de recalcular los quantums de todas las slices, de forma acorde al algoritmo, cuando hay cambios de escenario. Estos cambios se pueden dar en dos casos:
  - cuando se conecta o desconecta un cliente al AP y provoca modificaciones en la cantidad de clientes activos sobre una o más slices
  - cuando el quantum de una de las slices es modificado en tiempo de ejecución
- **Fase 3:** Generar una interfaz de usuario para administrar los recursos asignados a las slices.

  Se implementará una interfaz de comunicación que permita enviar mensajes entre el espacio de usuario y el kernel. De esta manera el usuario administrador de la red tiene la posibilidad de impactar directamente en el kernel para crear, modificar y eliminar slices del AP. Esto permite al administrador decidir qué clientes se asigna a cada una de las slices y qué ratio tendrá cada una de ellas.

4.3. Implementación

4.3.1. Algoritmo ATERR

Las principales modificaciones de este prototipo, con respecto a la implementación del algoritmo, se encuentran realizadas sobre el archivo xmit.c (ath/ath9k/xmit.c) [27]. En este módulo es donde se maneja la conexión de los nodos a nivel de TIDs y donde el scheduler decide quién debe ejecutar de acuerdo a los quantums disponibles. La estructura TID se encuentra definida en el archivo ath9k.h mediante el identificador ath_atx_tid. Para lograr las modificaciones propuestas en la sección anterior, se debieron realizar los siguientes cambios en el driver a nivel de código:

- Agregar una variable a la estructura ath_atx_tid para representar el tiempo de transmisión restante y disponible por la slice/nodo que representa, antes de ser enviado al final de la cola por el scheduler. Esta variable, que anteriormente se encontraba definida en la estructura que representa el nodo, se declara de la siguiente manera: s64 airtime_deficit[IEEE80211_NUM_ACS];

- Un detalle sobre la utilización de los TIDs, es que se debe utilizar siempre el mismo número AC (IEEE80211_NUM_ACS). Este valor originalmente es calculado en base al TID utilizado (los paquetes llevan una marca para indicar el TOS que se asocia con un TID) para darle distinta prioridad a los paquetes asignando distintos AC. Esta funcionalidad previa no es necesaria por lo que se quita y se aprovecha la estructura ya definida para el manejo de slices mediante los quantums asignados. Por lo tanto, en la inicialización de cada TID de un nodo (en el archivo xmit.c) se cambia la función TID_TO_WME_AC(tid->tidno) por una simple constante única.

- Definir una estructura para representar todas las slices, donde cada una es identificada por un entero tidno que se correponde con el valor tidno de la estructura ath_atx_tid de un nodo, que representa dicha slice. Los valores de tidno van de 0 a 15, cantidad de colas ath_atx_tid máximas por nodo, por lo que existe un número máximo de 16 slices representables en el sistema. Como ya se dijo, para las pruebas actuales se utilizan 8 slices de 0 a 7, con las cuáles se puede generar tráfico específico para uno de estos TIDs mediante la herramienta iPerf3. Dentro de la slice se define un valor de quantum global para la misma, y un valor de quantum por nodo perteneciente a la slice, valor que es la división del quantum global por la cantidad de nodos pertenecientes a la slice. Las slices estarán comprendidas en un arreglo de tamaño 8, que se inicializa al comenzar a funcionar el módulo.

- Definir otra estructura que represente cada uno de los nodos que se conectan al AP. En este caso la cantidad de nodos se espera que sea dinámica, por lo que se utiliza una lista de la forma list_head para representar todos los nodos conectados. Para la identificación de cada nodo se decidió utilizar la dirección MAC de cada dispositivo. La estructura del nodo contiene una variable en forma de arreglo de booleanos, que representa las slices a las que pertenece el nodo en cuestión, para saber cuáles TIDs de dicho nodo serán utilizados.

- Para modelar el ratio asignado a cada slice, se define una variable global, que es un arreglo de 8 enteros positivos, donde cada entrada representa el porcentaje de ancho de banda (ratio) asignado a cada slice, y la suma de todos sus elementos siempre debe ser 100.

- Por último, se implementaron funciones correspondientes al cálculo del algoritmo ATERR propiamente dicho, las cuales, siempre que hay un cambio de escenario, se ejecutan, recalculando y asignando el quantum a cada una de las 8 slices, en base al ratio asignado a cada slice y a la cantidad de nodos pertenecientes a cada una de ellas.

En las figuras 4.2 y 4.3 se muestra una representación de las estructuras de datos creadas para representar las slices y los nodos, respectivamente.

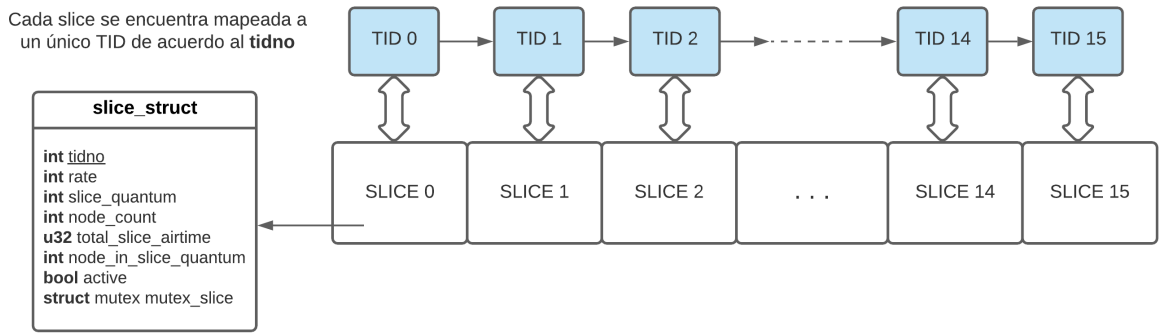


Figura 4.2: Representación de la estructura de slices utilizada en el módulo xmit

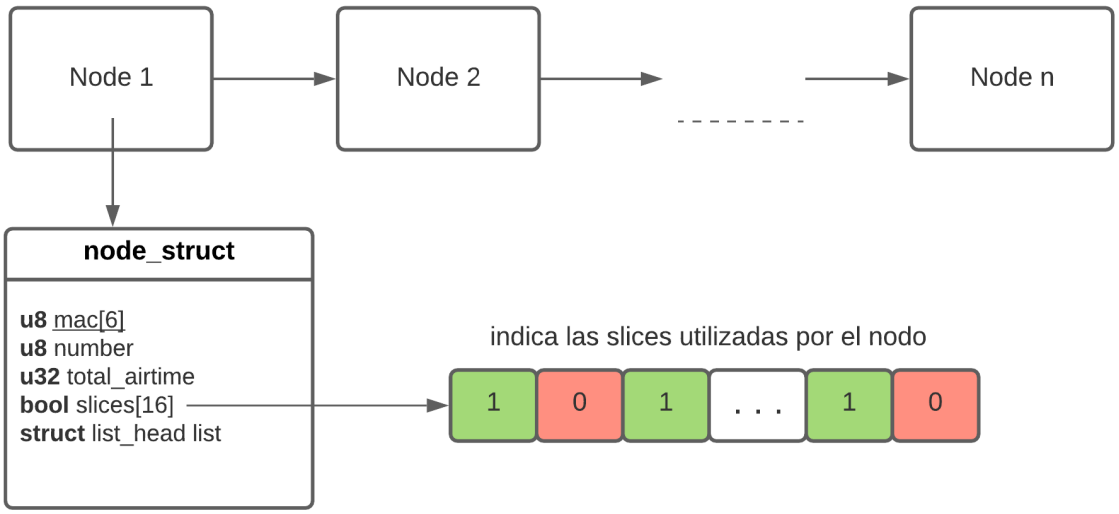


Figura 4.3: Representación de la estructura de nodos utilizada en el módulo xmit

4.3.2. Comunicación

El diseño de la comunicación se basa en utilizar NetLink [14] una interfaz de Linux basada en sockets que cumple el propósito de comunicar el espacio de usuario con un módulo dentro del kernel.

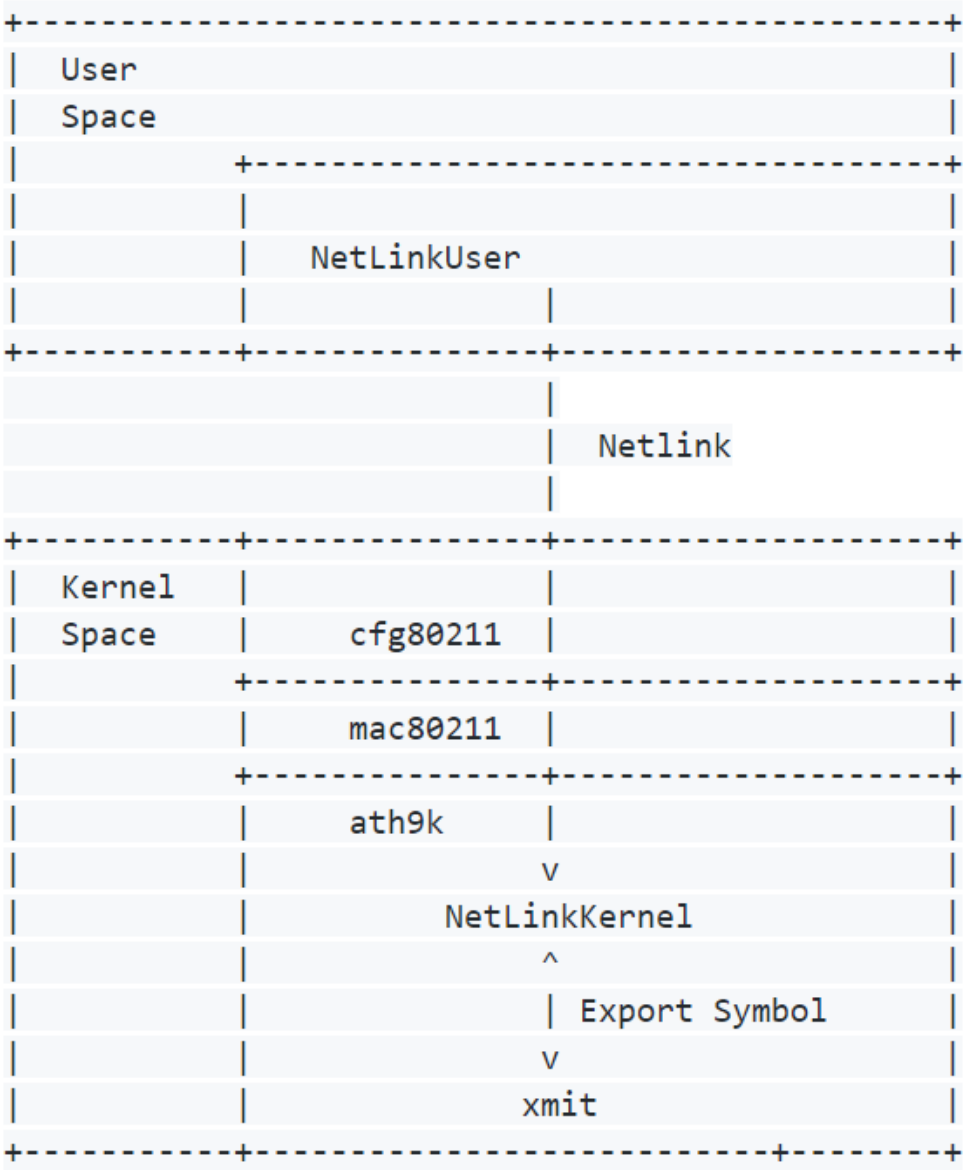


Figura 4.4: Esquema de comunicación utilizado entre los espacios de kernel y usuario.

Para lograr la comunicación buscada, se implementaron los módulos NetLinkUser y NetLinkKernel, ubicados en los espacios de usuario y de kernel, respectivamente, ambos basados en el uso de NetLink.

NetLinkUser es una aplicación de consola que, mediante el ingreso de comandos defi-

nidos, envía mensajes al módulo NetLinkKernel, que se encuentra en el otro extremo de la comunicación.

NetLinkKernel, mediante Export Symbol realiza llamadas a funciones del driver ATH9K dentro del módulo xmit, de acuerdo a los mensajes recibidos por parte del módulo de usuario. Por ultimo NetLinkKernel se encarga de recibir la respuesta del driver consultado, la procesa y envía a NetLinkUser para que pueda informarle al usuario el resultado de las operaciones realizadas.

Las instrucciones que puede ejecutar el usuario administrador a través de la aplicación NetLinkUser son las siguientes:

- Agregar un nodo a una slice. Esto puede provocar la creación de una nueva slice desde el punto de vista del usuario, si es que esta no admitía ningún nodo hasta el momento.
- Quitar un nodo de una slice.
- Modificar el ratio asignado a cada una de las slices (se verificará que la suma total sea 100, ya que cada ratio representa un porcentaje del total)

Cualquiera de estas instrucciones provoca cambios que derivan en que se deba ejecutar el algoritmo ATERR para recalcular todos los quantums asignados.

Capítulo 5

Pruebas realizadas

5.1. Objetivos

Se plantea como objetivo principal de las pruebas la evaluación del correcto funcionamiento del algoritmo en un ambiente práctico. Se busca comprobar que los resultados se asemejan a los valores teóricos especificados, así como buscar posibles causas y mejoras en los casos en que haya diferencias. Para esto, se plantean distintos escenarios en los cuales el AP que implementa el algoritmo, ofrece la conexión a uno o más clientes.

5.2. Configuración de los escenarios

Para la ejecución de las pruebas fue utilizada una computadora de escritorio para oficiar como el AP al que se conectan los nodos. Esta PC cuenta con una versión de Ubuntu 16.04 LTS y kernel 4.15, el cual fue modificada para soportar el algoritmo ATERR, implementado sobre el driver ATH9K. Para poder utilizar dicho driver modificado, se conectó a la computadora una tarjeta de red inalámbrica AR9227 de Qualcom Atheros, la cual utiliza este driver para las funciones de WiFi. En cuanto al equipamiento empleado para los clientes (nodos), se utilizaron routers TP-Link N750 en modo cliente. Para la definición de los distintos escenarios, se plantean dos distancias diferentes a las que se pueden encontrar los clientes con respecto al AP. La primera distancia, a la que llamaremos distancia 1, es de 4 metros y es la menor entre un cliente y el AP. A esta distancia los clientes alcanzan un 85 % de señal y valores de potencia entre los -20 y -26 dBm. La segunda distancia (distancia 2) es 2 metros mayor entre el cliente y el AP, con un total de

6 metros entre ambos. Para esta distancia los clientes tienen un 65 % de señal con valores de potencia entre los -16 y -22 dBm. Cabe destacar que la diferencia entre las distancias 1 y 2 no es la ideal para la visualización de los resultados, pero no fue posible definir una distancia superior debido al entorno físico en que se realizaron las pruebas. De todas maneras, como veremos más adelante, se pueden observar cambios ligeros en las velocidades de transmisión, que se comportan de acuerdo con lo esperado (a mayor distancia menor velocidad).

Para acceder a los routers vía SSH se conectó a los mismos a través de una laptop utilizando una LAN con ayuda de un switch. De esta manera los equipos A, B y C se conectan a la red wifi del AP y ejecutan las instrucciones iperf, para quedar escuchando a la espera de las transmisiones de las distintas pruebas.

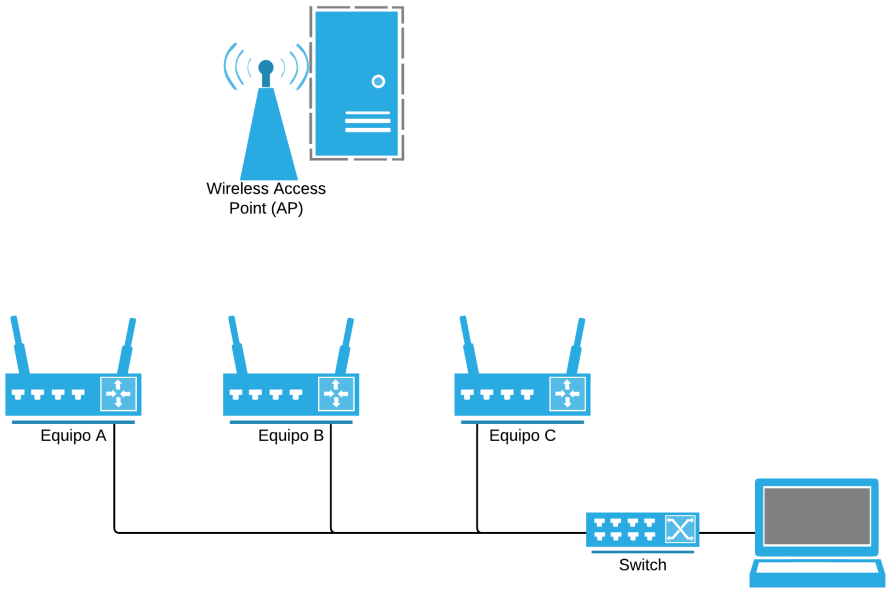


Figura 5.1: Esquema del ambiente de pruebas utilizado.

5.3. Herramientas utilizadas

Para todas las pruebas se utiliza la herramienta iPerf3 [25], la cual permite generar tráfico desde el AP hacia un nodo a través de la slice deseada (slices 1 a 7). Tanto en

el AP como en los clientes se instaló la versión de iPerf3 3.1.3. El AP es quien actúa como cliente iperf, enviando paquetes de datos a los nodos que actúan como servidores, escuchando a la espera de paquetes en un puerto distinto por cada slice que utilicen en la prueba. De esta forma se estudia la velocidad de descarga de los nodos. Cada instrucción iperf ejecutada envía paquetes UDP sin límites de velocidad definidos, durante un lapso determinado que permita que la transmisión sea estable. Las instrucciones se ejecutan con distintos parámetros, que buscan principalmente asegurar la saturación del canal, para poder corroborar que el algoritmo ATERR está realmente en funcionamiento. Para verificar que el canal se encuentra saturado se puede acceder al porcentaje de pérdida de paquetes, indicado en la salida de los instrucciones iperf.

5.4. Escenarios analizados

La ejecución de las pruebas se dividió en tres etapas, las cuales plantean distintos escenarios con el objetivo de cubrir los diferentes casos en que el comportamiento del algoritmo se pone en manifiesto.

Cada etapa tiene un objetivo definido y cuenta con distintas pruebas que a su vez presentan más de un escenario. Cada escenario se define con una configuración particular de nodos (a cierta distancia) asignados a las slices y de los ratios definidos para cada slice. Durante cada uno de los escenarios se ejecutan de forma simultánea instrucciones iperf hacia distintos nodos y slices con una configuración definida, guardando los valores registrados de velocidad de transmisión y tiempo de transmisión consumido (airtime) por cada nodo y slice.

Con el objetivo de automatizar las pruebas se desarrollaron distintos scripts (escritos en bash y python) para cada escenario de las pruebas. Dichos scripts son los encargados de la configuración, ejecución y posterior obtención y procesamiento de los datos de airtime y velocidad para todos los nodos y slices involucradas. Para identificar cada envío de datos dentro del escenario particular de una prueba, los nodos utilizados se nombran A, B y C, mientras que las slices se las identifica por los números del 0 al 7. Para identificar el envío de datos en un escenario particular y mostrar los mismo en los gráficos presentados, se utiliza la nomenclatura XY, donde X e Y se corresponden al nodo y a la slice involucrados en el envío, respectivamente. Como ejemplo, la ejecución correspondiente al nodo A en la slice 2 recibe el identificador A2.

Para las pruebas elegidas, se prefirió evitar el uso de las slices 6 y 7, ya que en la implementación de este trabajo, estas slices hacen uso de los valores de TOS 6 y 7, los cuales

se utilizan para control de red. Aunque es un tráfico que puede considerarse despreciable, ya que es bajo y se da en su mayoría al momento del handshake, se tomó esta decisión para obtener valores de airtime que sean lo más acorde posible al uso de recursos proveniente de las ejecuciones iperf3. Todo esto teniendo en cuenta que el resto de las slices disponibles (0 a 5) alcanzan igualmente para ejecutar los escenarios planteados.

5.4.1. Anexo de pruebas

Algunas de las pruebas realizadas, no serán descriptas en la presente sección, debido a que se consideró que no aportan real valor a los resultados que se quieren presentar. De todas formas, dichas pruebas se encuentran presentadas en la Sección "Anexo de pruebas" del repositorio de GitLab [2], en la cual se describen todos los escenarios ejecutados, y las salidas completas arrojadas. Allí se puede consultar también la Guía de pruebas, donde se describen los pasos a seguir para replicar las prueba realizadas, haciendo uso de los scripts disponibles también en el repositorio.

5.5. Etapas

Las primeras dos etapas buscan probar el correcto funcionamiento del sistema ATERR en su totalidad, mostrando en detalle el comportamiento dinámico de las slices en variedad de situaciones. Esto se comprueba al agregar o quitar nodos a las distintas slices y al modificar el ratio asignado a cualquiera de ellas, realizando estos cambios en tiempo de ejecución y desde el espacio de usuario. Las pruebas realizadas en las dos primeras etapas tendrán los mismos valores de configuración, con la diferencia que la etapa II agrega la complejidad de mantener distintas distancias para los clientes, mientras que la etapa I es más sencilla y utiliza siempre la misma distancia desde los clientes al AP (distancia 1). La tercer etapa busca monitorear en tiempo real el comportamiento de los equipos cuando hay cambios de escenarios. Por este motivo la etapa III plantea escenarios básicos donde el objetivo principal es visualizar las variaciones segundo a segundo de forma clara, mientras que la prueba más intensiva del algoritmo se realiza en las primera dos etapas con una gran variedad de escenarios.

5.5.1. Etapa I

Para simplificar los resultados y tener una representación inicial más clara, en esta primera etapa todos los nodos se encuentran siempre a la misma distancia del AP (distancia

1). Como ya se comentó, las variaciones de distancia se prueban luego en la etapa II.

5.5.1.1. Resultados esperados

En esta primer etapa se espera que los resultados de las pruebas se correspondan en proporción de airtime y velocidad según los ratios asignados a cada slice, de acuerdo a los lineamientos del algoritmo. También se espera que los resultados sean independientes del nodo seleccionado. Los valores en cada escenario deben ser similares para los distintos nodos cuando ejecutan en las mismas condiciones, ya sea obteniendo los mismos resultados cuando se ocupa todo el canal, con valores análogos cuando se invierten los ratios asignados a los distintos nodos, o con los mismos valores por nodo en una slice, cuando la misma es compartida por más de un nodo.

5.5.1.2. Prueba 0 - Transmisiones individuales para cada nodo.

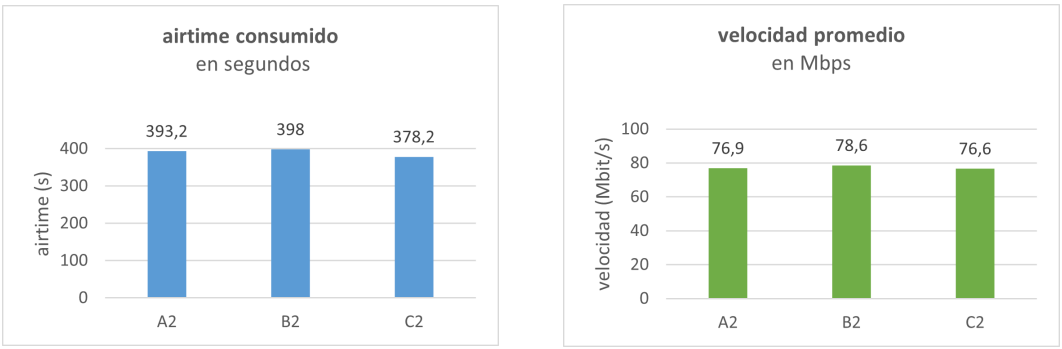
Es una prueba inicial para determinar los valores de airtime y velocidad de un equipo utilizando todo el ancho de banda disponible, sin competencia mediante. En cada escenario trabaja uno de los tres equipos utilizando todo el ancho de banda a través de una única slice, en este caso se eligió la slice 2, con un ratio equivalente al 100 %. De esta manera se obtienen números base para comparar con las siguientes pruebas y a su vez sirve para verificar que los equipos A, B y C funcionan de manera similar.

escenario 1		
slice	**ratio**	**equipos**
2	100	A

escenario 2		
slice	**ratio**	**equipos**
2	100	B

escenario 3		
slice	**ratio**	**equipos**
2	100	C

Tabla 5.1: Configuración de los escenarios en la prueba 0.



(a) Airtime total consumido

(b) Velocidad promedio

Figura 5.2: Gráficos que presentan el airtime total consumido y la velocidad promedio de cada equipo, en sus respectivas transmisiones individuales de la prueba 0, etapa I.

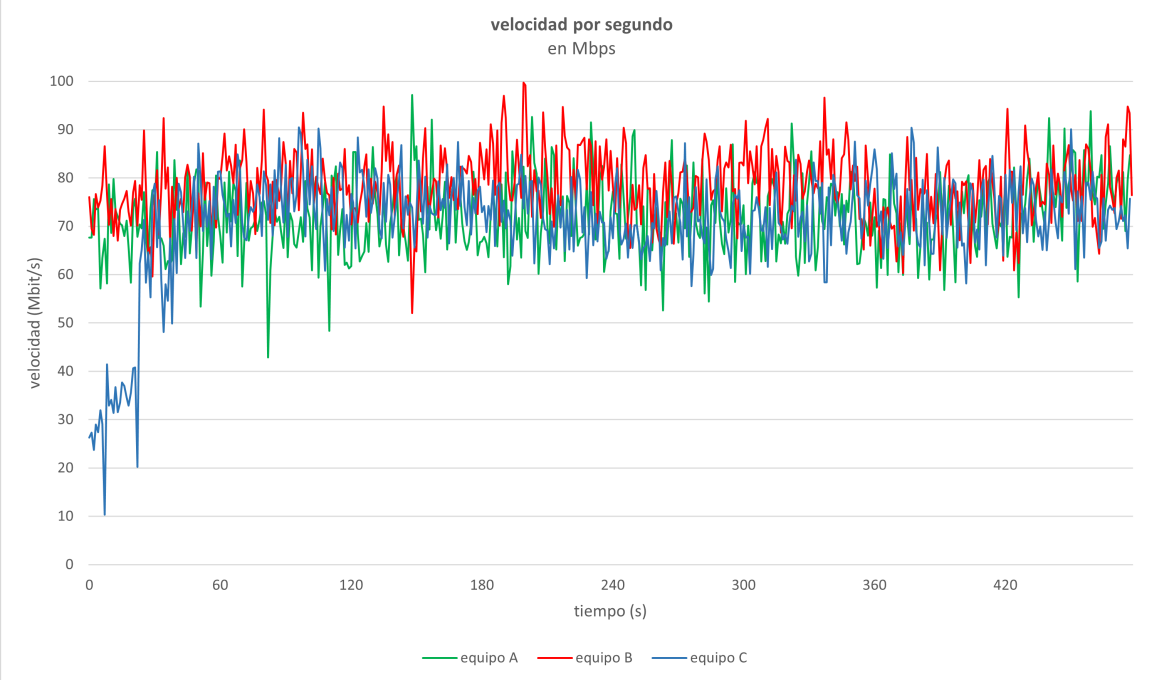


Figura 5.3: Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo, en sus respectivas transmisiones individuales de la prueba 0, etapa I.

Como se puede observar en los gráficos de la Figura 5.2 los tres clientes presentan valores similares de airtime y velocidad en sus respectivas ejecuciones para la distancia 1.

En el gráfico de la Figura 5.3 se muestra la velocidad en función del tiempo para cada nodo, donde se aprecia que todos los equipos alcanzan la velocidad promedio de una forma estable. Las variaciones de velocidad claramente están presentes pero se mantienen alrededor de los 75 Mbps. El nodo C tuvo problemas para alcanzar su velocidad máxima los primeros 30 segundos, pero luego pudo estabilizarse para transmitir en el mismo rango de velocidades que los nodos A y B. Los resultados obtenidos en esta prueba son los esperados y se tiene una correcta base para los siguientes escenarios que sean ejecutados, tal y como era el objetivo de esta prueba inicial.

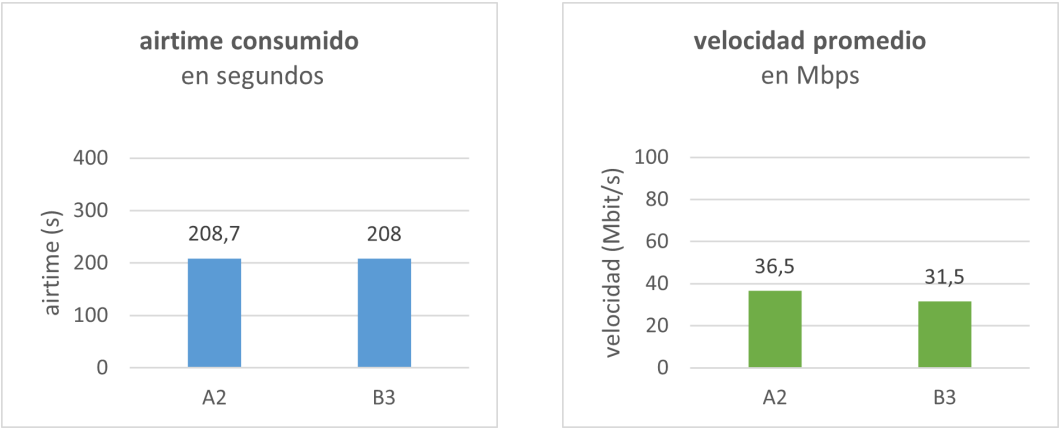
5.5.1.3. Prueba 1 - Dos nodos utilizando el canal, cada uno en una slice distinta.

En esta prueba los nodos A y B transmiten al mismo tiempo en slices distintas y se encuentran ubicados a la misma distancia. El nodo A transmite en la slice 2 mientras que el nodo B lo hace en la 3, y en cada escenario se utilizan distintos ratios para las slices de cada nodo. En el primer escenario ambas slices se reparten el ratio en partes iguales. En el segundo escenario es uno de los nodos el que ocupa la mayor parte del ratio, obteniendo tres cuartos del total. Con esta prueba se busca ver que el airtime consumido por cada slice es directamente proporcional al ratio asignado. La velocidad se espera que se acerque también a estas proporciones, aunque es comprensible que la relación no sea exacta por la naturaleza del medio utilizado.

escenario 1		
slice	**ratio**	**equipos**
2	50	A
3	50	B

escenario 2		
slice	**ratio**	**equipos**
2	75	A
3	25	B

Tabla 5.2: Configuración de los escenarios en la prueba 1.



(a) Airtime total consumido

(b) Velocidad promedio

Figura 5.4: Gráficos que presentan el airtime consumido y la velocidad promedio alcanzada por ambos nodos en el escenario 1 de la prueba 1, etapa I.

En la Figura 5.4 se pueden ver las gráficas del escenario 1, donde los valores de airtime son los mismos para los nodos A y B, mientras que las velocidades promedio en este caso son similares pero hay una diferencia de unos 5 Mbps. La Figura 5.5 muestra la velocidad de cada nodo en cada segundo del escenario 1, con resultados un tanto variables, que demuestran que ambos nodos en sus ejecuciones no pudieron alcanzar una estabilidad constante en sus transmisiones. Esto puede explicar que producto de estas variaciones uno de los nodos haya alcanzado una velocidad mayor en el total de la prueba. Como punto a favor se puede ver que más allá de estas variaciones que pueden ser producto del medio utilizado, ambos nodos obtienen resultados similares sostenidos en el tiempo, teniendo sus picos de diferencias a la par.

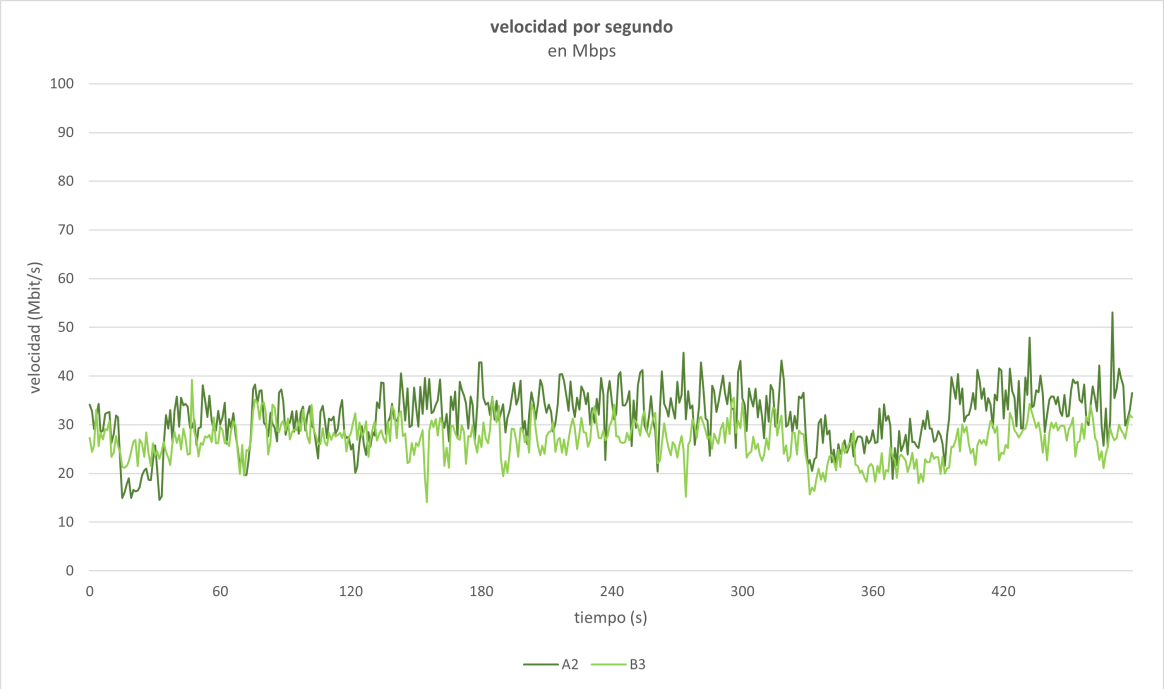
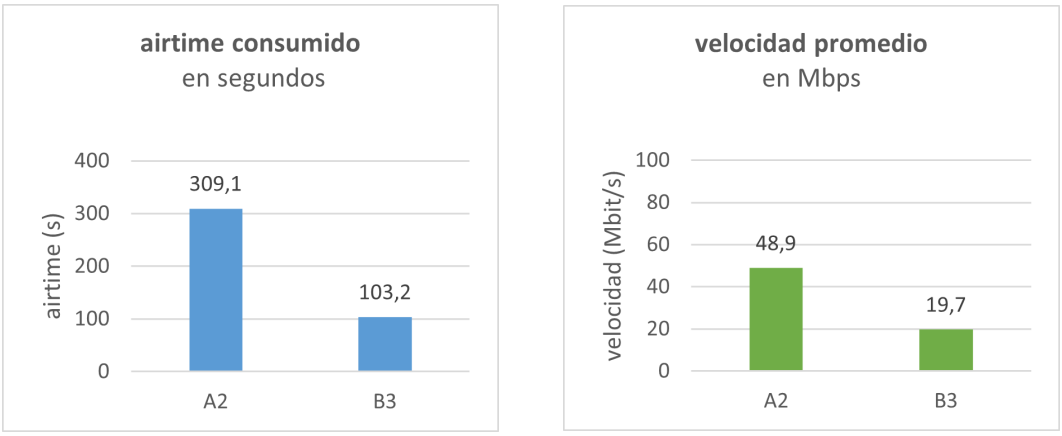


Figura 5.5: Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo, en el escenario 1 de la prueba 1, etapa I.



(a) Airtime total consumido

(b) Velocidad promedio

Figura 5.6: Gráficos que presentan el airtime consumido y la velocidad promedio alcanzada por ambos nodos en el escenario 2 de la prueba 1, etapa I.

Al igual que para el primer escenario, muestra una gráfica con la velocidad obtenida para el escenario 2, donde las distintas velocidades también se ven que son sostenidas en el tiempo.

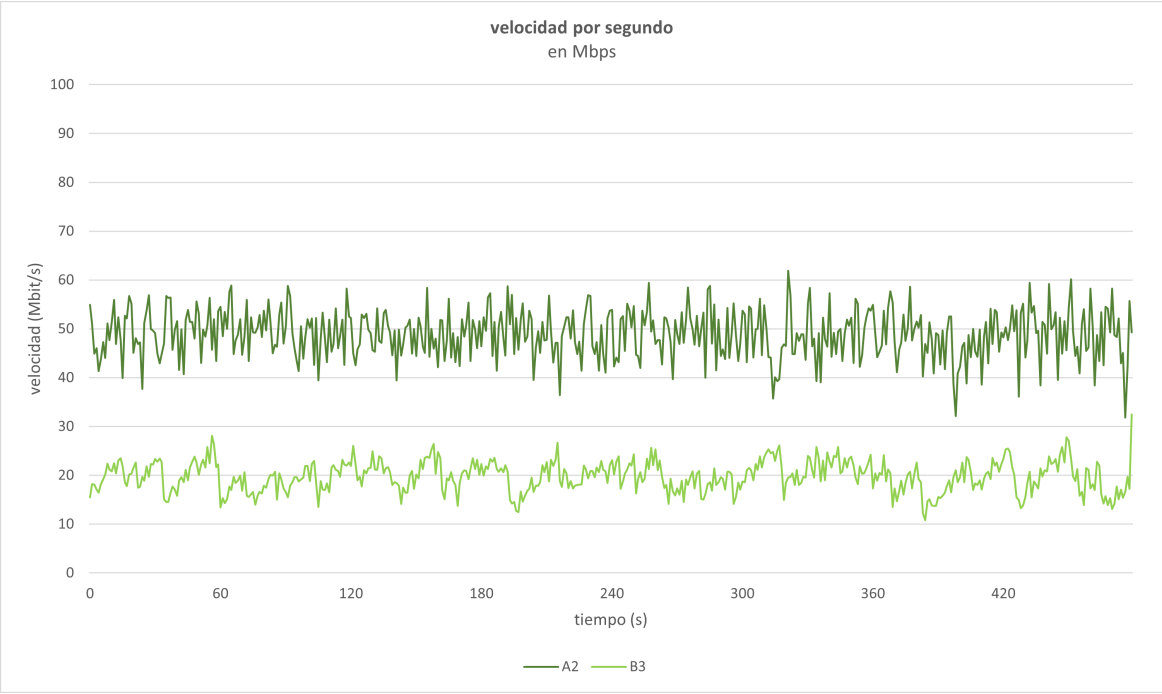


Figura 5.7: Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo, en el escenario 2 de la prueba 1, etapa I.

5.5.1.4. Prueba 2 - Nodos con múltiples slices

En esta prueba se busca evidenciar el funcionamiento del algoritmo mediante el uso de más de una slice por parte de un nodo. Para esto se asignan dos slices con distintos ratios a un nodo, mientras que una de esas slices es utilizada al mismo tiempo por un segundo nodo.

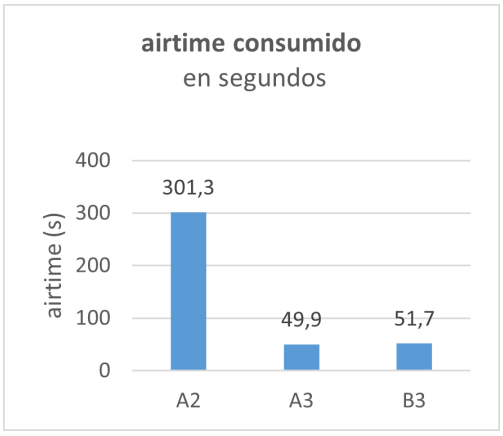
Aunque la prueba fue dividida en 3 escenarios, aquí se centró el análisis en el escenario número 2, que fue considerado el más representativo de esta prueba. Los escenarios 1 y 3 se encuentran presentados en el ya mencionado Anexo de pruebas. En este segundo escenario la slice de menor ratio es utilizada por dos nodos, mientras que la de mayor ratio es utilizada solamente por uno de ellos. Para esta prueba, las relaciones de airtime y velocidad deberán

corresponderse según la slice de la misma forma que en el escenario 2 de la prueba 1, ya que los ratios son los mismos, pero también deberán corresponderse individualmente para cada nodo de forma adecuada.

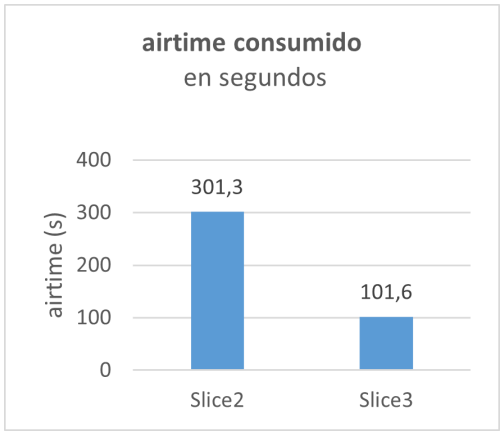
escenario 2		
slice	**ratio**	**equipos**
2	75	A
3	25	A, B

Tabla 5.3: Configuración del escenario 2 de la prueba 2.

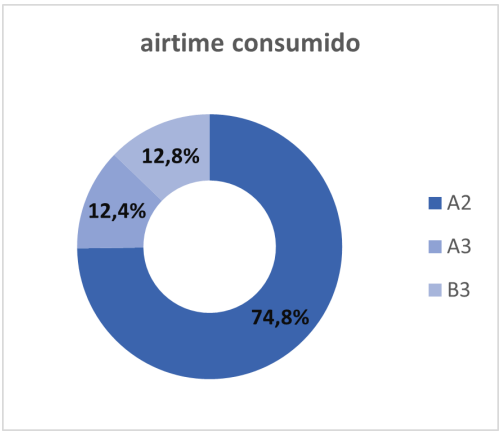
En las figuras 5.8, 5.9 y 5.10 se presentan gráficas de airtime consumido en segundos y en porcentaje total, velocidad promedio de la transmisión y velocidad por segundo. En los cuatro casos se diferencian dos tipos de gráficas, según el nodo y según la slice.



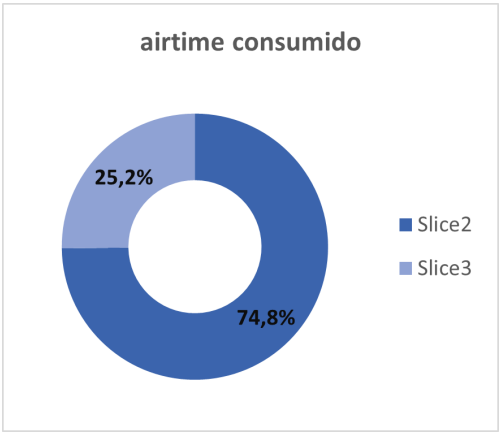
(a) Valores diferenciados por nodo y slice



(b) Valores agrupados por slice



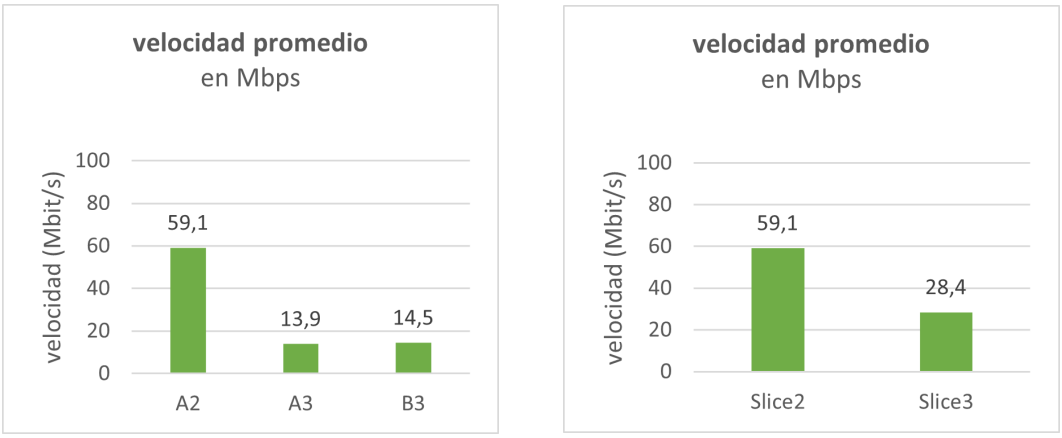
(c) Porcentajes por nodo y slice



(d) Porcentajes agrupados por slice

Figura 5.8: En estos cuatro gráficos se presenta el airtime consumido por los nodos y las slices involucradas en el escenario 2 de la prueba 2, etapa I.

Como se puede observar en las gráficas de airtime consumido, los valores se corresponden con los prometidos por el algoritmo según los ratios asignados y la cantidad de nodos por slice. El tiempo de transmisión del nodo se diferencia según qué slice utiliza y qué ratio tiene asignado la misma.



(a) Valores diferenciados por nodo y slice (b) Valores agrupados por slice

Figura 5.9: En estos dos gráficos se presenta la velocidad promedio alcanzada por los nodos y las slices involucradas en el escenario 2 de la prueba 2, etapa I.

Los valores de velocidad promedio se acercan a los esperados aunque en este caso la relación no es tan exacta como con el airtime, ya que la velocidad de la slice 2 no llega a triplicar la de la slice 3.

En las gráficas de velocidad por segundo se puede observar que los resultados son sostenidos en el tiempo más allá de algunos picos de subida o bajada aislados.

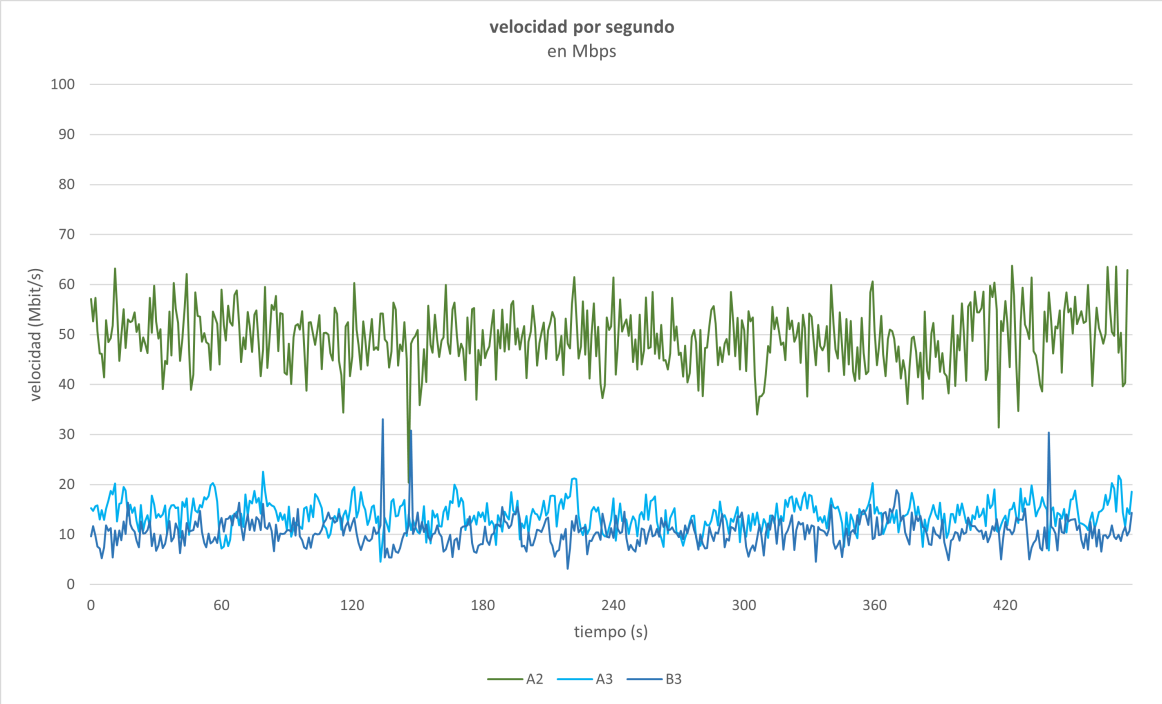


Figura 5.10: En Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo, en el escenario 2 de la prueba 2, etapa I.

5.5.1.5. Prueba 3 - Tres slices con ratio asignado.

Esta última prueba de la etapa utiliza tres slices. El objetivo principal es mostrar que, si una slice con ratio asignado no está en uso, las demás slices se repartirán ese ratio mientras tanto. En el escenario 1 el ratio del 20 % asignado a la slice 4 será repartido entre las slices 2 y 3. En el segundo escenario la slice 4 sí es utilizada, por lo tanto, las slices 2 y 3 utilizarán ahora solamente sus ratios asignados.

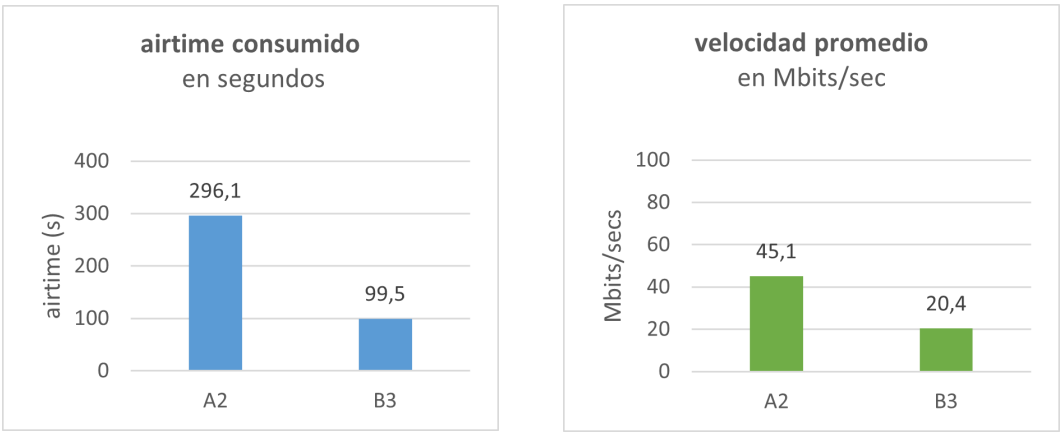
escenario 1		
slice	**ratio**	**equipos**
2	60	A
3	20	B
4	20	-

escenario 2		
slice	**ratio**	**equipos**
2	60	A
3	20	B
4	20	C

Tabla 5.4: Configuración de los escenarios de la prueba 3.

Observando los valores de airtime del primer escenario, se puede ver que son similares a los arrojados en el escenario 2 de la prueba 1, en que los ratios asignados eran 75 para la slice 2 y 25 para la slice 3. El ratio de valor 20 asignado a la slice 4 en desuso, se reparte entre las slices restantes de acuerdo al ratio de cada una de ellas, manteniendo la proporción que había de antemano. En este caso, como la slice 2 tiene tres veces más ratio que la slice 3, la slice 2 recibe tres veces más del ratio extra correspondiente a la slice 4. Por lo tanto la slice 2 recibe un ratio extra de 15 y la slice 4 un ratio extra de 5. De esta forma se llega a los mismos valores del escenario anteriormente mencionado de ratios 75 y 25.

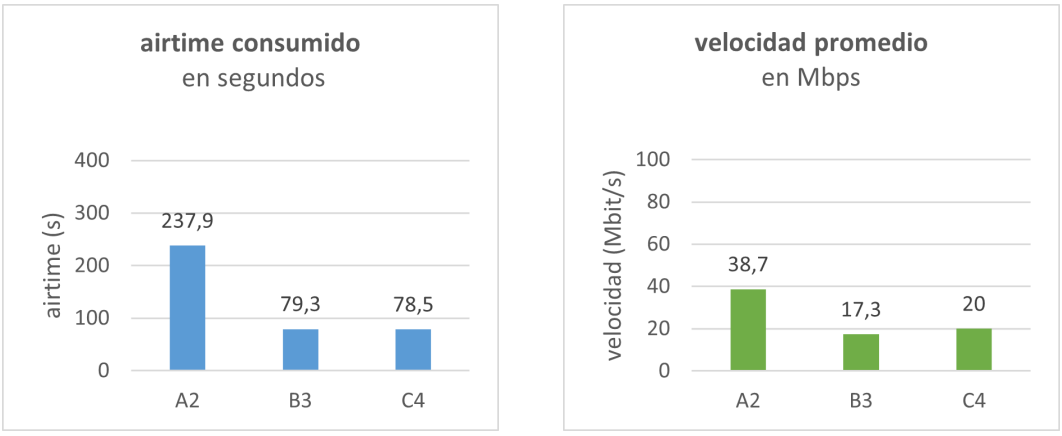
En el segundo escenario ya no hay slices con ratio asignado y que se encuentren en desuso, por lo que los tres nodos (cada una en la slice que tiene asignada) transmiten utilizando el airtime correspondiente al ratio de su slice. En los siguientes gráficos se puede observar este comportamiento, con la salvedad nuevamente de que los valores de velocidad de transmisión nuevamente son proporcionales a los ratios asignados pero no alcanzan los valores teóricos.



(a) Airtime total consumido

(b) Velocidad promedio

Figura 5.11: Gráficos que presentan el airtime consumido y la velocidad promedio alcanzada por ambos nodos en el escenario 1 de la prueba 3, etapa I.



(a) Airtime total consumido (b) Velocidad promedio

Figura 5.12: Gráficos que presentan el airtime consumido y la velocidad promedio alcanzada por ambos nodos en el escenario 2 de la prueba 3.

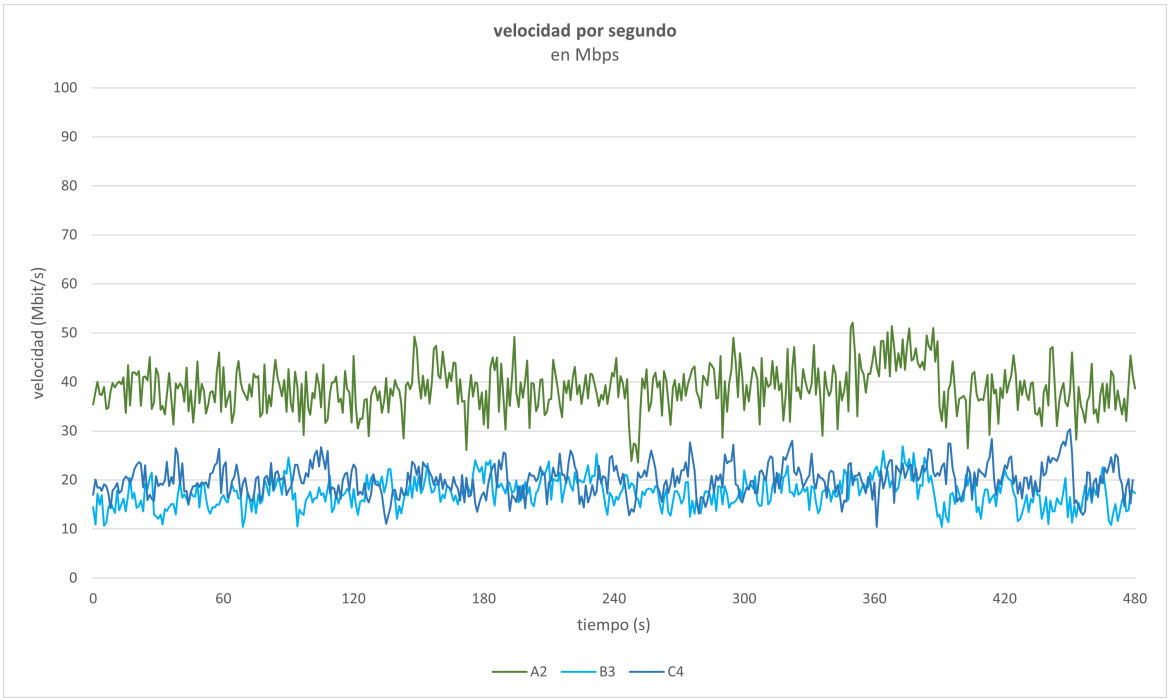


Figura 5.13: En Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo, en el escenario 2 de la prueba 3, etapa I.

5.5.1.6. Resultados obtenidos

Con respecto al airtime consumido por los nodos, se obtuvieron los valores esperados en todas las pruebas, con relación directa con los ratios asignados. Mientras tanto, los valores de velocidad obtenidos se acercan a los esperados pero no son exactos. Las slices con mayor ratio asignado alcanzaron las velocidades más altas, pero no superaron a las otras slices en la proporción exacta, como si ocurrió con el airtime. El ejemplo más claro es el escenario 2 de la prueba 1, donde la velocidad promedio del nodo A debería triplicar la del nodo B, sin embargo los resultados obtenidos fueron 48,9 Mbps y 19,7 Mbps respectivamente. De manera correcta la velocidad del nodo A supera la del nodo B, pero como se puede ver el nodo A debería haber alcanzado los 59,1 Mbps para triplicar la velocidad del nodo B. Este problema encontrado es posible que se deba a la cercanía de ambos nodos al AP, y la interferencia que ellos mismos generan, ya que como se podrá ver en la siguiente etapa, cuando ambos nodos se ubicaron a distancia 2 para esta misma prueba, la relación de velocidad sí se alcanzó de manera correcta.

5.5.2. Etapa II

Al igual que en la etapa anterior se utilizan los mismos 3 clientes, pero en esta oportunidad la distancia al AP a la que se encuentra cada nodo podrá ser distinta, según el escenario. Las distancias en que se ubiquen los equipos podrán ser las distancias 1 y 2, definidas al principio de la presente sección. Las pruebas a ejecutar en esta etapa son las mismas que en la etapa I, con la diferencia que en cada escenario se utilizan distintas distancias para los equipos, con el objetivo de demostrar que la distancia a la que se encuentren del AP, afecta su desempeño.

5.5.2.1. Resultados esperados

En esta segunda etapa se espera que los resultados de airtime sean los mismos que en la etapa anterior independientemente de la distancia de los nodos. Sin embargo, se espera que la velocidad alcanzada sí sea afectada negativamente por el incremento de distancia de los nodos hacia el AP. El hecho de que los valores de airtime se mantengan incambiados, se debe a que la naturaleza "justa" del algoritmo ATERR permite que se respete el tiempo de transmisión que le corresponde a cada equipo, sin importar la distancia a la que se encuentre. Los equipos a mayor distancia podrán transmitir menos datos en ese tiempo, por lo que la velocidad sí se verá disminuida.

5.5.2.2. Prueba 0

Al igual que en la etapa anterior, es una prueba inicial, en este caso para determinar los valores de base de la distancia 2. Se deberá observar que las velocidades registradas por los 3 equipos son menores a las registradas en la etapa anterior con distancia 1.

Se puede apreciar que los valores de airtime se corresponden con los de la etapa I para esta prueba y que los valores de velocidad son un poco inferiores (aproximadamente 64 Mbps cada nodo en esta etapa contra 77 Mbps de cada nodo en la etapa I). Como ya se mencionó, no fue posible aumentar la distancia para estas pruebas, lo que hubiera arrojado resultados más claros para visualizar las diferencias de velocidad. De todas formas se entendió que esta diferencia de velocidad es suficiente para mostrar que la distancia afecta la misma como establece el algoritmo.

También se presenta en la figura 5.15 una gráfica de velocidad por segundo para ver como las velocidades se mantienen cerca del promedio durante la duración de la prueba.

escenario 1		
slice	**ratio**	**equipos**
2	100	A

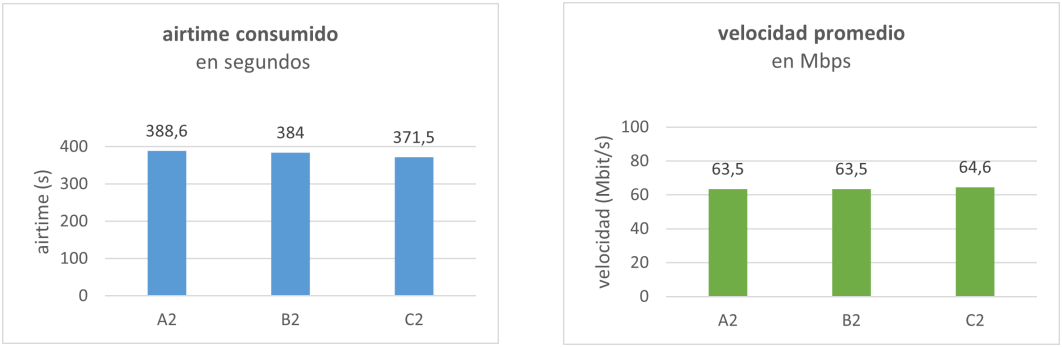
escenario 2		
slice	**ratio**	**equipos**
2	100	B

escenario 3		
slice	**ratio**	**equipos**
2	100	C

Tabla 5.5: Configuración de los escenarios en la prueba 0.

equipo	**distancia**
A	2
B	2
C	2

Tabla 5.6: Distancia de los equipos en la prueba 0.



(a) Airtime total consumido

(b) Velocidad promedio

Figura 5.14: Gráficos que presentan el airtime total consumido y la velocidad promedio de cada equipo, en sus respectivas transmisiones individuales de la prueba 0, etapa II.

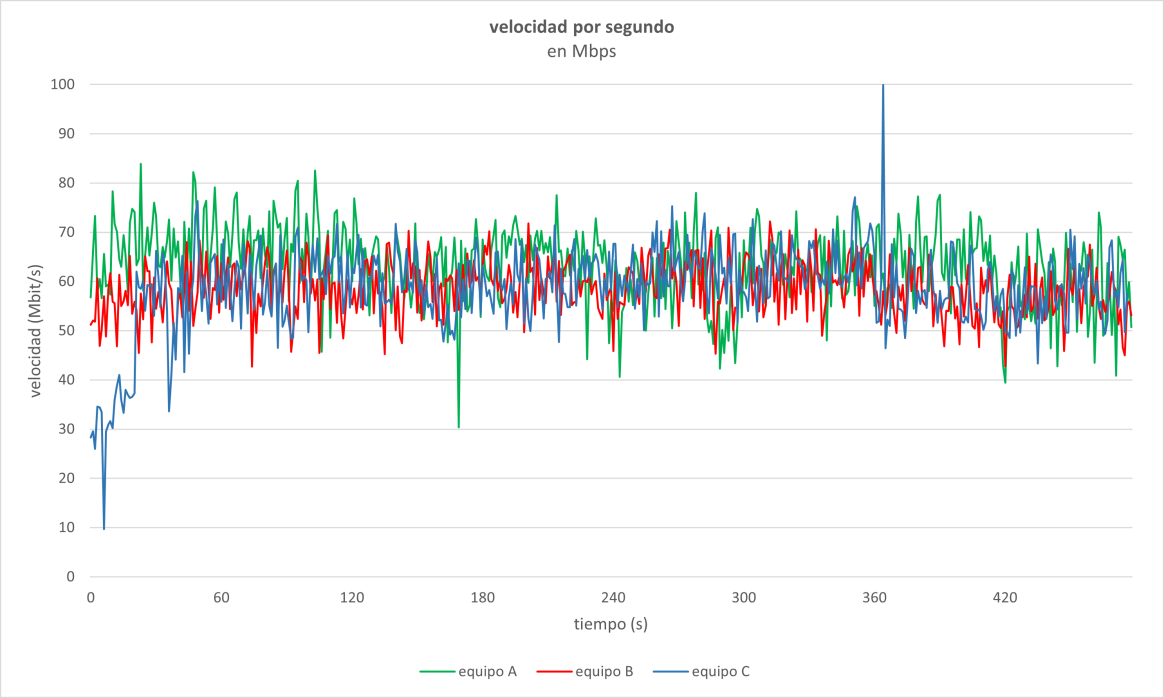
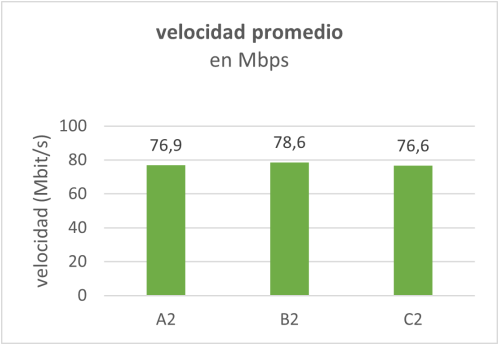
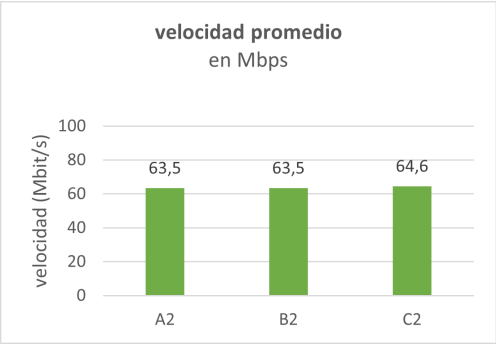


Figura 5.15: Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo, en sus respectivas transmisiones individuales de la prueba 0, etapa II.



(a) Valores de la etapa I



(b) Valores de la etapa II

Figura 5.16: Comparación de los gráficos de velocidad promedio en la prueba 0 de las etapas I y II.

5.5.2.3. Prueba 1

Esta prueba mantiene la distancia 2 para los dos equipos A y B, por lo que los resultados serán similares a la prueba 1 de la etapa anterior, ya que no hay distintas distancias entre ellos. Lo que si se espera es que las velocidades alcanzadas sean menores a las de la etapa I. En el escenario 1 los valores de airtime se mantienen, mientras que la velocidad se ve ligeramente disminuida. Para el escenario 2, como se puede observar en la gráfica 5.17 las velocidades también son menores que en la etapa anterior. A su vez se entiende que a mayor distancia, mejor se mantiene la proporción en las velocidades con respecto a los valores teóricos, ya que en este caso la velocidad de la slice 2 prácticamente triplica la de la slice 3, con velocidades de 45,8 Mpbs y 16,1 Mbps respectivamente.

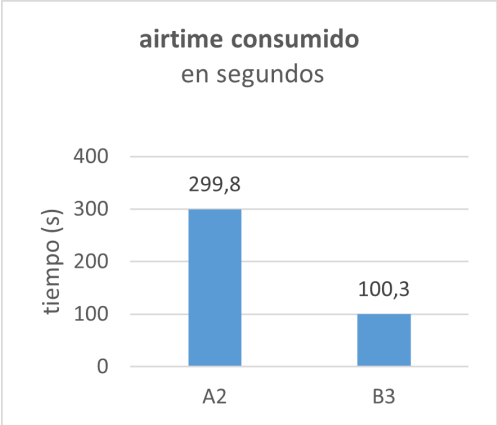
escenario 1		
slice	**ratio**	**equipos**
2	50	A
3	50	B

escenario 2		
slice	**ratio**	**equipos**
2	75	A
3	25	B

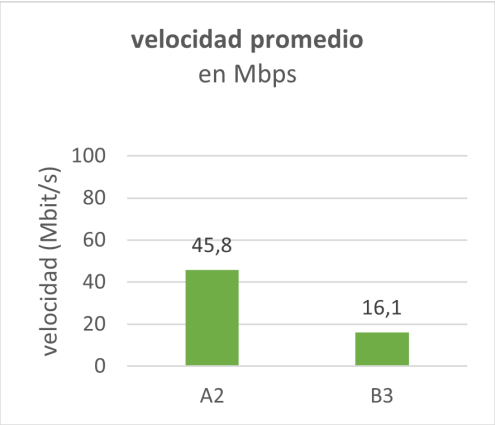
Tabla 5.7: Configuración de los escenarios en la prueba 1.

equipo	**distancia**
A	2
B	2

Tabla 5.8: Distancia de los equipos en la prueba 1.



(a) Airtime total consumido



(b) Velocidad promedio

Figura 5.17: Gráficos que presentan el airtime consumido y la velocidad promedio alcanzada por ambos nodos en el escenario 2 de la prueba 1, etapa II.

5.5.2.4. Prueba 2

Para la prueba 2 se involucran dos equipos con distancias distintas en cada escenario. En este caso, el equipo A se acerca al AP hasta la distancia 1, por lo que cada slice que transmita hacia el nodo A, deberá ver un incremento en su velocidad con respecto a la velocidad de las slices que transmiten en el nodo B, siempre teniendo en cuenta el airtime asignado a cada slice.

escenario 2		
slice	**ratio**	**equipos**
2	75	A
3	25	A, B

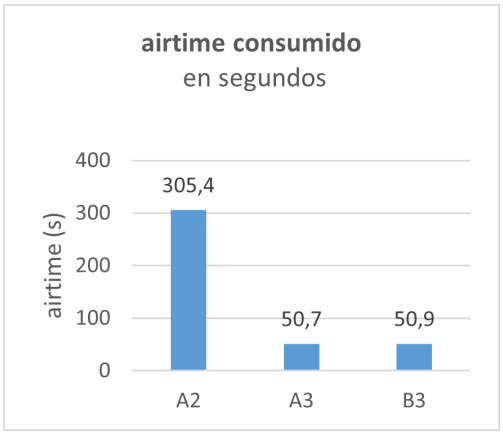
Tabla 5.9: Configuración del escenario 2 de la prueba 2.

equipo	**distancia**
A	1
B	2

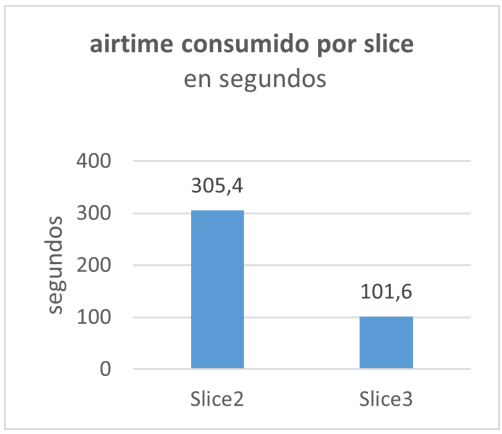
Tabla 5.10: Distancia de los equipos en la prueba 2.

Al igual que en la prueba 2 de la etapa I, el análisis se centra en el escenario 2. En este caso se puede ver que los valores de airtime se mantienen incambiados con respecto a la etapa anterior, sin embargo la velocidad se ve incrementada para el nodo A, tanto para la slice 2 (A2) como para la slice 3 (A3). La velocidad promedio del nodo A comparada a la etapa anterior creció de 49,7 Mbit/s a 65,6 Mbit/s y de 13,9 Mbit/s a 16,2 Mbit/s en las slices 2 y 3, respectivamente. La velocidad del nodo B en la slice 3 es menor que la de A3, ya que ambos comparten dicha slice pero el nodo B se encuentra a mayor distancia. Por lo tanto se puede afirmar que el comportamiento de las velocidades de transmisión al variar las distancias de los nodos es el esperado para esta prueba.

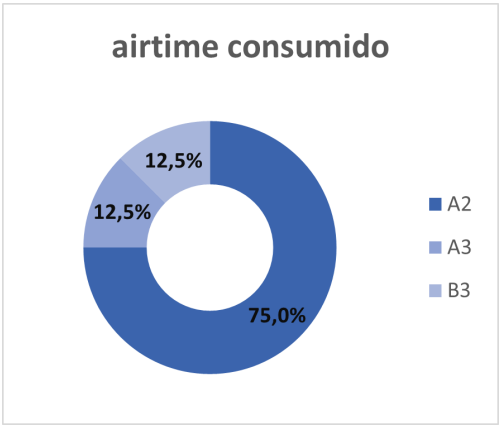
Como se ve en las gráficas de airtime consumido, los valores se corresponden por nodo y slice, obteniendo los mismos resultados que en la etapa I, ya que a pesar de haber nodos con distintas distancias para una misma slice, el airtime consumido no debe verse modificado con relación al ratio asignado.



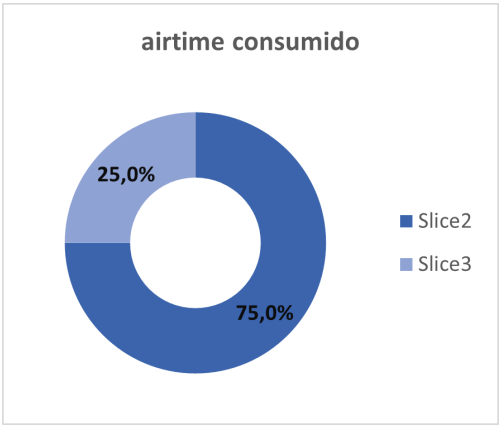
(a) Valores diferenciados por nodo y slice



(b) Valores agrupados por slice

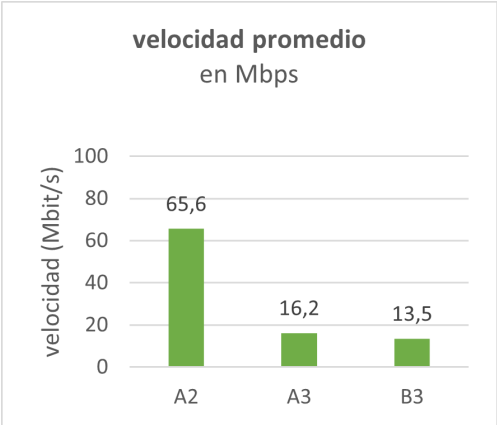


(c) Porcentajes por nodo y slice

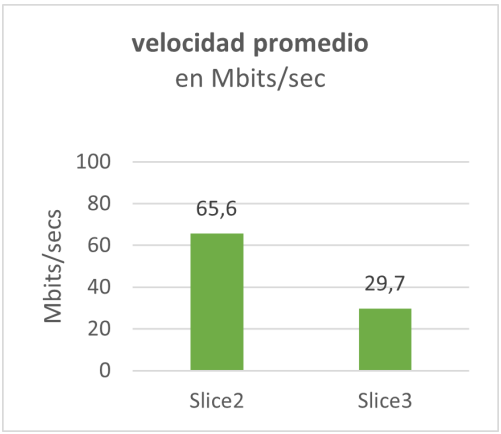


(d) Porcentajes agrupados por slice

Figura 5.18: En estos cuatro gráficos se presenta el airtime consumido por los nodos y las slices involucradas en el escenario 2 de la prueba 2, etapa II.



(a) Valores diferenciados por nodo y slice



(b) Valores agrupados por slice

Figura 5.19: En estos dos gráficos se presenta la velocidad promedio alcanzada por los nodos y las slices involucradas en el escenario 2 de la prueba 2, etapa II.

5.5.2.5. Resultados obtenidos

Los valores de airtime se mantuvieron incambiados con respecto a la etapa anterior, como era lo esperado. Se pudo apreciar que los valores de velocidad se vieron disminuidos cuando la distancia fue incrementada, a pesar de que la diferencia entre ambas distancias no era demasiado amplia. A su vez, las proporciones de velocidad obtenidas fueron mejores que las de la etapa I, debido a que los equipos se encontraban más distanciados del AP, evitando posibles interferencias.

5.5.3. Etapa III

El objetivo de la etapa III es monitorear en tiempo real el comportamiento de los equipos cuando hay cambios en la configuración de la prueba. Estos cambios se pueden dar cuando se modifica el ratio de una slice, o se mueve un nodo de lugar, por ejemplo. Para lograr este objetivo se ejecutan nuevos scripts de forma paralela, que permiten ejecutar comandos de comunicación con el módulo ATERR mientras se ejecutan las instrucciones iperf. De esta forma se puede obtener el airtime consumido por segundo por cada slice. Con estos nuevos valores de airtime, sumados a los de velocidad por segundo, se puede ver el comportamiento de los diferentes equipos y slices en el tiempo, mientras se ejecutan distintas configuraciones de forma ininterrumpida. Para esta etapa, cada prueba se divide en tres períodos distintos, dónde cada uno cuenta con una configuración particular pero los tres pertenecen a la misma transmisión contínua. Los gráficos de esta etapa, muestran para cada prueba el total de la transmisión, incluyendo todos los períodos de la misma. Por lo tanto se permite observar el momento exacto en que se cambia de un período al otro y varían los valores de velocidad y airtime. Cada período tiene una duración aproximada de 8 minutos.

5.5.3.1. Resultados esperados

En esta tercer etapa se espera mostrar la respuesta a los cambios de período en tiempo real, modificando ratios, asignaciones a las slices y distancias, mientras persiste la transmisión de los nodos. Se espera que los cambios de configuración de los distintos períodos se vean reflejados, según corresponda y de forma inmediata, en los resultados de airtime consumido y velocidad por segundo.

5.5.3.2. Prueba 0

La primera prueba parte de un período en que transmite únicamente el equipo A. El segundo período agrega la transmisión del equipo B y por último en el tercer período vuelve a dejar de transmitir, quedando solo el equipo A. Ambos equipos se encuentran a distancia 1 del AP y cuentan con un ratio del 50 % en la slice en que transmiten, slices 2 y 3 respectivamente. Cuando el equipo B no está transmitiendo, el equipo A utilizará el 100 % del tiempo de transmisión ya que es el único interesado, y cuando ambos transmitan los valores de airtime y velocidad por segundo deberán ser equitativos para ambos nodos.

período 1		
slice	**ratio**	**equipos**
2	50	A
3	50	-

período 2		
slice	**ratio**	**equipos**
2	50	A
3	50	B

período 3		
slice	**ratio**	**equipos**
2	50	A
3	50	-

Tabla 5.11: Configuración de la prueba 0.

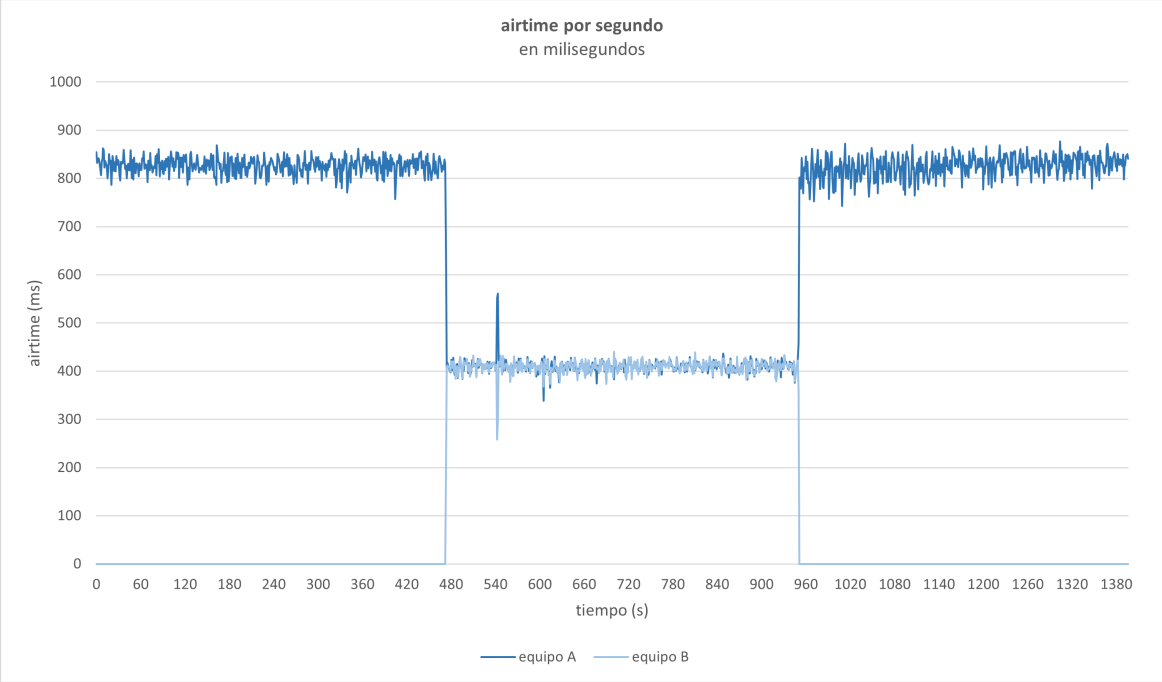


Figura 5.20: Gráfico que presenta el airtime consumido por los equipos en función del tiempo en la prueba 0, etapa III.

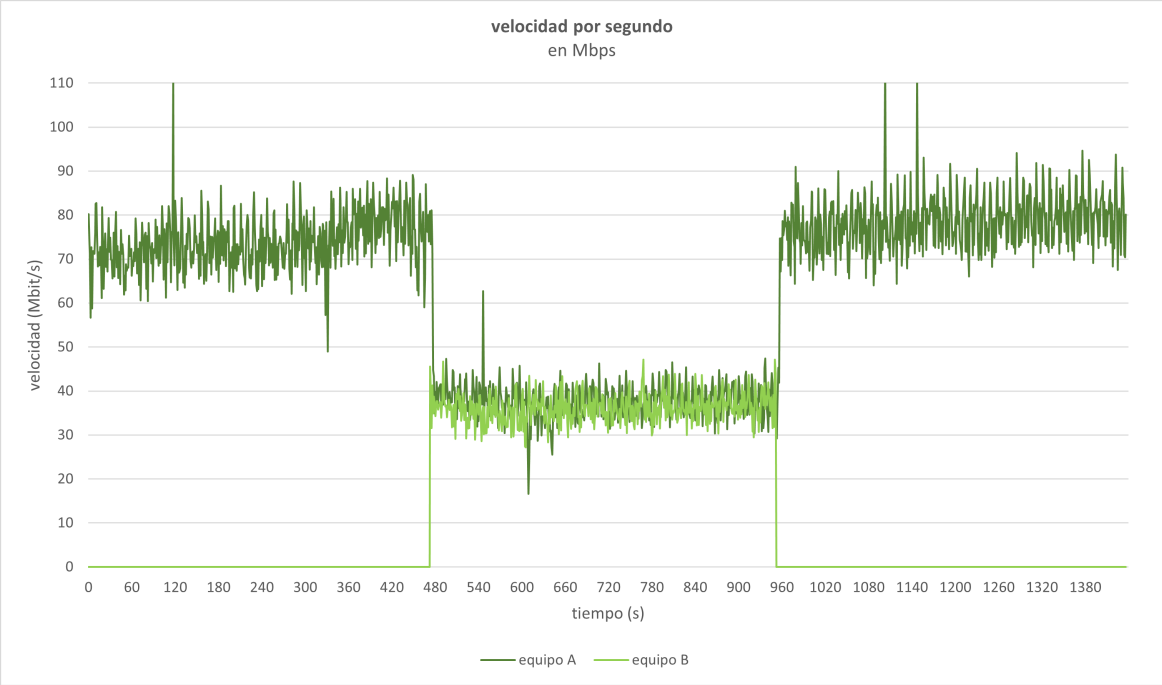


Figura 5.21: Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo en la prueba 0, etapa III.

En el período 1, el nodo A es el único involucrado en la transmisión, por lo que utiliza todo el airtime disponible (aunque la slice en que transmite tenga un ratio de 50, no se están utilizando las otras slices). Como se observa en la gráfica 5.20 el airtime por segundo oscila entre los 800 y los 900 msec y no alcanza los 1000 msec ya que no es posible en casos prácticos de la vida real, por diferentes motivos, siendo el principal el ruido originado en la transmisión. Para el segundo período, cuando se cumplen los 8 minutos desde el comienzo de la prueba, inicia su transmisión el nodo B en la slice 3, por lo que a partir de este momento ambas consumen el 50 % del airtime y también se reparten la velocidad de transmisión, que el nodo A la ve disminuida a la mitad con respecto al período anterior. El tercer período es igual al primero, por lo que se retoman los valores que se presentaron en este.

5.5.3.3. Prueba 1

En esta prueba los cambios de período están dados por la variación de distancia del equipo A. Ambos equipos transmiten durante los tres períodos y con el mismo ratio.

Durante el período 1 los dos equipos se encuentran a la distancia 2, por lo que están a la misma distancia del AP. Al pasar al período 2, el equipo A se acerca y se posiciona en la distancia 1, mientras ambos nodos siguen transmitiendo. Se podrá apreciar que el nodo A ve incrementada su velocidad de transmisión, mientras que el airtime por segundo permanece incambiado para ambos equipos. Para la etapa 3, se regresa el nodo A a su posición original (distancia 2) y los resultados pasarán a ser similares a los del período 1.

período 1	
equipo	***distancia***
A	2
B	2

período 2	
equipo	**distancia**
A	1
B	2

período 3	
equipo	**distancia**
A	2
B	2

Tabla 5.12: Distancia de los equipos en cada período en la prueba 1.

período 1		
slice	**ratio**	**equipos**
2	50	A
3	50	B

período 2		
slice	**ratio**	**equipos**
2	50	A
3	50	B

período 3		
slice	**ratio**	**equipos**
2	50	A
3	50	B

Tabla 5.13: Configuración de la prueba 1.

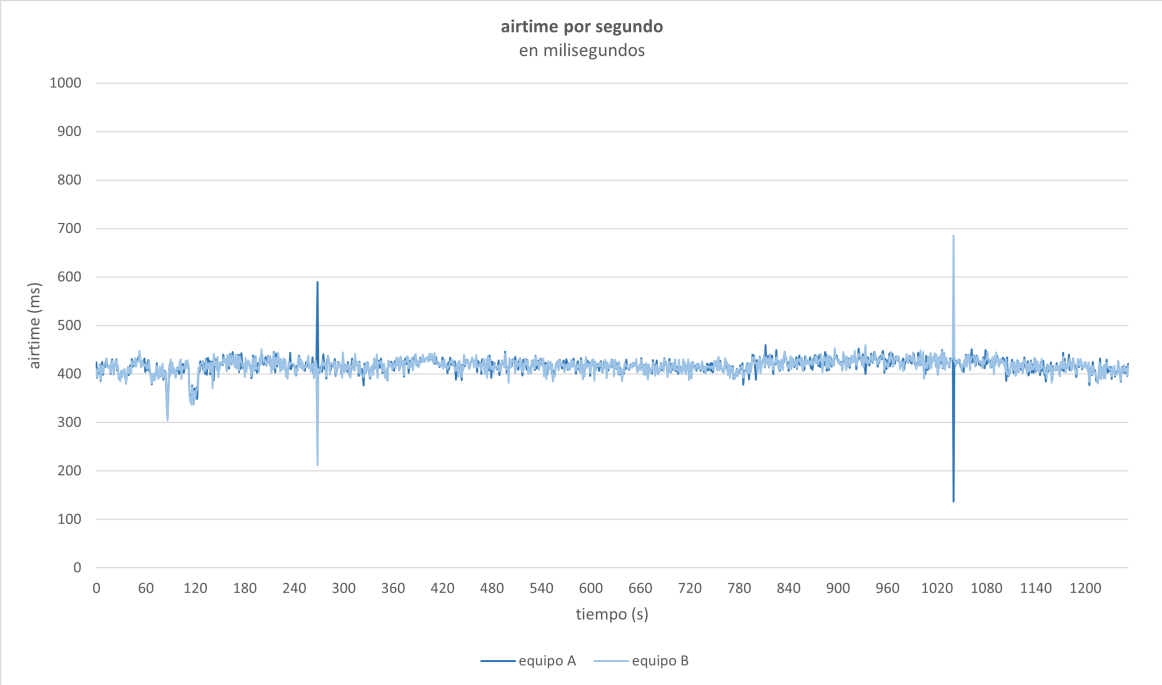


Figura 5.22: Gráfico que presenta el airtime consumido por los equipos en función del tiempo en la prueba 1, etapa III.

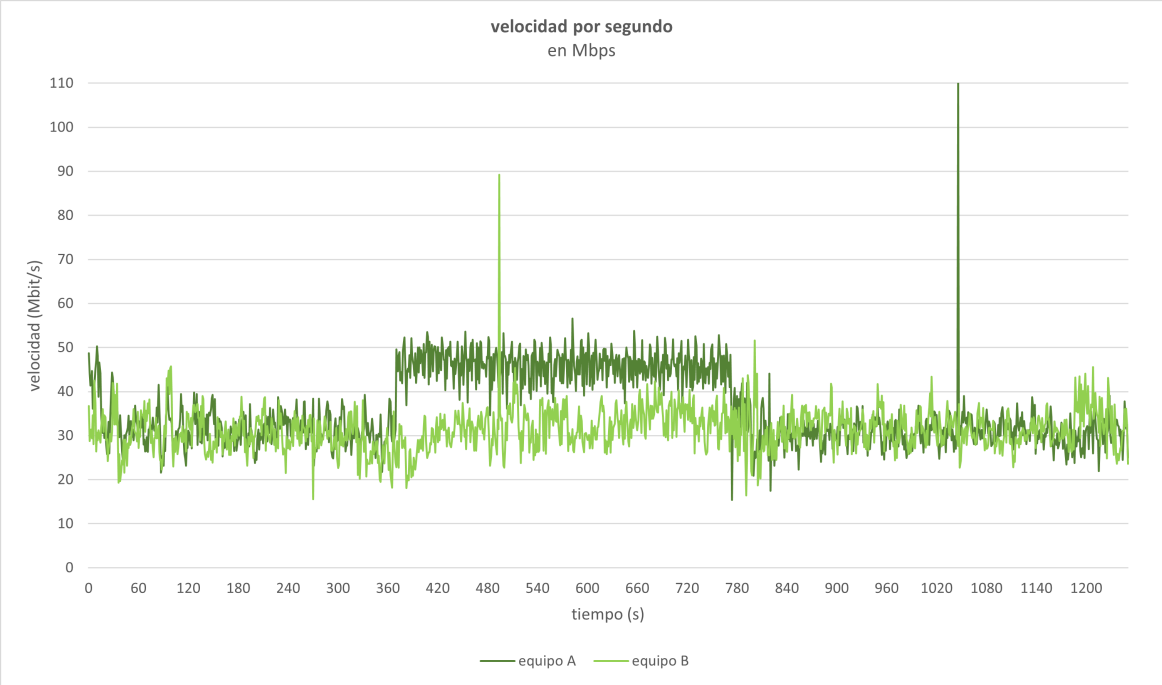
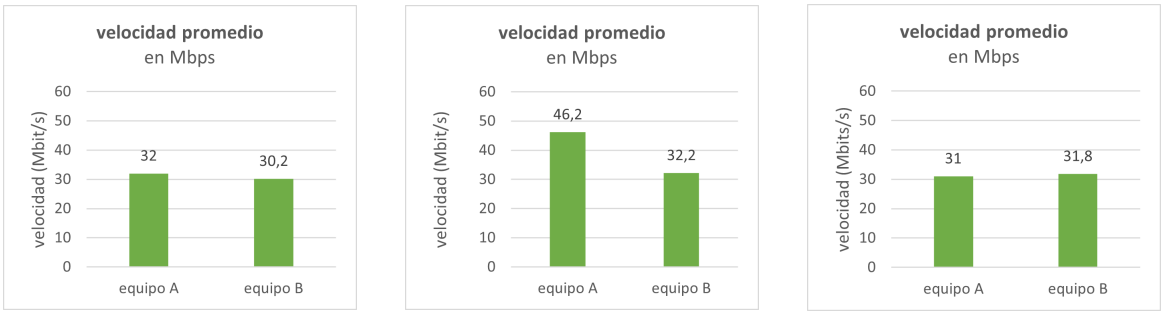


Figura 5.23: Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo en la prueba 1, etapa III.



(a) período 1 (b) período 2 (c) período 3

Figura 5.24: En estos dos gráficos se presenta la velocidad promedio alcanzada por los nodos A y B en cada uno de los períodos de la prueba 1, etapa III.

Ambos nodos transmiten la totalidad de la prueba en una slice cada uno con 50 de ratio, por lo que consumen el 50 % del airtime cada uno de principio a fin. La diferencia,

se encuentra en la velocidad de transmisión, a partir del período 2 cuando el nodo A es acercado al AP (hasta la distancia 1) y aumenta su velocidad un promedio de 14 Mbit/s. El nodo B no ve modificada su velocidad de transmisión al mover el otro nodo, ya que el tiempo con el que cuenta para transmitir y su posición permanecen incambiados. Este comportamiento muestra que se cumple la aislación entre slices, que no permite que se vean afectadas entre ellas. Esto evita que la alteración en la performance de una slice afecte el rendimiento de otra, como ocurre en este caso con las slices 2 y 3.

Al devolver el nodo A a la distancia 2 cuando comienza el período 3, se retoman correctamente los valores que ya se vieron en el primer período de la prueba. Para comparar estos valores de forma gráfica, se realizaron representaciones de la velocidad promedio según cada uno de los tres períodos, en la Figura 5.24. También se puede observar en este caso, que cuando los nodos A y B transmiten a la misma distancia y con el mismo ratio, las velocidades de ambos se asimilan más entre sí, que en pruebas anteriores con las mismas condiciones. Este cambio puede deberse a la ejecución de instrucciones iperf más extensas que en las etapas I y II, ya que le da mayor tiempo a los nodos a generar una mayor estabilidad en la transmisión. En las etapas anteriores cada escenario implicaba una ejecución iperf nueva por nodo/slice, mientras que en la etapa III es una única ejecución que cubre los tres períodos.

5.5.3.4. Prueba 2

En la prueba 2 ambos nodos transmiten ininterrumpidamente y sin modificar su posición, lo que si cambia es el ratio asignado a las slices 2 y 3, utilizadas por los nodos A y B respectivamente. En el primer período ambas slices tienen un ratio del 50 %, mientras que en los períodos siguientes una de las slices tiene el 75 % y la otra el 25 % y viceversa. Esta prueba es en definitiva la misma que la ejecutada en la prueba 1 de la etapa 1, únicamente con la diferencia de que aquí se realizan los cambios de ratio mientras los nodos siguen transmitiendo. En los gráficos 5.25 y 5.26 se puede ver como el airtime y la velocidad por segundo de cada período se corresponde a los valores obtenidos en cada escenario de la prueba anterior.

período 1		
slice	**ratio**	**equipos**
2	50	A
3	50	B

período 2		
slice	**ratio**	**equipos**
2	75	A
3	25	B

período 3		
slice	**ratio**	**equipos**
2	25	A
3	75	B

Tabla 5.14: Configuración de la prueba 2.

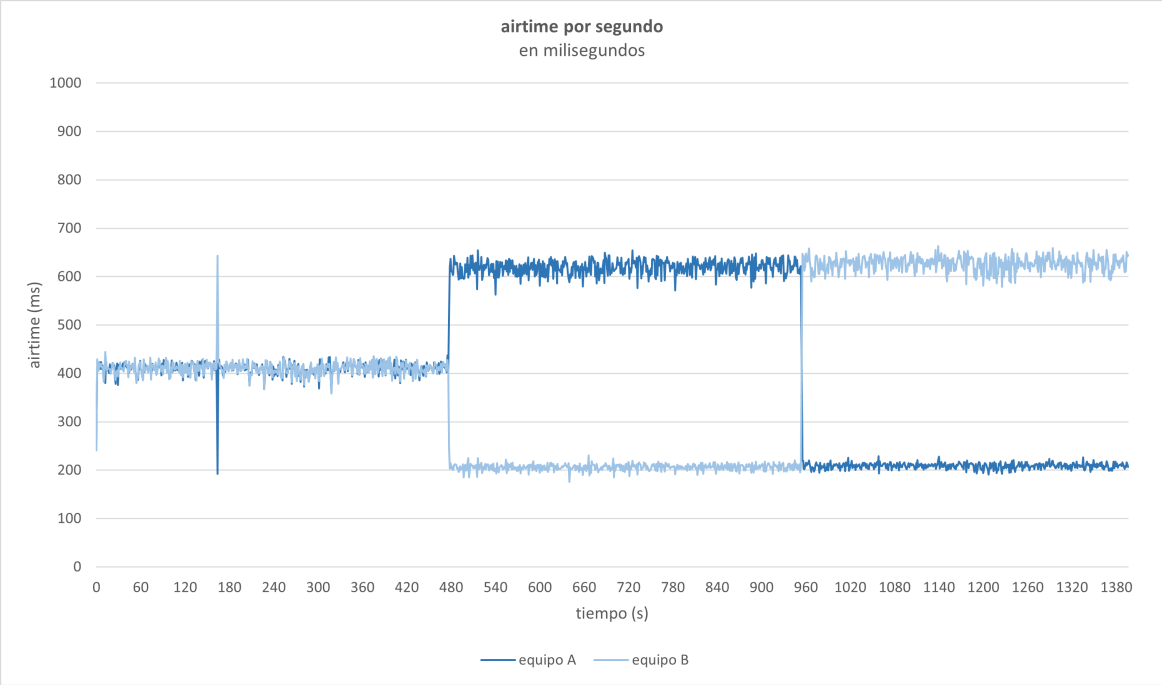


Figura 5.25: Gráfico que presenta el airtime consumido por los equipos en función del tiempo en la prueba 2, etapa III.

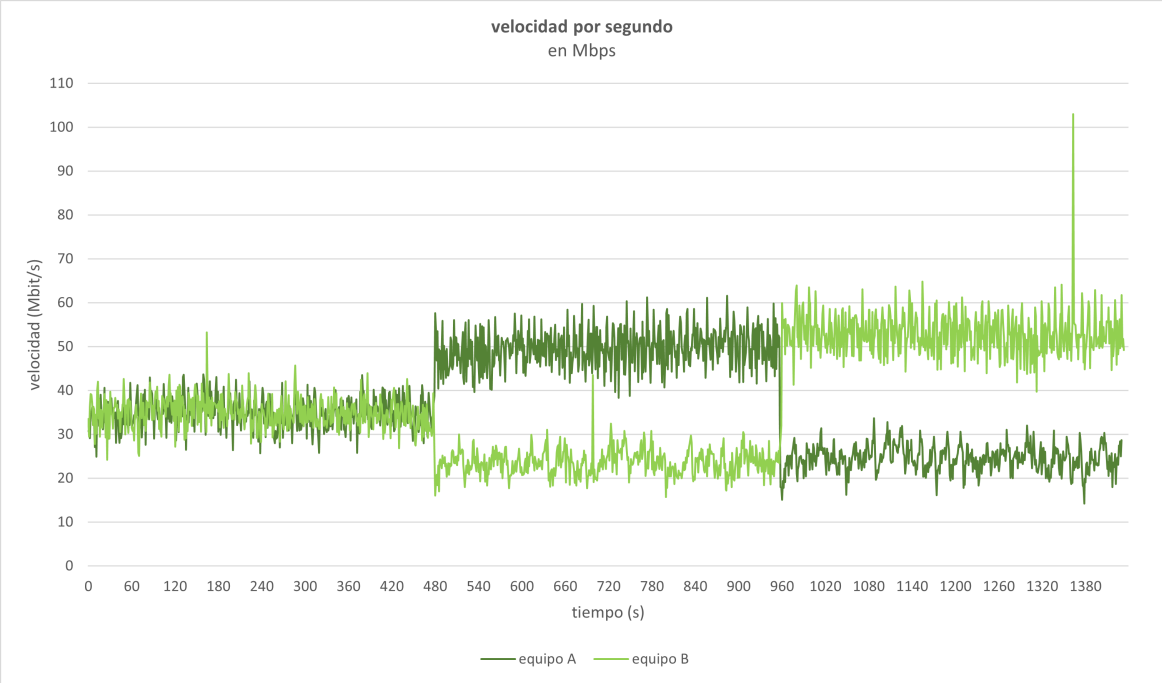


Figura 5.26: Gráfico que presenta la velocidad alcanzada por los equipos en función del tiempo en la prueba 2, etapa III.

5.5.3.5. Resultados obtenidos

Los resultados obtenidos fueron los esperados, se logró mostrar que el algoritmo responde correcta y rápidamente a los cambios de período planteados. Los resultados se muestran de forma gráfica, que permite una mayor claridad para ver los cambios bruscos en el comportamiento de las transmisiones en el momento exacto que se producen los cambios.

5.6. Comparación con resultados de simulación

A pesar de no haber realizado las mismas pruebas que se hicieron en la simulación del artículo "Slicing in WiFi Networks Through Airtime-based Resource Allocation" [1], se pueden hacer algunas comparaciones que reflejan resultados acordes en ambos casos, aunque con enfoques diferentes. Todos los datos correspondientes a la referida simulación fueron tomados del artículo mencionado.

Para las pruebas de la simulación que se estudiarán en esta sección, se utilizaron 3 slices, donde las slices 1 y 2 utilizan un 20 % del airtime cada una y la tercera un 60 %. Cada slice es utilizada por 4 clientes cada una, lo que difiere con la cantidad de clientes utilizados en estas pruebas prácticas, pero la comparación que se verá a continuación es desde el punto de vista del comportamiento de las slices, y como ya se vio anteriormente es independiente de la cantidad de clientes que participen en cada slice.

La primera prueba simulada que se presenta para esta configuración es una ejecución de tráfico TCP sin alteraciones y con las 3 slices transmitiendo a de forma ininterrumpida. A pesar de que en este caso se utiliza TCP, las condiciones de la prueba son las mismas que las presentadas en la etapa I - prueba 3 - escenario 2 (3 slices con ratios de 60, 20 y 20). En la Figura 5.27 se pueden ver los resultados obtenidos en el artículo de la simulación, donde en este caso los resultados están presentados como airtime por segundo, mientras que, en la prueba comparativa de este trabajo, los resultados se pueden ver como el airtime total consumido por cada slice a través de la prueba. Observando esta gráfica, se ve que el airtime por segundo no varía demasiado (más allá de los primeros segundos cuando se realiza el handshake de TCP) por lo que se puede concluir que el airtime total consumido va a tener valores realmente aproximados a los mostrados en la Figura 5.28, que reflejan lo obtenido en la prueba práctica.

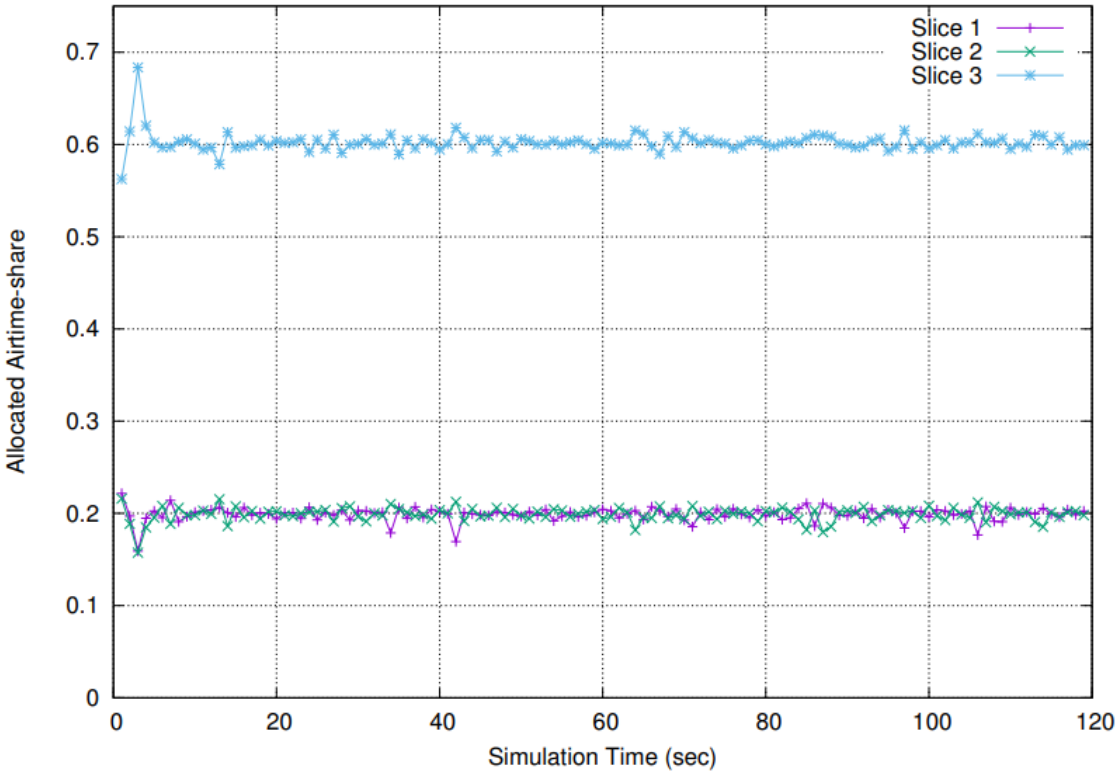
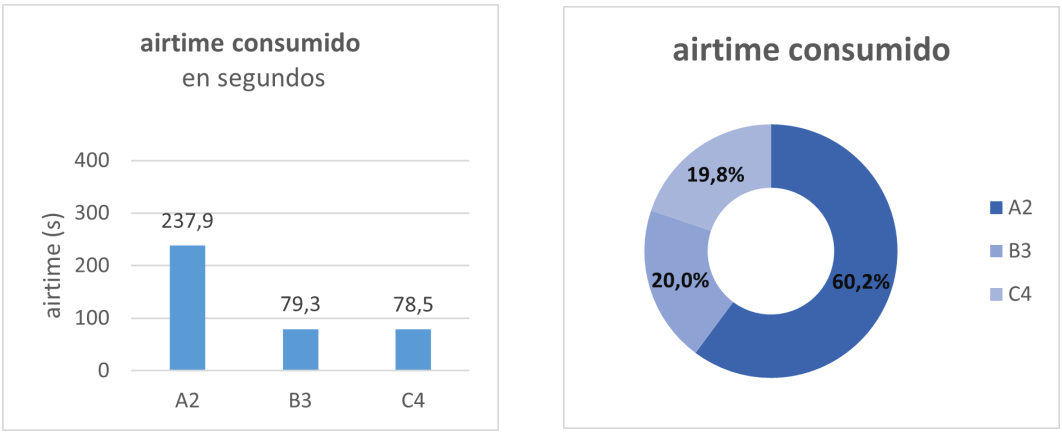


Figura 5.27: Gráfico correspondiente a la primera simulación. Fuente [1]



(a) Valores de airtime totales

(b) Valores porcentuales de airtime

Figura 5.28: Gráficos que presentan los valores de airtime de la etapa I - prueba 3 - escenario 2, con el objetivo de ser comparados con los resultados de simulación

La segunda prueba simulada sí usa UDP al igual que las pruebas prácticas del presente trabajo, y en este caso consiste en variar el flujo de las slices durante la transmisión, pero siempre manteniendo los mismos valores de airtime definido. Como se observa en la grafica 5.29 los resultados están diferenciados en distintas secciones. En la primera sección (segundos 0 a 30) las slices utilizan la totalidad del airtime y estos resultados coinciden con los vistos en la prueba anterior. En las siguientes 2 secciones de 30 segundos, el tráfico de una de las slices es limitado, por lo que las otras slices se reparten ese tráfico restante que no utiliza la slice a la que le correspondería. En estos casos los porcentajes de ratio no coinciden con ninguna de las pruebas prácticas, pero sí representan el espíritu de la etapa I - prueba 3 - escenario 1, donde una de las slices con 20 % se deja sin utilizar, para mostrar como las otras dos hacen uso de ese airtime sobrante.

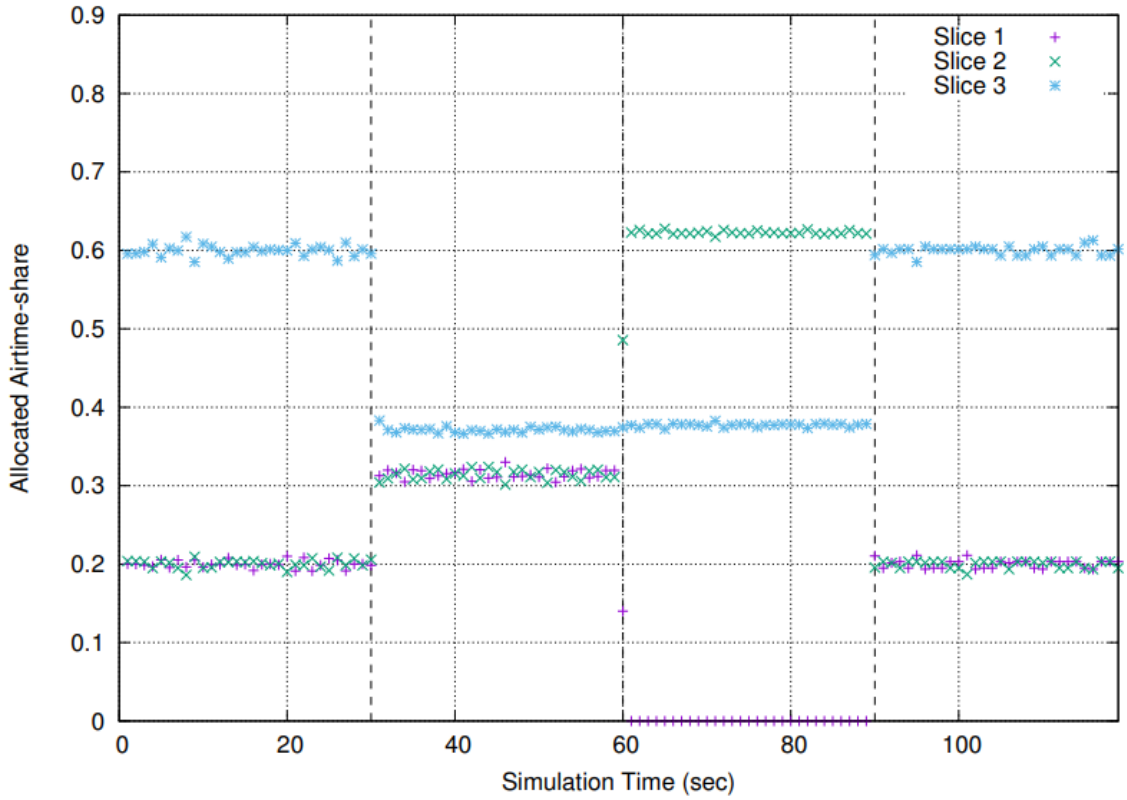


Figura 5.29: Gráfico correspondiente a la segunda simulación. Fuente [1]

La forma en que se presentan los resultados gráficos de la simulación es comparable a los resultados de la etapa III, a pesar de que representan escenarios de configuraciones distintas. En la etapa III se realizaron gráficas de airtime por segundo que reflejan como los resultados se mantienen acordes al ratio asignado de forma estable a través del tiempo, y sobre todo muestran que los cambios en los resultados son casi instantáneos al modificar algún ratio en tiempo real, como ocurre en la Figura 5.29 de la simulación.

5.7. Pruebas futuras

Algunas pruebas que podrían ser interesantes y no fueron alcanzadas en este proyecto, son aquellas que involucren una mayor cantidad de nodos conectados al AP y que los mismos puedan encontrarse a distancias más amplias. De esta forma se logrará probar el

funcionamiento del algoritmo en un ambiente más cercano a la realidad de un uso práctico, que en el presente conjunto de pruebas se encuentra acotado.

Por último, fueron valoradas pruebas que muestren como afecta el uso de las slices a distintas aplicaciones multimedia, como pueden ser transmisiones de audio o video, pero finalmente quedaron fuera del alcance del proyecto, por lo que se entiende que serán útiles en futuras pruebas.

Capítulo 6

Conclusiones y trabajo a futuro

6.1. Conclusiones

En el presente trabajo se logró concretar la implementación de un prototipo funcional del algoritmo ATERR, tal cual estaba planteado como objetivo principal. De la misma forma, se implementó una aplicación básica de consola que permite al usuario gestionar los valores de configuración deseados para los clientes de la red. El prototipo realizado se puede considerar una primera versión satisfactoria, luego de haber respondido de forma correcta ante un conjunto de pruebas que abarca distintos escenarios en un ambiente práctico.

A diferencia de las simulaciones realizadas en la presentación del algoritmo [1], para obtener un prototipo real, se contó con la dificultad de asociar cada paquete a la cola correspondiente a la slice correcta. Para enfrentar este desafío se contó con la colaboración del trabajo presentado por Toke Høiland-Jørgensen [16], donde se introduce el concepto TID, el cual fue utilizado para implementar las slices. Beneficiándose del hecho de que la estructura para el manejo de colas de paquetes por TID ya se presentó desarrollada [19], se pudo adaptar dicha estructura para manejar las slices, y haciendo uso del campo TOS de IPv4 asignar cada paquete a la slice indicada.

El principal aspecto positivo de las pruebas ejecutadas fue que todos los valores de airtime obtenidos fueron los esperados. Sin embargo, en contraparte los valores de velocidad no siempre coincidieron con los valores teóricos. Más allá de que las slices de mayor ratio obtienen mayor velocidad, las proporciones no fueron las deseadas en todos los casos.

Si se comparan los resultados obtenidos en la simulación con los de las pruebas prácticas de este trabajo, se puede resaltar que el algoritmo en ambos casos responde de forma

semejante. Esto se aprecia fundamentalmente en dos aspectos: los valores de airtime de las slices se corresponden con los ratios asignados y los cambios de ratios se ven reflejados de manera inmediata mientras las transmisiones de datos se mantienen ininterrumpidas.

Como conclusión general el objetivo del proyecto fue cumplido, logrando la implementación de un prototipo en su primera versión funcional. Esta versión es básica pero apta para demostrar el funcionamiento del algoritmo ATERR en un ambiente práctico.

6.2. Trabajo a futuro

Como se vio en el capítulo de pruebas, hay escenarios interesantes que no llegaron a validarse y se recomiendan como punto de partida para la continuidad de este trabajo.

La implementación del algoritmo fue realizada con estructuras de datos que siguen la arquitectura utilizada por el driver ATH9K y se buscó una utilización prolija de memoria. Sin embargo, el foco del desarrollo no estuvo puesto en la optimización máxima del código, sino en su funcionalidad. Es por esto que se cree que el módulo podría ser optimizado por parte de desarrolladores más experimentados en el área de sistemas operativos, y en el kernel de Linux particularmente.

Otra mejora a futuro, fundamentalmente pensando en una versión comercial del producto y que quedó por fuera del alcance, es la implementación de una interfaz gráfica para el usuario administrador, preferentemente web, que ofrezca una forma más confortable y clara para gestionar los clientes y sus recursos.

Capítulo 7

Lecciones aprendidas

La estimación inicial del tiempo de ejecución del proyecto fue errónea, encontrando complicaciones en varias etapas del mismo, que llevaron a incrementar la duración total del trabajo, pero pudieron ser superadas, a través de una experiencia ardua pero que resultó enriquecedora en muchos aspectos.

En un primer momento de la investigación previa al desarrollo de las funcionalidades, se buscó una comprensión del kernel a nivel general que resultó innecesariamente profunda en ciertas áreas, y que no terminaron aportando a la realización del trabajo. Tras enfocar la investigación en temas más puntuales y estrechamente relacionados al área de las redes inalámbricas, se aceleró la curva de aprendizaje y se pudo ver que el enfoque inicial no era el mejor, más allá de que el driver ATH9K cuenta con escasa documentación.

La compilación customizada del kernel y sus módulos fue un trabajo complejo en sus primeras etapas, mientras que acorde se fue avanzando en la implementación se ganó experiencia en el proceso y se logró disminuir los tiempos gradualmente.

Con respecto a la depuración del código, no se cuenta con herramientas avanzadas de debugging que indiquen posiciones exactas de excepciones o errores en el manejo de memoria. Por esto la depuración del código fue más lenta que en un ambiente típico de desarrollo, debiendo incluso restaurar el sistema operativo ante ciertos errores que lo volvieran infructuoso.

La etapa de pruebas también fue un desafío en todas sus etapas, donde poco a poco se fue mejorando el ambiente de trabajo y superando problemas encontrados por interferencias externas, o inconsistencia en los resultados. La posibilidad de contar con tres routers exactamente iguales como clientes en las pruebas finales, fue fundamental para la correcta aplicación de las pruebas, ya que en etapas previas la utilización de distintos equipos

como clientes (notebooks y celulares personales) afectaba las velocidades de transmisión, debido a las diferencias de capacidad de procesamiento en los equipos según el potencial del hardware con el que contaban.

Desde el punto de vista del crecimiento personal, se logró un entendimiento tal del funcionamiento del kernel, que permite agregar y modificar funcionalidades dentro de un módulo encargado del funcionamiento de la red inalámbrica.

Apéndice A

Anexos

A.1. Secciones de GitLab

Pruebas finales En esta carpeta se encuentra todo relacionado a las mismas, los scripts necesarios para la ejecución, instructivos de cómo realizarlas y los datos en bruto obtenidos.

Anexo de pruebas También se cuenta con un anexo de pruebas donde se describe la totalidad de las pruebas ejecutadas, y se presentan los resultados de todas ellas de forma gráfica en archivos .xlsx. Este anexo será referenciado en la sección de pruebas, para referirse a aquellas que quedaron excluídas del presente informe

Comunicación Esta carpeta contiene todos los archivos referentes a la implementación de la interfaz de usuario.

Código para pruebas finales

Se definió una rama llamada pruebas finales, que cuenta con el código en el estado exacto en que se realizaron las pruebas. Esta versión difiere del código final en dos aspectos:

- En primer lugar cuenta con funciones y salidas de consola propias de la etapa de desarrollo que no aportan valor real al código final resultante, por lo que fueron quitadas en la versión principal.

- En segundo lugar, se quitó la ejecución del código que controla el airtime en recv.c. Esto se hizo con el objetivo de despreciar el consumo de airtime al momento de recibir paquetes por parte del AP, para centrar el cálculo en la parte realmente importante para los resultados, que es el envío de datos por parte el AP a los clientes, lo cuál ocurre en xmit.c.

  Se piensa que esto no tuvo un impacto importante los resultados, ya que los envíos de los clientes al AP durante las pruebas, no deberían ser más que unos pocos datos de control. De todas maneras, se conserva en esta rama aparte la versión exacta con que se ejecutaron las pruebas.

A.2. Guía de trabajo

Requisitos del dispositivo

- Debe contar con una tarjeta de red inalámbrica con chipset Atheros que utilice el driver ATH9K [9].
- Sistema operativo Linux con una versión de kernel 4.11 o superior. Para el presente proyecto se trabajó con Ubuntu 16.04 y kernel 4.15 (en particular la versión 4.15.0-29-generic, que se utiliza a modo de ejemplo para los comandos necesarios)

La primera parte de la guía debe realizarse al menos una vez. Luego cada vez que se desee modificar únicamente el módulo ATH9K, se siguen solo los pasos de la segunda parte.

El código fuente a partir del cual se genera el nuevo kernel y que se usa como base para la implementación se obtiene desde el repositorio git oficial del kernel de Linux [28].

Primera parte

Se listan los pasos a seguir para obtener una versión modificada del kernel:

1. sudo apt-get update
2. sudo apt-get install git libssl-dev
3. cd
4. git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
5. cd linux-stable
6. git checkout v4.15
7. cp /boot/config-4.15.0-29-generic .config
8. make olddefconfig
9. make -j3
10. sudo make modules_install install

11. sudo reboot

El nuevo kernel compilado se encontrará bajo el nombre 4.15.0+ y será accesible luego de reiniciar el equipo [1]. Opcionalmente luego del paso 7 se pueden modificar archivos del código fuente como se describe en la segunda parte.

Segunda parte

Luego de ingresar al kernel 4.15.0+ se pueden modificar los archivos del código fuente para el módulo ATH9K, ubicados en /git/linux-stable/drivers/net/wireless/ath/ath9k/ [2].

Para compilar los cambios realizados sobre el código fuente de ATH9K y volver a instalar el módulo, se pueden seguir los siguientes pasos y luego de reiniciar el equipo se verán reflejados los cambios sobre el mismo kernel 4.15.0+.

1. cd /linux-stable
2. sudo make M=drivers/net/wireless/ath/ath9k/
3. sudo make modules_install M=drivers/net/wireless/ath/ath9k
4. sudo reboot

Para obtener la versión funcional del prototipo implementado en este trabajo, se pueden descargar los archivos modificados desde el repositorio GitLab y sustiturilos por los del repositorio original. Asumiendo que se clonó el repositorio GitLab en la carpeta home, se pueden sustituir los archivos con el siguiente comando: cp -aTr /aterr_/src/ath9k /linux-stable/drivers/net/wireless/ath/ath9k

[1] Para poder seleccionar de una lista con cuál kernel iniciar, se debe editar el archivo /etc/default/grub y luego ejecutar 'sudo update-grub'.

[2] Se pueden agregar salidas en el log de los módulos mediante el comando 'printk' para verificar mediante el comando 'dmesg' que el módulo modificado se haya cargado de forma correcta.

Bibliografía

[1] M. Richart, J. Baliosian, J. Serrat, J. Gorricho, R. Agüero y N. Agoulmine. «Resource allocation for network slicing in WiFi access points». En: 13th International Conference on Network y Service Management (CNSM 2017). IEEE, nov. de 2017. DOI: 10.23919/CNSM.2017.8256046.

[2] *Prototipo ATERR*. 2021. URL: https://gitlab.fing.edu.uy/ignacio.prandi/ateer_.

[3] M. Richart, J. Baliosian, J. Serrat y J.L Gorricho. «Resource Slicing in Virtual Wireless Networks: A Survey». En: *IEEE Transactions on Network and Service Management* (sep. de 2016), págs. 462, 476. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2597295.

[4] *ATH9K products - Atheros chips*. URL: https://wireless.wiki.kernel.org/en/users/drivers/ath9k/products.

[5] B. P. Crow, I. Widjaja, J. G. Kim y P. T. Sakai. «IEEE 802.11 Wireless Local Area Networks». En: *IEEE Communications Magazine* 35.9 (sep. de 1997), págs. 116-126. ISSN: 0163-6804. DOI: 10.1109/35.620533.

[6] *Cisco Annual Internet Report (2018–2023) White Paper*. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[7] GSMA. *An Introduction to Network Slicing*. 2017. URL: https://www.gsma.com/futurenetworks/wp-content/uploads/2017/11/GSMA-An-Introduction-to-Network-Slicing.pdf.

[8] Faqir Zarrar Yousaf, Michael Bredel, Sibylle Schaller y Fabian Schneider. «NFV and SDN—Key Technology Enablers for 5G Networks». En: *IEEE Journal on Selected Areas in Communications* (sep. de 2017), págs. 2468-2478. ISSN: 0733-8716. DOI: 10.1109/JSAC.2017.2760418.

[9] *Driver ATH9K*. URL: https://wireless.wiki.kernel.org/en/users/drivers/ath9k.

[10] *OpenWrt*. URL: https://openwrt.org/.

[11] Sven Zehl. «Flexible Information Broadcasting using Beacon Stuffing in IEEE 802.11 Networks». Tesis doct. Mayo de 2015. DOI: 10.13140/RG.2.2.19588.94084.

[12] Sven Zehl, Anatolij Zubow y Adam Wolisz. *Enabling Hybrid TDMA/CSMA on IEEE 802.11 Hardware*. 2016. URL: https://arxiv.org/abs/1611.05376.

[13] Johannes Berg. *Framework mac80211*. 2009. URL: https://wireless.wiki.kernel.org/en/developers/Documentation/mac80211.

[14] *NetLink man pages*. URL: http://manpages.ubuntu.com/manpages/xenial/en/man7/netlink.7.html.

[15] Johannes Berg. *API cfg80211*. 2007. URL: http://www.hep.by/gnu/kernel/80211/cfg80211-developers-guide.html.

[16] Toke Høiland-Jørgensen, Michał Kazior, Dave Täht, Per Hurtig y Anna Brunstrom. «Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi». En: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, jul. de 2017, págs. 139-151. ISBN: 978-1-931971-38-6. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/hoilan-jorgesen.

[17] Sven Zehl. *ATH9K HMAC*. Nov. de 2016. URL: https://github.com/szehl/ath9k-hmac.

[18] S. Zehl, A. Zubow y A. Wolisz. «Hotspot slicer: Slicing virtualized home Wi-Fi networks for Air-Time guarantee and traffic isolation». En: https://www2.informatik.huberlin.de/ zubow/zehl17hotspotslicer.pdf. A World of Wireless, Mobile y Multimedia Networks (WoWMoM). IEEE 18th International Symposium, 2017.

[19] Toke Høiland-Jørgensen. *Linux kernel source tree*. ath9k: Introduce airtime fairness scheduling between stations. 2016. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=63fefa050477b0974ab34f65.

[20] Jonathan Corbet, Alessandro Rubini y Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. 3.ª ed. O'Reilly Media. ISBN: 9780596005900. URL: https://lwn.net/Kernel/LDD3/.

[21] *Linux 802.11 Driver Developer's Guide*. URL: https://www.kernel.org/doc/html/latest/driver-api/80211/index.html.

[22] *Unreliable Guide To Hacking The Linux Kernel*. URL: https://www.kernel.org/doc/html/v4.15/kernel-hacking/hacking.html.

[23] *Bootlin - Linux source code*. URL: https://elixir.bootlin.com/linux/v4.15/source.

[24] *The Linux Kernel Module Programming Guide*. URL: https://tldp.org/LDP/lkmpg/2.6/html/.

[25] *iPerf3*. The ultimate speed test tool for TCP, UDP and SCTP. URL: https://iperf.fr/.

[26] *TOS hexadecimal values*. URL: https://www.tucny.com/Home/dscp-tos.

[27] *ATH9K xmit docs*. URL: https://github.com/erikarn/ath9k-docs.

[28] *Linux kernel source tree*. URL: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git.