UNIVERSIDAD DE DEUSTO

VALIDACIÓN DE ARQUITECTURAS DE PROVISIÓN DE SERVICIOS TCP ESCALABLES CON DISPOSITIVOS DE RECURSOS LIMITADOS EN EL PARADIGMA DE LA COMPUTACIÓN EN LA NIEBLA

JUAN JOSÉ ECHEVARRIA MARTÍNEZ

Bilbao, mayo de 2017

UNIVERSIDAD DE DEUSTO

VALIDACIÓN DE ARQUITECTURAS DE PROVISIÓN DE SERVICIOS TCP ESCALABLES CON DISPOSITIVOS DE RECURSOS LIMITADOS EN EL PARADIGMA DE LA COMPUTACIÓN EN LA NIEBLA

Tesis doctoral presentada por
JUAN JOSÉ ECHEVARRIA MARTÍNEZ
dentro del Programa de Doctorado en
INGENIERÍA INFORMÁTICA Y TELECOMUNICACIÓN

Dirigida por el
Dr. JON LEGARDA MACÓN
y el
Dr. PABLO GARAIZAR SAGARMINAGA

Bilbao, mayo de 2017

Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time.

— Thomas Alva Edison
(1847 - 1931)

Agradecimientos

Me gustaría que estas líneas sirvieran para expresar mi más sincero agradecimiento a todas aquellas personas e instituciones que con su ayuda han contribuido al desarrollo de esta tesis.

A decir verdad, acabada la etapa universitaria el doctorado no estaba entre mis planes, pero lo que en un principio iba a ser una breve estancia en DeustoTech cambió por completo esos planes. Allí pude valorar la investigación, conocer su fondo, forma y metodología, y todavía hoy sigo valorando ese aprendizaje continuo que la investigación requiere. Por tanto, en primer lugar quiero dar las gracias a Nekane Sáinz y Juanma López Garde, sin su ayuda no hubiera tenido la opción de entrar en DeustoTech, y a Iñaki Vázquez por darme la oportunidad de formar parte de esta gran familia.

Quisiera hacer extensiva mi gratitud a mis compañeros del grupo de investigación MoreLab del que he formado parte todos estos años, tanto a las personas que actualmente forman parte del grupo, como aquellas con las que he coincidido en algún momento. En particular, me gustaría agradecer a Jonathan Ruiz de Garibay su ayuda durante la fase de experimentación.

Especial reconocimiento merecen Jon Legarda y Pablo Garaizar, mis directores de tesis, por su dedicación y la supervisión de mi trabajo a lo largo de este tiempo. Su dirección ha sido fundamental no sólo en la realización de esta tesis, sino también en mi formación como investigador.

Tampoco puedo olvidar que este trabajo de investigación no podría haberse realizado sin el apoyo financiero proporcionado por el Programa Predoctoral de Formación de Personal Investigador perteneciente al Plan de Ciencia, Tecnología e Innovación del Gobierno Vasco.

Finalmente, quiero agradecer de todo corazón a mis padres su apoyo constante y amor incondicional, además de ser un referente que me ha ayudado a ser quien soy. Por desgracia, mi padre falleció unos meses antes del depósito de la tesis tras años de enfermedad. A modo de homenaje, me gustaría dedicarle esta tesis. De él aprendí el valor del esfuerzo y el gusto por el trabajo bien hecho, y en los momentos más duros de la tesis siempre fue el motor que me ayudó a seguir adelante.

Eskerrik asko, aita.

Resumen

Desde su creación, Internet ha evolucionado hasta convertirse en una herramienta esencial de comunicación y progreso. Actualmente Internet es considerada como una red global diseñada para acceder principalmente a fuentes de información generadas por personas. Uno de los paradigmas dentro de la Internet del futuro –denominado como Internet de las Cosas (IoT)– pretende transformar Internet y sus modelos de interacción. La convergencia de personas, objetos y procesos tendrá un gran impacto a nivel social e industrial.

Desde una perspectiva simplificada, la estructura de IoT se compone de tres elementos conceptuales: dispositivos finales, como sensores o actuadores que a menudo realizan una función dedicada; dispositivos de interconexión, como puertas de enlace o routers que se ubican en los puntos de unión clave de la topología de red para conectar una red local con Internet; y por último, la nube. El concepto de la nube se refiere al uso de servicios bajo demanda a través de Internet, como el almacenamiento y procesamiento de datos en granjas de servidores web.

Para el año 2020 se prevé que el número de dispositivos finales supere los 28.000 millones. Gestionar ese crecimiento de datos supondría una demanda cada vez mayor de ancho de banda, pero también de capacidad de cómputo en la nube. Además, para aplicaciones sensibles a la latencia, el retardo causado por la transferencia de datos a la nube y la espera de la respuesta correspondiente es inaceptable. IoT necesita una forma más eficiente y escalable de trabajar con el volumen de datos generado.

La computación en la niebla (*Fog computing*) es un modelo de computación distribuido propuesto por algunas de las empresas tecnológicas más importantes (ARM, Cisco, Dell, Intel y Microsoft son los miembros fundadores del consorcio OpenFog). Este modelo representa un cambio en la infraestructura de computación en la que los recursos y servicios se distribuyen a lo largo de la topología de red que une las fuentes de datos con la nube, es decir, en los dispositivos de interconexión. Este modelo puede distribuirse aún más, utilizando las capacidades de computación, almacenamiento y comunicación de los dispositivos finales. Trasladar parte de la inteligencia a los dispositivos finales se traduce en una mayor autonomía, tiempos de respuesta predecibles y una reducción de la congestión de red en los dispositivos de interconexión.

Cada uno de estos dispositivos finales requerirá, como mínimo, un procesador para añadir inteligencia al dispositivo, un sensor para capturar alteraciones del entorno, una interfaz de comunicación para la transmisión y recepción de datos, y un componente de memoria. No obstante, la necesidad de reducir consumos y costes en IoT provoca que la mayoría de estos dispositivos tengan restricciones de recursos. El RFC 7228 define que el conjunto de protocolos de Internet se utiliza cada vez más en microcontroladores de recursos limitados, y establece una clasificación en 3 clases atendiendo a

sus restricciones de memoria. En el modelo de la computación en la niebla, incluso estos dispositivos con restricciones podrían prestar un servicio a otros dispositivos locales. Los dispositivos clase 2 son moderadamente limitados, y por tanto son los mejores candidatos para ofrecer esos servicios locales.

El RFC 7547 describe la necesidad de un transporte fiable de mensajes para determinados servicios, y que esa fiabilidad puede conseguirse sobre la base de un protocolo de transporte como TCP. Sin embargo, las limitaciones en computación y memoria de un dispositivo clase 2 hacen que solamente sea posible gestionar un número muy limitado de conexiones TCP al mismo tiempo. Ante una alta demanda –ya sea puntual o deliberada– el servicio podría verse interrumpido. Esta circunstancia ofrece oportunidades para la colaboración de estos dispositivos.

Combinando un número de dispositivos clase 2 en un clúster de balanceo, las limitaciones individuales pueden compensarse parcialmente. Esta arquitectura distribuida y transparente para el cliente debe tener cierta tolerancia a ataques de denegación de servicio (DoS), y ser capaz de distribuir las conexiones TCP entre los dispositivos disponibles.

El objetivo es aplicar en los dispositivos clase 2 las técnicas de balanceo de carga y defensa ante ataques de inundación SYN existentes en los sistemas operativos de propósito general (como Linux), y por tanto, para procesadores con unas características muy superiores a las definidas en el RFC 7228. En definitiva, este trabajo quiere demostrar que es posible trasladar conceptos de la computación en la nube, como la alta disponibilidad y la escalabilidad, al borde más extremo de la red, a los dispositivos de recursos limitados. Al igual que en la nube se configuran varios servidores detrás de un balanceador de carga para procesar el tráfico y hacerlos parecer como una sola unidad, el mismo concepto puede aplicarse a la computación en la niebla.

La propuesta ha sido evaluada mediante el despliegue de un pequeño clúster de dispositivos clase 2, y se ha comparado su rendimiento de red con tres tipos de balanceadores en un experimento real. Los resultados demuestran que un dispositivo clase 2 puede distribuir tráfico TCP a un clúster de hasta cinco servidores clase 2 y veinte clientes concurrentes sin reducir el rendimiento de red respecto a balanceadores de carga de capacidades muy superiores a las definidas en el RFC 7228. Se demuestra por tanto que los dispositivos clase 2 pueden formar por sí mismos arquitecturas de balanceo de carga. Este concepto permitiría crear bajo demanda clústeres de dispositivos limitados con el objetivo de ofrecer servicios TCP escalables y flexibles.

Por otra parte, el método de balanceo de carga seleccionado hace que el tráfico de retorno no pase por el balanceador. Esto significa que deben aplicarse mecanismos de defensa ante ataques DoS en los servidores clase 2. Se han analizado dos mecanismos de mitigación de ataques SYN definidos en el RFC 4987: SYN cookies y reciclar las conexiones semiabiertas más antiguas. Tras analizar las dos implementaciones de SYN cookies disponibles en dos sistemas operativos de tipo Unix, se demuestra que la implementación de Linux es más lenta pero más robusta que la de FreeBSD. Se propone una implementación híbrida que combina el rendimiento de FreeBSD y la robustez de Linux, y se aplica a un dispositivo clase 2. La experimentación demuestra que el rendimiento de red de un dispositivo clase 2 en presencia de un ataque SYN de baja tasa se ve menos afectado con el mecanismo SYN cookies que utilizando una estrategia de reemplazo de conexiones semiabiertas en memoria.

Índice general

Índice de figuras

Índice de tablas

Índice de listados

Acrónimos

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks.
ANOVA	ANalysis Of VAriance.
API	Application Programming Interface.
BSD	Berkeley Software Distribution.
CoAP	Constrained Application Protocol.
CPU	Central Processing Unit.
CSV	Comma-Separated Values.
DDoS	Distributed Denial of Service.
DNS	Domain Name System.
DoS	Denial of Service.
DSR	Direct Server Return.
FIFO	First in, first out.
HTTP	Hypertext Transfer Protocol.
ICMP	Internet Control Message Protocol.
IETF	Internet Engineering Task Force.
IoT	Internet of Things.
IP	Internet Protocol.
MAC	Media Access Control.
MANET	Mobile Ad Hoc Network.
MQTT	Message Queue Telemetry Transport.
NAT	Network Address Translation.
NFC	Near Field Communication.

OSI	Open System Interconnection.
RAM	Random Access Memory.
RFC	Request for Comments.
RFID	Radio Frequency IDentification.
ROM	Read-Only Memory.
RTOS	Real-Time Operating System.
SMTP	Simple Mail Transfer Protocol.
SSH	Secure SHell.
TCB	Transmission Control Block.
TCP	Transmission Control Protocol.
UDP	User Datagram Protocol.
URL	Uniform Resource Locator.
W3C	World Wide Web Consortium.
WoT	Web of Things.

CAPÍTULO

1

Introducción

Internet ha evolucionado hasta convertirse en una herramienta esencial de comunicación y progreso [Rainie y Wellman, 2012]. Desde sus orígenes y hasta hace relativamente poco tiempo, Internet se entendía como una red de telecomunicaciones diseñada para conectar ordenadores. De esta forma, Internet se ha convertido en el medio preferente para el intercambio de información entre personas. El Internet de las cosas (IoT) representa un nuevo capítulo en la historia de Internet [Feki et al., 2013]. Este paradigma extiende la conectividad a una amplia gama de objetos y procesos que generan, procesan y consumen datos sin intervención humana [Al-Fuqaha et al., 2015].

En 1999, Ashton[1] se encargó de preparar el terreno para lo que más adelante se conocería como IoT. En primer lugar, señaló que prácticamente toda la información disponible en Internet había sido creada por y para personas. Como resultado, había agujeros de información en procesos de ámbito industrial. La solución que propuso fue realizar un seguimiento de los productos con sensores conectados a Internet. De esta forma, se podría combinar información en tiempo real de múltiples fuentes y anticipar decisiones. Ashton se convirtió en uno de los pioneros que concibieron la fusión entre el mundo físico (objetos) y el mundo digital (Internet). Sin embargo, la tecnológica disponible en aquel momento no permitía implementarlo.

Hoy en día, esa visión se ha materializado gracias a tres leyes específicas que el progreso tecnológico parece respetar. La ley de Moore, descrita en 1965 por uno de los fundadores de Intel –Gordon Moore– especifica que el número de transistores en un circuito integrado se duplica cada 18 meses. Como resultado, se obtiene más potencia de cálculo en el mismo espacio, reduciendo de esta forma el precio por unidad de cálculo.

La segunda ley –postulada por Jonathan Koomey– establece que cada 18 meses la energía requerida para la misma cantidad de cálculo se reduce a la mitad. Esta tendencia fue documentada por Koomey en el año 2010 [Koomey et al., 2011] tras analizar el número de cálculos por julio de energía disipada y llegar a la conclusión de que se duplicaban cada año y medio de manera estable desde la década de 1950. Las leyes de Moore y Koomey determinan que la incorporación de la computación sea económicamente

[1] http://www.rfidjournal.com/articles/view?4986

viable y técnicamente factible en cualquier objeto. Los procesadores se hacen más pequeños y más baratos, y su consumo de energía disminuye.

La tercera ley fue formulada en los años 80 por uno de los inventores de Ethernet –Bob Metcalfe– y establece que el valor de una red de telecomunicaciones es proporcional al cuadrado del número de usuarios conectados. Esto significa que cuantos más dispositivos haya, más valiosa se convierte una red. La ley de Metcalfe ha demostrado ser cierta para las redes de ordenadores durante las últimas décadas, por lo que es de esperar que siga siendo cierta para IoT. El Internet de la década de 1990 conectó unos mil millones de personas a través de ordenadores. El Internet de la década de 2000 conectó dos mil millones de personas a través de los teléfonos inteligentes. Actualmente, está en camino de llegar a los seis mil millones. No obstante, ya hay más objetos conectados a Internet que personas, y esta tendencia no muestra señales de detenerse hasta alcanzar los 28 mil millones de objetos conectados para el año 2020[2].

El rápido crecimiento que está experimentando IoT se traduce en la generación de un gran volumen de datos no estructurados. Determinar cómo gestionar esos datos para que se conviertan en información útil es crucial. Todos los datos recogidos por estos dispositivos carecen de valor si no hay una infraestructura para analizarlos y tomar las decisiones correspondientes. En definitiva, el valor de IoT proviene de la captura de datos, su análisis, y la capacidad de tomar decisiones rápidamente.

Desde una perspectiva simplificada, la estructura de IoT se compone de tres elementos conceptuales: i) dispositivos finales, como sensores o actuadores que dependiendo de la aplicación generan y/o consumen datos; ii) dispositivos de interconexión, como puertas de enlace o routers que se ubican en los puntos de unión clave de la topología de red para conectar una red local con Internet; y por último, iii) la nube. El concepto de la nube se refiere al uso bajo demanda de un conjunto de recursos de computación a través de Internet, como el almacenamiento y procesamiento de datos en granjas de servidores web. El término nube se inspiró en el símbolo de la nube que se utiliza a menudo para representar a Internet en los diagramas de red.

El enfoque predominante en la actualidad es enviar los datos generados por los dispositivos finales a servicios de computación en la nube [Botta et al., 2016]. Un ejemplo podría ser un sistema inteligente de control de partículas en el aire. El polvo, polen u otras partículas que pueden provocar alergias estacionales se monitorizan por medio de sensores inalámbricos y los datos se envían a Internet. El servicio correspondiente en la nube interpreta los datos procedentes de todos estos sensores. Una vez procesada, la información es accesible a los usuarios a través de sus dispositivos móviles. De esta forma pueden evitar zonas de la ciudad donde se acumulen más partículas por millón.

Los impulsores de este modelo de almacenamiento y computación remota defienden que la mayor parte de la computación residirá en la nube en un futuro. Sin embargo, la realidad es bien distinta[3]. Mover los datos a la nube es más complejo de lo que muchas veces se quiere admitir, y el principal problema es el ancho de banda [Olanrewaju et al., 2014]. Si el objetivo es simplemente ahorrar el coste de almacenar datos de manera local, la nube es útil, siempre y cuando la transferencia sea a través de una

[2] https://hbr.org/2014/10/the-sectors-where-the-internet-of-things-really-matters
[3] http://www.wsj.com/articles/SB10001424052702304908304579566662320279406

conexión de alta velocidad. El paradigma de la computación en la nube presupone que existe suficiente ancho de banda para enviar los datos desde el dispositivo final al servicio remoto. En el ámbito de IoT esto no es así. El ancho de banda disponible es mucho menor, y por tanto, las redes actuales simplemente no son lo suficientemente rápidas como para transmitir datos desde los dispositivos finales a la nube al mismo ritmo que se generan.

Incluso en el supuesto de que el ancho de banda no fuera un problema, el modelo de computación en la nube seguiría sin ser viable para muchas aplicaciones IoT debido a la latencia. La computación en la nube reside en los centros de datos remotos, y por tanto el tiempo necesario para enviar los datos, analizarlos, tomar decisiones, y, finalmente, recibir los comandos para actuar puede ser incompatible con una aplicación en tiempo real [Sarkar et al., 2015]. Para cuando los datos generados por los dispositivos finales llegan a la nube, pueden haberse generado nuevos datos y la oportunidad de actuar sobre ellos puede haber desaparecido. Un ejemplo común para referirse a esta limitación es el siguiente: "el paradigma de la nube es como tener al cerebro controlando las extremidades a kilómetros de distancia, no ayudará si se necesitan reflejos rápidos". En pocas palabras, si las grandes predicciones sobre IoT resultan ser ciertas, incluso las arquitecturas en la nube más avanzadas y distribuidas no serán capaces de soportar las necesidades de computación y comunicación de una amplia gama de aplicaciones IoT. Se necesita por tanto una forma más eficiente y escalable de trabajar con el volumen de datos generado.

Varias de las empresas tecnológicas más importantes (ARM, Cisco, Dell, Intel y Microsoft son los miembros fundadores del consorcio OpenFog) proponen un enfoque alternativo: trasladar la computación y los servicios a un lugar más cercano a donde se generan los datos [Shi et al., 2016]. Esta propuesta es una vieja práctica en el entorno industrial, y en la actualidad tiene un gran valor teniendo en cuenta el creciente número de dispositivos finales.

En 2014, Cisco realizó una presentación en la LinuxCon titulada "*From Cloud to Fog & The Internet of Things*", con el objetivo de introducir su enfoque de procesamiento para IoT. El nombre de computación en la niebla (*Fog computing*) es una alusión a lo que ocurre con las nubes a medida que se acercan al suelo.

Cisco presentó este modelo distribuido porque consideró evidente que los miles de millones de dispositivos IoT que se desplegarán en un futuro cercano no deberían comunicarse con servicios remotos, sino locales[4]. La computación en la niebla es un modelo en el que los recursos y servicios están alojados en el borde de la red, en los dispositivos de interconexión [Stojmenovic y Wen, 2014]. Este modelo de comunicación imita las capacidades de la nube pero los datos son procesados de forma distribuida en los puntos de unión de la topología de red. Al hacerlo, la computación en la niebla soporta aplicaciones emergentes de IoT que demandan tiempos de respuesta cortos y predecibles [Zhang et al., 2015].

Dentro del concepto de la computación en la niebla, existe una versión que permite distribuir aún más los recursos y servicios, y se llama *mist computing*[5], que se podría traducir como computación en la neblina. Este modelo lleva los conceptos de la computación en la niebla un paso más allá, directamente al borde extremo de la arquitectura de red, utilizando las capacidades de computación, almacenamiento

[4] https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf
[5] http://embedded-computing.com/guest-blogs/fluid-computing-unifying-cloud-fog-and-mist-computing/

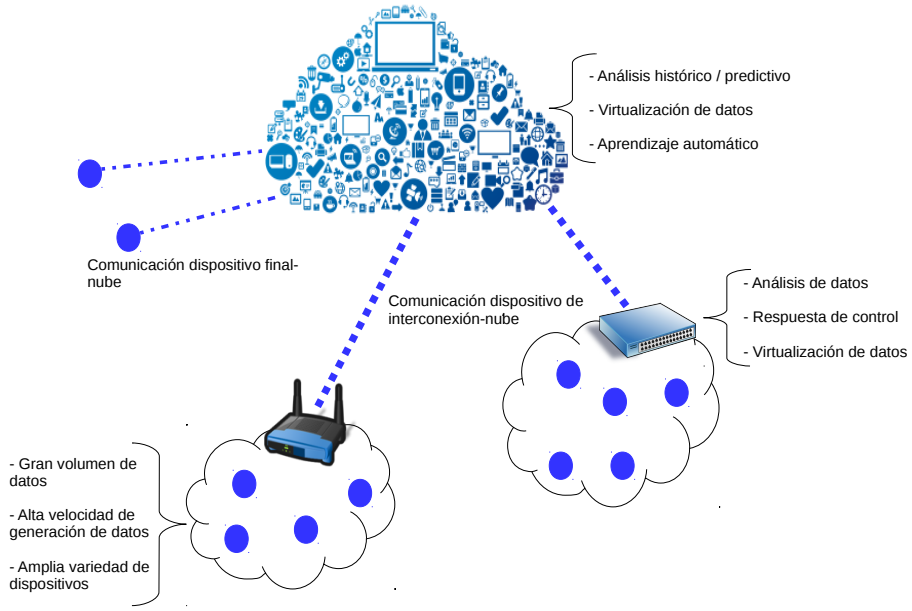


Figura 1.1 Arquitectura de computación en la niebla

y comunicación de los dispositivos finales [Preden et al., 2016]. Sistemas de iluminación inteligente en las ciudades serían un ejemplo de este modelo de computación[6].

Trasladar parte de la inteligencia a los dispositivos finales se traduce en una mayor autonomía, tiempos de respuesta menores y una reducción de la congestión de red en los dispositivos de interconexión y la nube.

Cada uno de estos dispositivos finales requerirá, como mínimo, un procesador para añadir inteligencia al dispositivo, un sensor para capturar alteraciones del entorno, una interfaz de comunicación para la transmisión y recepción de datos, y un componente de memoria. Como es de imaginar, estos dispositivos son heterogéneos en cuanto a plataformas, recursos y conectividad. No obstante, la necesidad de reducir consumos y costes en IoT provoca que muchos de estos dispositivos tengan restricciones de recursos [Martinez et al., 2015].

El RFC 7228 "Terminology for Constrained-Node Networks"[7] describe precisamente que el conjunto de protocolos de Internet se utiliza cada vez más en microcontroladores con limitaciones en memoria y recursos de computación, creando de esta manera las denominadas redes de dispositivos limitados. Este RFC define tres clases de dispositivos teniendo en cuenta combinaciones de restricciones en el tamaño de las memorias ROM y RAM.

[6] http://www.cityntel.com/
[7] http://www.rfc-editor.org/rfc/rfc7228.txt

- Los dispositivos de clase 0 (C0) son dispositivos muy limitados que suelen actuar como sensores y tienen tamaños de memoria RAM/ROM por debajo de 10/100 KB. Estos dispositivos necesitan un dispositivo más potente que actúe como puerta de enlace para comunicarse con Internet. Un ejemplo de dispositivo clase 0 es WISP [Sample et al., 2008], una etiqueta RFID pasiva con sensores.

- Los dispositivos de clase 1 (C1) son menos limitados que los de clase 0, con aproximadamente 10/100 KB de memoria RAM/ROM, y no requieren de la ayuda de otros dispositivos para comunicarse con Internet. Implementan una pila IP mínima y por lo general utilizan protocolos diseñados específicamente para dispositivos limitados, como CoAP. La mayoría de sensores inalámbricos entran en esta categoría, por ejemplo la mota TelosB.

- Los dispositivos de clase 2 (C2) son moderadamente limitados, con alrededor de 50/250 KB de memoria RAM/ROM. Estos dispositivos pueden conectarse directamente a Internet, ya que pueden implementar prácticamente la misma pila de protocolos IP que se utiliza en dispositivos más potentes. No obstante, incluso estos dispositivos pueden beneficiarse de un uso más eficiente de sus recursos. La mayoría de microcontroladores de 32 bits entran dentro de esta categoría.

El paradigma IoT pretende expandir la infraestructura de Internet a cualquier dispositivo con computación, almacenamiento y conectividad, y los dispositivos con restricciones no son una excepción[8]. Entre ellos, los dispositivos C2 son menos limitados y por tanto mejores candidatos para ofrecer servicios locales dentro del modelo *mist computing*.

En conclusión, la convergencia de los tres modelos de computación será necesaria para acelerar la adopción de IoT. Los dispositivos finales están más cerca de la acción, pero por el momento no tienen los recursos de computación y memoria necesarios para realizar tareas complejas, como el análisis de grandes volúmenes de datos o el aprendizaje automático. La nube, por otro lado, sí dispone de la potencia necesaria, pero está demasiado lejos para procesar los datos y responder a tiempo. De esta forma, las tareas que requieran una computación más intensiva pueden ser procesadas en los dispositivos de interconexión (*Fog*), mientras que las tareas menos intensivas se pueden procesar en los dispositivos finales (*Mist*). Por último, dado que en IoT los datos locales solo se guardan durante un breve período, los datos que necesiten un almacenamiento o procesamiento a largo plazo pueden enviarse a la nube.

1.1 Motivación

La aplicación de los protocolos de Internet para dispositivos con limitaciones a menudo requiere guías adicionales para la implementación de técnicas eficientes y otras consideraciones. El grupo de trabajo de la IETF denominado "*Guía de implementación ligera (LWIG WG)*" está trabajando en estas especificaciones. Entre ellas, el RFC 7228 define la terminología común de las redes de dispositivos limitados sobre el protocolo IP.

[8] http://www.ti.com/lit/wp/sszy005/sszy005.pdf

Por otra parte, el RFC 7547 proporciona una descripción de los requisitos funcionales para la gestión de redes donde están involucrados estos dispositivos limitados. Entre los requisitos funcionales con una alta prioridad, en concreto el requisito 10.002 que tiene por título "Transporte confiable de mensajes unicast", se especifica que diversas aplicaciones necesitan un transporte fiable de mensajes, y que esa fiabilidad puede conseguirse sobre la base de un protocolo de transporte orientado a conexión como TCP.

Debido a las limitaciones en computación y memoria de un dispositivo C2, solamente es posible gestionar un número muy limitado de conexiones TCP al mismo tiempo. La falta de disponibilidad de un servicio se refleja en la incapacidad de establecer nuevas conexiones. Este tipo de interrupciones en el servicio se producen por una sobrecarga de los componentes individuales de la infraestructura de red, como los servidores. Si esta condición es causada intencionalmente por agentes externos, se habla de un ataque de denegación de servicio (DoS).

Los ataques DoS han afectado a las redes de telecomunicación desde hace mucho tiempo [Garber, 2000]. Si la historia se repite, una gran variedad de dispositivos IoT serán expuestos a estos ataques [Wu et al., 2016], pero en una escala mucho más grande[9], y generalmente con una protección limitada o nula [Ravi et al., 2004].

Los ataques DoS se producen de diferentes formas, y por lo tanto una protección efectiva pasa por implementar defensas para cada capa del modelo OSI, que define el proceso de transmisión de la información entre dispositivos. El ataque más común –y sencillo– en la capa de transporte (capa 4) se denomina inundación SYN, y su característica más relevante respecto a otros ataques TCP es que el atacante no tiene por qué utilizar su IP real. El atacante dirige al servidor más solicitudes de las que éste puede procesar, y por tanto llena la cola de conexiones con solicitudes falsas. La reducida memoria disponible para la cola de conexiones en un dispositivo C2 hace que incluso inundaciones SYN de baja tasa resulten efectivas.

Esta limitación de memoria hace que también sea posible una denegación de servicio provocada por tráfico legítimo. Los servidores más pequeños –como los C2– solo están preparados para gestionar un tráfico muy limitado, por lo que un aumento repentino en el tráfico del servidor puede tener el mismo efecto que un ataque DoS.

Conseguir una alta disponibilidad en dispositivos C2 es prioritario. Los nuevos modelos de computación y comunicación promueven una mayor interacción entre dispositivos[10] [Kum et al., 2016]. Además, debido a sus ámbitos de aplicación, estos dispositivos tienen unas expectativas de fiabilidad mucho mayores que otros sistemas (algunos de estos dispositivos no se pueden detener o retrasar ya que realizan tareas de monitorización industrial, médica o energética, entre otras actividades críticas). Sin embargo, sus limitaciones son un hándicap para conseguir esa disponibilidad.

Un principio general en el diseño de un sistema de alta disponibilidad es utilizar un mayor número de componentes hardware. Estos componentes pueden utilizarse para conseguir redundancia (el componente redundante se utiliza en caso de que falle el primario), o una mayor capacidad (los componentes hardware

[9] http://tecnologia.elpais.com/tecnologia/2016/10/22/actualidad/1477140616_601734.html
[10] http://www.gartner.com/newsroom/id/3143521

trabajan juntos). En este último caso, un balanceador de carga se encargaría de distribuir la carga de trabajo entre los componentes.

En general, un servicio web en la nube no depende de un único servidor. En estos casos se utiliza la regla divide y vencerás, y por tanto se utilizan arquitecturas que distribuyen las conexiones entre múltiples servidores de una manera transparente para el cliente. A esta arquitectura se le denomina clúster, y debido a su naturaleza distribuida mejora el rendimiento y disponibilidad respecto a un solo servidor [Thiruvathukal, 2005]. Si los dispositivos que forman el clúster tienen todos la misma configuración hardware y software se habla de un clúster homogéneo, de lo contrario se trata de un clúster heterogéneo.

Aunque más complejo de implementar, este concepto de la unión hace la fuerza debe ser considerado en dispositivos de recursos limitados. La combinación de un número de dispositivos C2 en un clúster puede compensar parcialmente sus limitaciones individuales. Esta arquitectura distribuida y transparente para el cliente debe ser capaz de distribuir las conexiones entre los dispositivos configurados en el clúster y tener cierta tolerancia a ataques DoS. Las implementaciones actuales de balanceadores y mecanismos de defensa están basadas en sistemas operativos de propósito general, y por tanto ejecutadas en dispositivos con unas características muy superiores a las descritas en el RFC 7228. Implementar estas funcionalidades en clústeres de dispositivos limitados C2 y evaluar su rendimiento respecto a balanceadores de carga de recursos no limitados determinará si es posible trasladar conceptos de la computación en la nube, como la alta disponibilidad y la escalabilidad, al borde más extremo de la red.

Si los resultados son positivos, se podría pensar en diseñar arquitecturas de red locales que apliquen el concepto de elasticidad utilizado en la computación en la nube [Herbst et al., 2013], es decir, “adaptarse a los cambios de la carga de trabajo mediante la reserva y liberación de recursos de manera autónoma, de modo que en cada momento los recursos disponibles se ajusten a la demanda actual lo mejor posible”. Esa elasticidad en la asignación de los recursos puede ser en términos de potencia de procesamiento, almacenamiento, ancho de banda, etc. De esta forma, los dispositivos C2 podrían organizarse de forma autónoma [Kalwar, 2010] en clústeres temporales para atender una alta demanda de un servicio TCP o realizar tareas distribuidas con otros dispositivos locales [Preden et al., 2015].

1.2 Hipótesis y objetivos

A continuación se define la hipótesis de investigación, que puede formularse de la siguiente manera:

Hipótesis: Un clúster homogéneo de dispositivos de recursos limitados puede distribuir y procesar conexiones TCP concurrentes sin reducir su rendimiento de red respecto a balanceadores de carga de recursos no limitados.

1.2.1 Objetivos

La presente tesis doctoral tiene un objetivo principal y varios objetivos específicos u operativos. El objetivo principal del trabajo de investigación es confirmar o refutar la hipótesis planteada. Para ello,

será necesario desplegar un clúster homogéneo de dispositivos limitados, C2 según el RFC 7228, para formar una arquitectura distribuida que ofrezca un servicio TCP y analizar su rendimiento de red ante conexiones concurrentes, en comparación con balanceadores de carga de mayor capacidad. Además, debido a la exposición de cualquier red a ataques de denegación de servicio, se quiere determinar qué técnica de mitigación penaliza menos el rendimiento de red ante ataques SYN de baja tasa y cuando la cola de conexiones está llena.

A continuación se definen una serie de objetivos específicos:

- **Objetivo 1:** Analizar y seleccionar el método de balanceo de carga más apropiado para dispositivos C2.
- **Objetivo 2:** Analizar y seleccionar el dispositivo C2 y la implementación de la pila de protocolos TCP/IP que cumpla lo establecido en el RFC 7228. Este dispositivo C2 se utilizará tanto para el balanceador como para los servidores (clúster homogéneo).
- **Objetivo 3:** Diseñar e implementar un balanceador de carga en un dispositivo C2.
- **Objetivo 4:** Implementar un balanceador de carga en dos dispositivos con capacidades muy superiores a las definidas en el RFC 7228.
- **Objetivo 5:** Diseñar e implementar los servidores C2 que forman el clúster. Cada servidor aceptará hasta cuatro conexiones concurrentes, y el clúster lo formarán hasta cinco servidores.
- **Objetivo 6:** Definir un conjunto de experimentos que combinen un número de clientes y servidores, y comparar el rendimiento de red del clúster usando los tres balanceadores de carga. El rendimiento de red (o *throughput* en inglés) se define como el número de bytes transmitidos de manera correcta por unidad de tiempo.
- **Objetivo 7:** Analizar las técnicas comunes de mitigación aplicables cuando la cola de conexiones se llena de solicitudes falsas: SYN cookies y reciclar conexiones semiabiertas según el RFC 4987.
- **Objetivo 8:** Analizar los algoritmos de generación y validación de SYN cookies de los sistemas operativos tipo Unix que las implementan.
- **Objetivo 9:** Adaptar e implementar el algoritmo de generación y validación de SYN cookies, y aplicarlo a un dispositivo C2.
- **Objetivo 10:** Medir en un dispositivo C2 el rendimiento de red de las dos técnicas de mitigación ante un ataque SYN de baja tasa.

1.3 Antecedentes de la investigación

En esta sección se presenta el origen de este trabajo de investigación. Hace unos años se observó que los modelos de interacción de etiquetas RFID con sensores estaban principalmente basados en protocolos e

interfaces de usuario ad hoc. En la actualidad, se puede encontrar esta afirmación recogida en el RFC 7547 acerca de sensores IP que utilizan protocolos ad hoc de capa de aplicación.

Para que IoT se convierta en una realidad, se necesita un protocolo de capa de aplicación universal. Es decir, la capa física de comunicación podría ser cualquier opción que soporte la creación de una red IP, pero desde la capa de red a la capa de aplicación se debe optar por un modelo común.

En lugar de crear un nuevo protocolo de aplicación o utilizar protocolos que han ido apareciendo en los últimos años, ¿por qué no simplemente reutilizar el protocolo que ya se emplea para crear aplicaciones interactivas como la propia web?. El problema no es que los protocolos de aplicación como CoAP o MQTT no estén bien diseñados o incluso sean más eficientes para dispositivos IoT [Thangavel et al., 2014], simplemente no son los protocolos que el resto de Internet utiliza para comunicarse. Como ilustración, mientras que todos los navegadores soportan HTTP, no hay ningún navegador ni servidor web que soporte CoAP.

Reutilizar y aprovechar los protocolos y estándares web para que los datos y los servicios ofrecidos por los dispositivos IoT sean más accesibles es el paradigma que defiende la Web de las Cosas [Heuer et al., 2015]. En la página del W3C se especifica que "los servidores de la Web de las Cosas estarán disponibles para microcontroladores, teléfonos inteligentes, concentradores en casas y granjas de servidores en la nube. Los servidores más potentes soportarán una variedad de lenguajes, mientras que los servidores más pequeños podrían utilizar comportamientos precompilados"[11].

De esta forma, se determinó que el protocolo HTTP, aunque no era perfecto[12], permitía a los sensores RFID tener un acceso genérico y formar parte del resto de Internet. Éste era el objetivo principal del proyecto Webtag [Echevarria et al., 2011, 2012], utilizar un servidor HTTP sencillo y una comunicación de campo cercano que aprovechase el hardware existente en los teléfonos inteligentes actuales. De esta forma, es posible acceder a la información de los sensores utilizando el propio navegador web del teléfono, sin necesidad de una interfaz de usuario ad hoc [Duquennoy et al., 2009].

Como prueba de concepto se diseñó una etiqueta del tamaño de una tarjeta de crédito y se utilizó como banco de pruebas para evaluar el rendimiento de los protocolos de Internet y los modelos de navegación web en un dispositivo de clase 0. La primera versión se basó en una interfaz NFC, una interfaz de comunicación que no estaba en principio diseñada para soportar IP. Por lo tanto, se desarrolló un driver que hacía que una comunicación NFC cumpliese los requisitos de TCP/IP. Es el primer trabajo que consigue una integración total de NFC con TCP/IP.

El estado del arte actual ha demostrado que el trabajo realizado iba por el buen camino. Recientemente Google ha empezado a desarrollar un nuevo proyecto denominado *Physical Web*, patrocinado por la división Chrome. Su planteamiento es sencillo: la denominada revolución de las aplicaciones aventuraba que habría una aplicación móvil para cada dispositivo IoT. Sin embargo, esta dependencia en aplicaciones ad hoc no es escalable. Google busca que cada dispositivo IoT que requiera una interacción

[11] https://www.w3.org/blog/2015/05/building-the-web-of-things/

[12] El grupo de trabajo HTTPbis ha finalizado recientemente la especificación para el protocolo HTTP/2, que es más adecuado para escenarios IoT que las versiones anteriores de HTTP gracias a un formato más compacto y a reglas de procesamiento simplificadas.

Figura 1.2 WebTag

con el usuario tenga una dirección web, a través de la cual se pueda acceder a su información. De esta forma Google propone utilizar direcciones URL y el protocolo HTTP, y no aplicaciones móviles (o app en inglés), como el futuro de la interacción bajo demanda en IoT[13].

Por otra parte, el grupo activo de trabajo de la IETF *IPv6 over Networks of Resource-constrained Nodes (6lo)*, cuyos esfuerzos se centran en facilitar la conectividad IPv6 en las redes de dispositivos limitados definidos en el RFC 7228, ha propuesto el estándar "Transmission of IPv6 Packets over Near Field Communication (draft-ietf-6lo-nfc-05)" que describe cómo transmitir paquetes IPv6 a través de NFC usando técnicas 6LoWPAN[14].

En definitiva, el trabajo desarrollado en Webtag demostró antes que ningún otro que los dispositivos más limitados y con conectividad inalámbrica de corto alcance eran capaces de soportar los protocolos estándar de Internet y sus requisitos básicos de servicio, y por tanto formaban parte intrínseca de IoT. La continuación de este trabajo pasa por evaluar la organización de estos dispositivos de recursos limitados en clústeres para ofrecer servicios distribuidos y escalables.

1.4 Metodología de la investigación

Para cumplir los objetivos del trabajo de investigación propuesto, se ha establecido una estrategia de investigación que se presenta a continuación, y en la que se irán explicando las diferentes fases que la componen. La figura 1.3 representa cada una de estas fases, junto con sus subfases, usando un diagrama de flujo.

Una vez identificado el problema de investigación, se analiza el estado del arte en ese ámbito. Dentro de esta revisión no solo se valoran los trabajos publicados en conferencias y revistas académicas, sino también las prácticas usadas en la actualidad en los principales sistemas operativos para configurar clústeres y esquemas de balanceo de carga, así como los mecanismos para defenderse –a nivel local– de ataques de denegación de servicio.

[13] https://google.github.io/physical-web/
[14] https://tools.ietf.org/html/draft-ietf-6lo-nfc-05

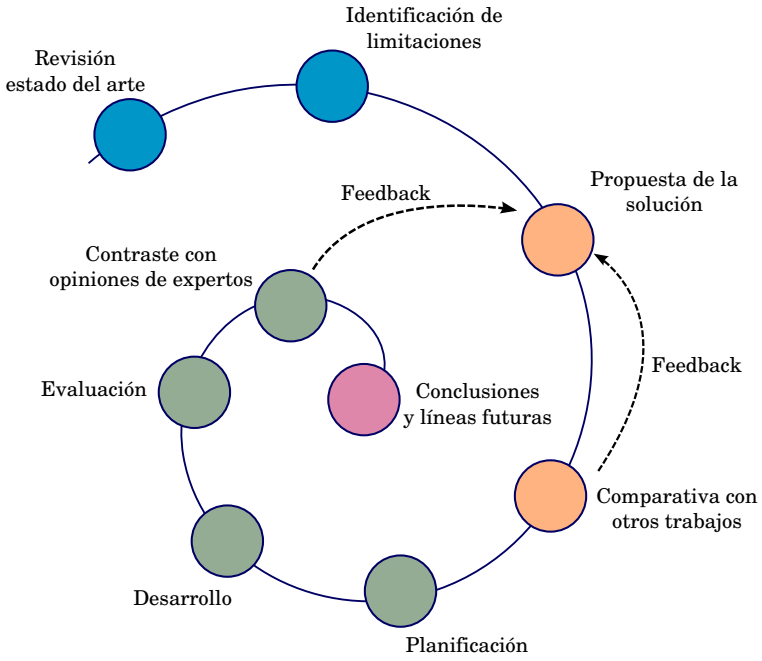


Figura 1.3 Metodología de la investigación

Una vez analizado el estado del arte, se identifica una línea de trabajo que permita continuar con la investigación. Una vez hecha la propuesta, se debe comparar con otros trabajos –si existiesen– con el objetivo de redefinir la propuesta inicial si fuera necesario.

A continuación se procede a la planificación del trabajo de investigación restante, su alcance, y la definición de sus tareas y plazos. En la subfase de desarrollo se diseña e implementa la solución propuesta. La evaluación se encarga de diseñar y ejecutar los experimentos que permitan medir el funcionamiento de la solución, y en definitiva obtener unos resultados que determinen si la solución es válida.

Los resultados de la evaluación, junto con las sugerencias de expertos en el tema de investigación, determinarán cambios en la propuesta inicial. Esas sugerencias se pueden obtener de muchas fuentes, como documentos técnicos o foros, e incluso las críticas constructivas de artículos rechazados pueden ser valiosas. Esta fase será iterativa hasta conseguir unos resultados aceptables que validen la hipótesis de investigación planteada.

La última fase de la metodología presenta las conclusiones y las líneas futuras del trabajo.

1.5 Organización de los capítulos restantes

A continuación se desglosa la organización y el contenido de los capítulos restantes.

El capítulo 2 está dividido en dos secciones. La primera presenta una breve introducción a las características y fases de las comunicaciones TCP con objeto de conocer la secuencia de la conexión y las vulnerabilidades de la fase de establecimiento. A continuación se introduce el concepto de balanceo

de carga, su arquitectura básica, y los métodos de balanceo más comunes. Los algoritmos de balanceo de carga no se cubren en el capítulo. Por último, se analiza un ataque contra la fase de establecimiento TCP, incluyendo una revisión bibliográfica de los mecanismos de mitigación.

En el capítulo 3 se explican las dos propuestas que permitirán mejorar el servicio de dispositivos C2. La primera propuesta se basa en implementar un clúster de dispositivos C2 que permita aumentar la cantidad de conexiones concurrentes al servicio proporcionado. Para ello se escoge el mecanismo de balanceo más apropiado para este tipo de dispositivos y se implementan sus funciones tanto en los servidores como en el balanceador. La segunda parte del capítulo presenta un análisis de seguridad y rendimiento de dos implementaciones de SYN cookies. A continuación se diseña un algoritmo basado en estas dos implementaciones. La sección se cierra con unos experimentos que permiten determinar qué implementación se comporta mejor en un dispositivos C2. Esta implementación se utilizará en los experimentos definidos en el capítulo 4.

El capítulo 4 presenta la configuración y los resultados de los experimentos realizados. En la primera sección se presentan las diferentes configuraciones del clúster y la metodología utilizada. Los experimentos se realizan para cada balanceador de carga disponible. La segunda sección mide el rendimiento de red de un dispositivo C2 ante ataques de inundación SYN de baja tasa. Los mecanismos de mitigación utilizados serán el reciclado de conexiones semiabiertas y las SYN cookies. Los resultados de ambos experimentos se analizan y se extraen las conclusiones correspondientes.

Por último, el capítulo 5 recoge las conclusiones extraídas de la investigación y presenta algunas líneas de investigación que permitan continuar el trabajo desarrollado.

CAPÍTULO

2

Fundamentos y estado del arte

Al diseñar una arquitectura de red para dar soporte a un servicio, la disponibilidad es un requisito fundamental. Los beneficios de IoT no se pueden disfrutar si los servicios o recursos no están disponibles cuando se necesitan. No obstante, este requisito a menudo se pasa por alto durante la fase de diseño de redes con dispositivos limitados [Ravi et al., 2004], dejando así a muchos dispositivos vulnerables a picos puntuales de tráfico, ya sean legítimos o no.

La disponibilidad de un servicio puede conseguirse a través de la escalabilidad y la seguridad, requisitos necesarios para el diseño e implementación de arquitecturas que aseguran un cierto grado de continuidad operacional [Hassan et al., 2014].

La capacidad de superar los límites de rendimiento mediante la adición de recursos se define como escalabilidad. Existen herramientas que ayudan a conseguir esta escalabilidad, siendo una de las más importantes los balanceadores de carga. A mayor número de recursos, mayor capacidad tendrá el servicio [Limoncelli, 2017]. Este capítulo introducirá el concepto de balanceo de carga, se explicarán los métodos más comunes que se utilizan en la actualidad y se analizará su viabilidad para el problema de investigación planteado.

Garantizar la disponibilidad implica también la prevención o mitigación de ataques DoS. Estos ataques se han convertido en una de las principales amenazas para los servicios de Internet. El objetivo de un ataque DoS es interrumpir un servicio agotando los recursos disponibles o explotando alguna vulnerabilidad. De esta forma se impide el acceso al servicio a usuarios legítimos. En el capítulo se presenta un ataque DoS específico –conocido como inundación SYN– y se analizan las soluciones existentes para su mitigación.

Debido a las limitaciones en computación y memoria de un dispositivo C2, la revisión del estado del arte tendrá en cuenta el uso de estos recursos.

No obstante, antes de conocer en detalle cómo funciona el balanceo de carga y el ataque SYN, es necesario conocer los fundamentos básicos del protocolo TCP. En la primera sección se explican brevemente esos fundamentos. Si se desea más información, se puede obtener a través de varias fuentes, incluyendo libros [Stevens, 1993] y los RFC que describen formalmente su funcionamiento.

2.1 Fundamentos de TCP

En primer lugar se analizarán las características fundamentales del protocolo TCP, actualmente documentado por el IETF en el RFC 793.

TCP es un protocolo de transporte orientado a conexión perteneciente a la pila de protocolos IP, y su uso está muy extendido en Internet. Las aplicaciones de red más populares –como la navegación web– utilizan TCP en sus comunicaciones.

La función principal del nivel de transporte dentro de la arquitectura de protocolos TCP/IP es la de permitir la comunicación extremo a extremo entre dos aplicaciones de forma ordenada y sin pérdida. Para ello TCP se encarga de la retransmisión de los paquetes perdidos, de reorganizar los paquetes fuera de orden y de controlar la congestión de red entre otros.

2.1.1 Formato de la cabecera

Un paquete es la unidad básica de transporte de datos a través de una red. Está compuesto de tres elementos: i) una cabecera que contiene la información necesaria para transportar el paquete desde el emisor hasta el receptor, ii) una carga útil con los datos que se desean transferir, y iii) una cola que contiene el código de detección de errores. Cuando el paquete se transfiere utilizando la pila TCP/IP, los protocolos de cada capa agregan campos a esa cabecera.

A continuación se explica de forma resumida la funcionalidad de los campos TCP:

Tabla 2.1 Cabecera TCP

<table>
<tr><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td><td colspan="8">4</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">Puerto de origen</td><td colspan="16">Puerto de destino</td></tr>
<tr><td colspan="32">Número de secuencia</td></tr>
<tr><td colspan="32">Número de reconocimiento</td></tr>
<tr><td colspan="4">Longitud de cabecera</td><td colspan="3">Reservado</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Tamaño de ventana</td></tr>
<tr><td colspan="16">Suma de verificación</td><td colspan="16">Puntero urgente (si URG activo)</td></tr>
<tr><td colspan="32">Opciones (si Longitud de cabecera>5, relleno con "0")</td></tr>
</table>

- Puerto de origen (16 bits): identifica el puerto emisor.
- Puerto de destino (16 bits): identifica el puerto receptor. Estos dos valores identifican la aplicación receptora y la emisora, y junto con las direcciones IP del emisor y receptor identifican de forma unívoca una conexión.
- Número de secuencia (32 bits): Si el flag SYN está activo, este número es el número de secuencia inicial. Si no está activo, es el número de secuencia acumulado del primer byte de datos de la conexión actual.
- Número de reconocimiento (32 bits): contiene el valor del siguiente número de secuencia que el emisor espera recibir. Una vez que la conexión ha sido establecida, este número se envía siempre y se valida con el flag ACK activado.

- Longitud de cabecera (4 bits): especifica el tamaño de la cabecera. Es necesario porque la longitud del campo opciones es variable.
- Reservado (3 bits): para uso futuro.
- Flags (9 bits):
  - NS (1 bit): sirve como protección frente a paquetes maliciosos que se aprovechan del control de congestión para ganar ancho de banda de la red.
  - CWR (1bit): se activa por el emisor para indicar que ha recibido un paquete TCP con el flag ECE activado y ha respondido con el mecanismo de control de congestión.
  - ECE (1 bit): indicación de congestión. A pesar de su utilidad para detectar la congestión, también son susceptibles de ser usados de forma maliciosa, por lo que estos flags no se suelen utilizar y la congestión se trata mediante mecanismos de gestión de colas.
  - URG (1 bit): indica que el campo del puntero urgente es válido.
  - ACK (1 bit): indica que el paquete recibido es válido. Todos lo paquetes enviados después del paquete SYN inicial deben tener activo este flag. Si el paquete solo tiene activo este flag, suele denominarse como paquete ACK.
  - PSH (1 bit): el receptor debe pasar los datos a la aplicación tan pronto como sea posible, sin tener que esperar a recibir más datos.
  - RST (1 bit): reinicia la conexión, cuando falla un intento de conexión o al rechazar paquetes no válidos.
  - SYN (1 bit): establecimiento de la conexión. Sincroniza los números de secuencia para iniciar la conexión. Un paquete con este flag activo suele denominarse como paquete SYN.
  - FIN (1 bit): solicitud de liberación de la conexión. Si el paquete solo tiene activo este flag, suele denominarse como paquete FIN.
- Tamaño de ventana (16 bits): número máximo de bytes que pueden guardarse en el espacio de memoria de recepción, es decir, el número máximo de bytes pendientes de reconocimiento. Es un mecanismo de control de flujo.
- Suma de verificación (16 bits): utilizado para la comprobación de errores tanto en la cabecera como en los datos. Protege la integridad de TCP.
- Puntero urgente (16 bits): cantidad de bytes desde el número de secuencia que indica el lugar donde acaban los datos urgentes.
- Opciones: permite añadir características no soportadas por la cabecera fija. Algunas opciones solo se pueden enviar en la fase de establecimiento:

  - Tamaño máximo de segmento (MSS): define la mayor cantidad de datos, en bytes, que TCP está dispuesto a recibir en un solo paquete.
  - Escalado de ventana: ya que el campo de tamaño de la ventana no se puede ampliar, se utiliza un factor de escala. Ambas partes deben enviar la opción para permitir el escalado de ventana en las dos direcciones.
  - Acuse de recibo selectivo (SACK): permite que el receptor reconozca bloques discontinuos de paquetes que se recibieron correctamente.
  - Timestamp: ayuda a determinar el orden en que se enviaron los paquetes.

- Relleno: se utiliza para asegurarse que la cabecera acaba con un tamaño múltiplo de 32 bits.

- Datos: los datos que se envían en el paquete. Longitud variable.

2.1.2 Estados de la conexión y transiciones

Al igual que cualquier tipo de comunicación, una conexión TCP progresa a través de una serie de estados y en respuesta a determinados eventos. De forma resumida, se presentan los estados de la conexión con el objetivo de entender las fases de la comunicación que se explicarán en la siguiente sección.

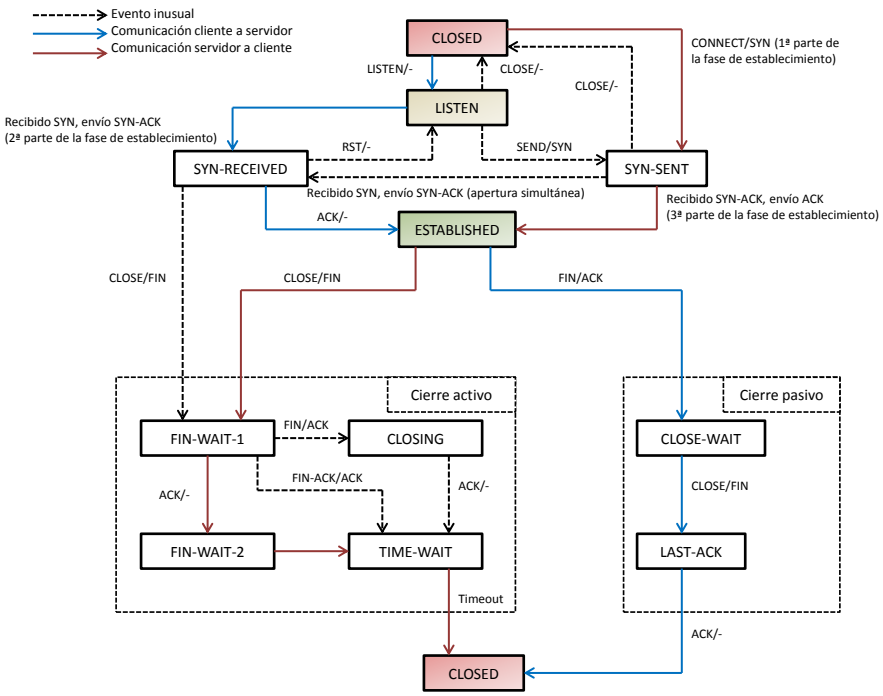


Figura 2.1 Diagrama de transición de estados de TCP

LISTEN

Estado solo aplicable a la parte servidora. Representa la espera de una petición de conexión TCP. Se denomina inicio pasivo de la conexión.

SYN-SENT

Estado solo aplicable a la parte cliente. Representa la espera de una respuesta después de haber enviado una solicitud de conexión (SYN). Se denomina inicio activo de la conexión. Si se recibe un paquete SYN-ACK, se responde con un ACK y la conexión transita al estado ESTABLISHED.

SYN-RECEIVED

Estado solo aplicable a la parte servidora. Representa la espera de una confirmación de la solicitud de conexión. Este estado indica que la conexión está semiabierta (se ha recibido un paquete SYN y se ha contestado con el correspondiente paquete SYN-ACK), y por tanto la confirmación de la solicitud todavía no se ha recibido. Si se recibe el ACK, el estado de la conexión transita al estado ESTABLISHED.

ESTABLISHED

Estado aplicable a la parte servidora y cliente. Representa una conexión abierta, y si ambas partes de la conexión están en este estado, los datos se pueden intercambiar. Este estado se mantendrá hasta que se cierre la conexión.

FIN-WAIT-1

Estado aplicable a la parte que inicia el cierre de la conexión. Después de enviar un paquete FIN, espera un reconocimiento de esa solicitud de finalización y que la otra parte envíe su propio FIN. En este estado todavía se pueden recibir datos, pero no enviarlos. Si se recibe un ACK, el estado de la conexión transita al estado FIN-WAIT-2. Si se recibe un paquete FIN al mismo tiempo, el estado de la conexión transita al estado CLOSING.

FIN-WAIT-2

Estado aplicable a la parte que inicia el cierre de la conexión. Representa la espera de una solicitud de finalización (FIN). Si se recibe un paquete FIN, se contesta con un último paquete ACK y el estado de la conexión transita al estado TIME-WAIT.

CLOSING

Estado aplicable a la parte que inicia el cierre de la conexión (solo en el caso de un cierre simultáneo). Representa la espera de un reconocimiento a una petición de finalización de conexión. Si se recibe un ACK para el paquete FIN enviado, el estado de la conexión transita al estado TIME-WAIT.

TIME-WAIT

Estado aplicable a la parte que inicia el cierre de la conexión. Se espera un período de tiempo equivalente al doble del tiempo máximo de vida del segmento (MSL) para asegurarse que el ACK

enviado fue recibido. En este intervalo de tiempo, el puerto local no está disponible para nuevas conexiones. Este estado evita errores en el caso de que paquetes con retardo pertenecientes a la conexión actual se entreguen en conexiones posteriores. Al expirar el tiempo, el estado de la conexión transita al estado CLOSED.

CLOSE-WAIT

Estado aplicable a la parte que no inicia el cierre de la conexión. Una vez recibido el paquete FIN, se envía un ACK para confirmar el proceso de finalización de la conexión. A continuación, se espera a que la aplicación indique que está lista para cerrarse. Cuando se recibe esa confirmación de la capa de aplicación, se envía el paquete FIN correspondiente. Una vez enviado el paquete FIN, el estado de la conexión transita al estado LAST-ACK.

LAST-ACK

Estado aplicable a la parte que no inicia el cierre de la conexión. Representa la espera a un reconocimiento (ACK) de la solicitud de finalización (FIN) enviada previamente. Cuando se recibe ese paquete ACK, se cierra la conexión, y por tanto el estado de la conexión transita al estado CLOSED.

CLOSED

Estado aplicable a la parte servidora y cliente. Representa que la conexión está cerrada.

2.1.3 Estructura de datos y fases de la conexión

Los mecanismos que utiliza TCP para proporcionar fiabilidad, control de flujo y control de congestión requieren que el protocolo establezca y mantenga cierta información sobre el estado de la conexión. TCP utiliza una estructura de datos conocida como Bloque de Control de la Transmisión (TCB) para realizar este seguimiento. Un TCB contiene información sobre el estado de la conexión y parámetros necesarios para procesarla, como los números de puerto que la identifican, los punteros a las posiciones de memoria donde se guardan los datos entrantes y salientes, el tamaño de la ventana actual, o las variables que controlan el número de bytes recibidos y reconocidos. Como es de suponer, cada dispositivo –cliente y servidor– mantiene su propio TCB para la misma conexión. Esta estructura se crea justo al inicio del proceso de establecimiento de la conexión.

Por lo tanto, como cualquier protocolo orientado a la conexión, TCP requiere que se establezca la conexión antes de que comience la transferencia de datos. Una vez finalizada, se liberan esos recursos para futuras conexiones.

La figura 2.2 muestra estas fases. En ella, además, se identifican los diferentes estados de la conexión TCP.

La fase de establecimiento consiste en una negociación en tres pasos. Cabe destacar que antes de permitir conexiones, el servidor debe abrir el puerto correspondiente para permitir solicitudes de clientes, denominada como apertura pasiva. Una vez abierto, los clientes pueden iniciar conexiones a ese puerto. A continuación se muestran los tres pasos de la fase de establecimiento:

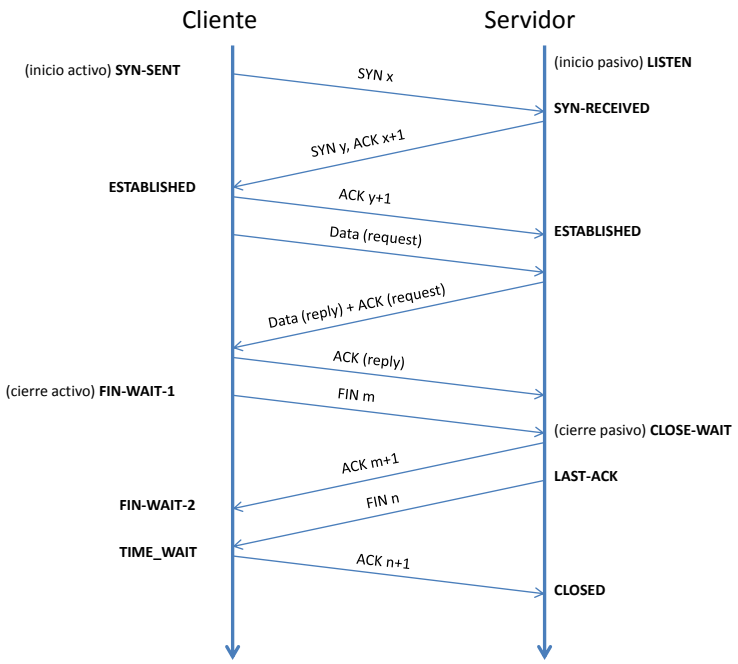


Figura 2.2 Diagrama de una conexión TCP

1. SYN: el cliente envía un paquete SYN al servidor con un número de secuencia aleatorio (x) y entra en el estado SYN-SENT. Se crea un TCB para la conexión iniciada.

2. SYN-ACK: el servidor cambia de estado LISTEN a SYN-RECEIVED, asigna recursos para la conexión entrante (TCB) y responde al cliente con un paquete SYN-ACK que contiene otro número de secuencia (y). Además incluye en su respuesta el número de reconocimiento, que se incrementa en uno respecto al número de secuencia recibido del cliente (x+1).

3. ACK: el cliente envía de vuelta un paquete con el número de secuencia actualizado (x+1) y el número de reconocimiento también se incrementa en uno respecto al número de secuencia recibido del servidor (y+1). En este punto, el cliente y el servidor han recibido un reconocimiento del establecimiento y se establece la conexión (ESTABLISHED).

Uno de los objetivos de esta fase es la prevención de conexiones duplicadas. Durante esta fase no se admite el intercambio de datos, y en el caso de encontrarlos, éstos se almacenan hasta acabar la fase de establecimiento.

Una vez que ambos dispositivos han completado el establecimiento de conexión, TCP informa a la capa de aplicación que puede iniciar la transferencia de datos. En esta fase la pila TCP envía los datos de la capa de aplicación en paquetes con el flag PSH activo. El otro extremo recibe los datos y responde con un paquete con el flag ACK activo y el número de reconocimiento actualizado para indicar que ha

recibido bien los datos. Algunos paquetes pueden llegar desordenados o incluso pueden perderse. Los números de secuencia y de reconocimiento permiten gestionar estas situaciones.

Por último, cliente y servidor realizan la desconexión de forma independiente. Cada extremo de la conexión envía un paquete FIN cuando no tiene más datos que enviar, y espera al paquete ACK correspondiente. El extremo que envía el primer FIN realiza el cierre activo, y el otro extremo el cierre pasivo. Técnicamente, cualquiera de los dos extremos puede iniciar el cierre, aunque en la mayoría de aplicaciones suele ser el cliente. Cuando se realiza un cierre activo, la conexión entra en estado TIME-WAIT. Por el contrario, el cierre pasivo lleva directamente al estado CLOSED. Al cerrar la conexión, tanto la combinación de puertos de origen y destino como el TCB asignado quedan disponibles de nuevo.

Como se ha explicado, la información de la conexión se guarda en un TCB, pero, ¿dónde se guardan estos TCBs? Los TCBs se guardan en una estructura de datos abstracta denominada cola de conexiones, cuyo tamaño está determinado por un parámetro de la pila TCP/IP. Cada posición en la cola se denominada entrada, y la tasa a la que se pueden aceptar nuevas conexiones (un nuevo TCB) es igual al número de entradas que caben en la cola de conexiones dividido por el tiempo promedio que cada entrada pasa en la cola. Por lo tanto, cuanto mayor sea la cola, mayor será la tasa a la que se pueden aceptar nuevas solicitudes de conexión. No obstante, el tamaño máximo de la cola de conexiones dependerá en gran medida de la memoria disponible, es decir, dispositivos con una memoria reducida tendrán una cola de conexiones también reducida. Por último, la pila TCP/IP tiene dos opciones para implementar la cola de conexiones:

- La implementación utiliza una cola. Cuando se recibe un paquete SYN, se responde con un paquete SYN-ACK y se agrega el TCB generado para esa conexión a la cola. Cuando se recibe el ACK correspondiente, el TCB cambia su estado a ESTABLISHED y puede traspasarse a la aplicación.

- La implementación utiliza dos colas. Cuando se recibe un paquete SYN, se responde con un paquete SYN-ACK y se agrega el TCB a la cola de conexiones incompletas. Cuando se recibe el ACK correspondiente, el TCB se mueve a la cola de conexiones completas y puede traspasarse a la aplicación. Por lo tanto, la cola de conexiones incompletas solo puede contener TCBs en estado SYN-RECEIVED.

Las implementaciones TCP derivadas del sistema operativo BSD usan el primer enfoque. Por el contrario, Linux utiliza el segundo enfoque desde la versión 2.2.

2.2 Balanceo de carga

A medida que Internet creció en la década de 1990, la demanda de tráfico superó la capacidad de servidores individuales. Se necesitaba una manera de recibir múltiples solicitudes de un recurso y distribuirlas de manera efectiva a través de varios servidores para compartir la carga de trabajo. De

esta manera, la tecnología de balanceo de carga se convirtió en herramienta de apoyo fundamental para sistemas que buscaban escalabilidad y una alta disponibilidad.

Un balanceador de carga es un dispositivo dedicado que se sitúa entre los servidores y los clientes y, tal como su nombre indica, distribuye las solicitudes entrantes a un servidor sobre la base de un algoritmo de planificación, siendo el algoritmo estándar más conocido, simple y utilizado el Round Robin –selecciona todos los elementos de un grupo siguiendo un turno rotativo. A medida que aumenta la demanda, se puede añadir un nuevo servidor al clúster y el balanceador empezará a asignarle tráfico, siempre de manera transparente para el cliente. En un clúster de balanceo de carga puede existir también un espacio compartido de almacenamiento para los servidores, de modo que sea sencillo y transparente tener el mismo contenido y proporcionar los mismos servicios.

Existen dos formas principales de utilizar los balanceadores de carga: para aumentar la capacidad, y para mejorar la fiabilidad y la tolerancia a fallos [Limoncelli, 2017]. Si un servidor deja de estar disponible, el balanceador de carga redirige el tráfico a los servidores restantes. Entre estos dos objetivos operativos, se considera prioritario aumentar la capacidad del clúster de dispositivos C2.

Los balanceadores de carga se agrupan generalmente en dos categorías: una combinación de técnicas de enrutamiento basadas en las capas OSI 2/3/4 (generalmente denominados como capa 4, su capa más alta en el modelo OSI), y de capa 7 (funcionando como un proxy inverso). Qué categoría escoger para implementar el balanceador depende del tipo de aplicación.

Los balanceadores de capa 4 utilizan la información definida en las capas de enlace datos, red y transporte como la base para decidir cómo distribuir las solicitudes de clientes a través de un grupo de servidores. Por ejemplo, basarán la decisión de distribución de la carga en parámetros como las direcciones MAC e IP de origen y destino, y por tanto sin tener en cuenta el contenido del paquete.

Los balanceadores de carga de capa 7 operan al nivel más alto del modelo OSI, la capa de aplicación (como el protocolo HTTP), funcionando como proxy. Por lo tanto estos balanceadores basan sus decisiones de enrutamiento en el contenido del paquete, p. ej. la dirección URL del servicio o el tipo de datos (texto, vídeo, gráficos). Si el balanceador no utiliza la información de la carga útil para distribuir la conexión, se considera un balanceo de capa 4.

Por último, los balanceadores de carga se pueden clasificar también en balanceadores software y hardware. Los balanceadores hardware son equipos especializados que incluyen circuitos integrados de aplicación específica (ASIC) que permiten la distribución de tráfico de red sin la sobrecarga de un sistema operativo de propósito general. Sus inconvenientes son el precio, que son poco o nada flexibles, y que son difíciles de actualizar o configurar. Los balanceadores de carga basados en software, por otra parte, se ejecutan en sistemas operativos y equipos hardware estándar, como ordenadores de sobremesa, y por lo tanto son más flexibles, escalables y fáciles de configurar. En general, los balanceadores de carga basados en hardware son más eficientes que las soluciones basadas en software. Sin embargo, a medida que los ordenadores han ido aumentando prestaciones, la línea que separa estos balanceadores se ha ido desvaneciendo, y en la actualidad la mayoría de balanceadores son software.

2.2.1 Revisión bibliográfica

En primer lugar, conviene destacar que los trabajos existentes en la literatura van en dos direcciones: a) diseños de algoritmos más eficientes para distribuir la carga entre los distintos componentes del clúster, y b) esquemas de balanceo de carga y enrutado. La literatura presenta multitud de trabajos en la primera dirección, mientras que en la segunda dirección hay pocos trabajos y además son antiguos, ya que los esquemas de balanceo se han mantenido con el paso del tiempo [Milani y Navimipour, 2016]. Esta sección solo va a centrarse en los trabajos relativos a esquemas de balanceo, ya que el objetivo es seleccionar e implementar el esquema más eficiente y ligero para dispositivos C2, y por lo tanto el algoritmo que se utilice para el balanceo queda fuera del ámbito de este trabajo.

En cuanto a los balanceadores, como se ha comentado antes, se agrupan en dos categorías. Funcionado como un proxy, lo que significa que el balanceador divide la comunicación entre cliente y servidor en dos conexiones independientes, es decir, además de la conexión entre el cliente y el balanceador de carga, hay otra conexión entre el balanceador y el servidor que se inicia cuando acaba el establecimiento de la primera conexión. Esta división tiene varias ventajas, como utilizar cookies persistentes para que el cliente se comunique con el mismo servidor o funcionar como una defensa contra ataques DoS, ya que el servidor nunca recibirá paquetes de un atacante.

No obstante, este modo de balanceo es el más costoso en términos de cómputo y memoria –además de provocar una mayor latencia– ya que se deben tomar en consideración muchos más campos del paquete. En este modo, además de la reescritura de las capas 2, 3 y 4 del encabezado, los números de secuencia y reconocimiento deben mantenerse sincronizados para la conexión desde el cliente al balanceador de carga, y desde el balanceador al servidor. El objetivo de este trabajo es implementar un balanceador de carga en un dispositivo con capacidades de cómputo y memoria limitadas, por lo tanto se analizarán métodos comunes de balanceo de capa 4 por ser menos exigentes en recursos. Al no utilizar la información de la carga útil para distribuir una conexión (balanceo de capa 7), todos los servidores del clúster deben proporcionar el mismo servicio.

Para realizar el análisis de estos métodos de balanceo se han utilizado varios trabajos: un artículo referente al balanceo [Cardellini et al., 1999] y los libros *Data Center Fundamentals* de Mauricio Arregoces y Maurizio Portolani, *Server Load Balancing* de Tony Bourke, y *Load Balancing Servers, Firewalls, and Caches* de Chandra Kopparapu, en los que se describen los conceptos básicos de los centros de procesamiento de datos para construir clústeres de servidores que mejoren el rendimiento de un sitio web.

2.2.1.1 Anycast

Anycast es una técnica de enrutamiento de capa 3 en la que varios equipos tienen exactamente la misma dirección IP. Si estos equipos duplicados proporcionan el mismo servicio, los clientes que intentan acceder a esa dirección IP reciben el servicio del equipo más cercano a nivel de red.

Aunque esta técnica de enrutamiento evita tener que utilizar un dispositivo dedicado, no fue diseñada originalmente para funcionar como balanceador de carga, y por tanto no comprueba si hay servidores

inactivos o inaccesibles. Además, el balanceo de carga debe dividir de la manera más equitativa posible el trabajo, y esta técnica no puede garantizarlo ya que para los clientes un mismo servidor podría ser el equipo más cercano Anycast.

No obstante, es posible utilizar Anycast como una solución complementaria para la conmutación en caso de fallo entre diferentes clústeres [Weiden y Frost, 2010].

2.2.1.2 DNS

El protocolo DNS (Domain Name System) se utiliza para asignar a una dirección IP un nombre, de esta forma la dirección IP de un servidor puede cambiar, pero el nombre no lo hace. Un servidor DNS por tanto permite traducir nombres de dominio en direcciones IP y viceversa.

Este protocolo se puede utilizar también como esquema de balanceo de carga mediante la configuración de un dominio, de tal manera que las solicitudes al dominio se distribuyan a través de un grupo de servidores.

El balanceo de carga DNS se basa en el hecho de que la mayoría de los clientes utilizan la primera dirección IP que reciben para un dominio. En la mayoría de distribuciones Linux, el servidor DNS por defecto envía la lista de direcciones IP en un orden diferente cada vez que responde a un nuevo cliente, usando el algoritmo Round Robin. Como resultado, cada cliente dirige sus peticiones a diferentes servidores, y de esta manera distribuyen la carga entre el grupo de servidores. Por lo tanto, este método de balanceo tiene un uso de cómputo y memoria mínimo en el servidor DNS. No obstante, el servicio necesita un nombre de dominio y por tanto un mayor tráfico de red debido a la comunicación con el servidor DNS.

Al igual que en el caso anterior, el protocolo DNS no fue diseñado originalmente para funcionar como balanceador de carga, y por tanto no comprueba si hay errores en los servidores o en la red. Además, siempre se devuelve el mismo conjunto de direcciones IP para un dominio, incluso si los servidores están inactivos o son inaccesibles. Las direcciones resueltas por lo general se almacenan en caché, tanto por los servidores DNS intermedios como por los clientes, para reducir el tráfico DNS en la red. Esto puede provocar que un cliente utilice siempre el mismo servidor, y por tanto el balanceo de carga no sea equitativo.

En definitiva, este esquema se utiliza generalmente como una solución complementaria, para distribuir la carga entre centros de datos en diferentes localizaciones [Hong et al., 2006], es decir, cuando los servidores no están en la misma ubicación física. Una vez la conexión llega a uno de esos centros de datos, se utilizan los modos de balanceo que se presentan a continuación.

2.2.1.3 NAT

El mecanismo NAT (Network Address Translation) hace que los equipos externos a una red de área local (LAN) vean una única dirección IP; mientras que los equipos internos vean las direcciones IP de cada equipo. Por lo tanto, NAT permite a un equipo de red modificar los paquetes en la capa 3 (direcciones IP)

para la comunicación entre equipos pertenecientes a diferentes redes. Este mecanismo también permite desplegar configuraciones de balanceo.

En la configuración de dos brazos (*Two-Arm*), el clúster de servidores estará en un lado de la red, los clientes en otro, y el balanceador de carga será la puerta de enlace que conecta esos dos segmentos de red. Los brazos hacen referencia al número de conexiones lógicas de red que el balanceador tiene. Por lo tanto, en esta configuración los servidores deben establecer su puerta de enlace predeterminada para que apunte al balanceador de carga.

En este esquema de balanceo, el cliente se conecta a una dirección de destino virtual (dirección IP del servicio asociada al balanceador) y el balanceador de carga hace NAT de la dirección IP de destino a una de las direcciones de los servidores, además de cambiar la dirección MAC de destino a la del propio servidor. El tráfico de retorno vuelve a través del balanceador, y se hace NAT de la dirección IP de origen (servidor) a la dirección IP del servicio (balanceador) para enviar la respuesta al cliente. Este esquema recibe también el nombre de Half-NAT ya que solo cambia una de las dos direcciones IP en cada paso por el balanceador.

En una configuración NAT de un solo brazo (*One-Arm*), el balanceador no está físicamente en la ruta del tráfico como en el caso anterior, sin embargo tiene que entrar en la ruta del tráfico de alguna manera para tener control sobre las conexiones en ambos sentidos. En esta configuración, el balanceador solo tiene una interfaz de red (un brazo), y por tanto esta interfaz debe configurarse en la misma subred (capa 2) que los servidores.

El tráfico del cliente llegará al balanceador, que tiene la IP virtual del servicio. El algoritmo de planificación escogerá un servidor al que el balanceador de carga reenviará el tráfico con la dirección IP de destino cambiada mediante NAT. A diferencia de la configuración de dos brazos, el balanceador de carga también tiene que hacer NAT en la dirección IP de origen para que la respuesta del servidor llegue al balanceador, y no directamente al cliente ya que éste no espera una respuesta desde la IP del servidor. Desde la perspectiva de los servidores, todo el tráfico llega desde el balanceador de carga.

Este esquema recibe también el nombre de Full-NAT, ya que las direcciones IP de origen y de destino se reescriben para garantizar que el tráfico pasa por el balanceador de carga en ambas direcciones. Esta topología NAT permite conectarse a la dirección IP del servicio desde la misma subred en la que están los servidores.

En definitiva, el balanceo NAT es transparente para los servidores ya que éstos mantienen sus direcciones IP y permite la inspección del tráfico de red ya que el balanceador ve tanto el tráfico entrante como el saliente, y suele utilizarse para escalar servicios web [Patel et al., 2013]. Este último hecho requiere más cómputo en el balanceador, ya que el balanceador tiene que realizar modificaciones en las cabeceras de las capas 2 y 3 para el tráfico entrante y saliente. El uso de memoria será proporcional al número de conexiones activas, ya que se debe guardar en una estructura de datos cada conexión para que los siguientes paquetes se envíen al mismo servidor.

2.2.1.4 DSR

El modo Direct Server Return (DSR) –denominado también Direct Routing (DR)– tampoco necesita cambios en la infraestructura y por lo tanto el balanceador puede estar en la misma subred que los servidores.

El tráfico entrante desde el cliente es recibido por el balanceador de carga, que tiene la dirección IP del servicio. El balanceador selecciona un servidor y envía el paquete modificando solo la dirección MAC de destino para que coincida con la del servidor. El balanceador no reescribe la información de la cabecera de capa 3, lo que requiere que los servidores del clúster se configuren con la dirección IP del servicio –denominada como IP virtual o VIP– para recibir tráfico. Por lo tanto, se necesita tener una interfaz de red habilitada para esa VIP, pero esa interfaz no debe responder a solicitudes ARP (Address Resolution Protocol). Estas solicitudes solo serán respondidas por el balanceador, haciendo que todo el tráfico entrante sea dirigido a él. Como el servidor tiene la misma dirección VIP configurada en una de sus interfaces, acepta el paquete y lo procesa. Dado que el tráfico de retorno utiliza la dirección VIP como la dirección IP de origen, el cliente asume que el tráfico se envía desde el balanceador de carga, y por tanto no es necesario que el balanceador esté involucrado en esta fase. De ahí viene su nombre.

Al no procesar el tráfico de retorno, el rendimiento del balanceador se incrementa. Como ejemplo, el tráfico web tiene una relación de 1:8, lo que significa un paquete entrante por cada 8 paquetes salientes. Si se implementa DSR, la carga de trabajo del balanceador de carga puede reducirse en un factor de 8 [Bourke, 2001]. Este modo de balanceo por tanto es útil cuando se quieren velocidades altas, p.ej. para optimizar el balanceo de carga en OpenFlow [Koerner y Kao, 2013].

En este esquema el uso de cómputo será menor que NAT ya que solo se manipulan los paquetes a capa 2, y el de memoria seguirá siendo proporcional al número de conexiones activas para las que se necesita guardar en una estructura de datos la información básica de la conexión para que los sucesivos paquetes se envíen al mismo servidor. No obstante, se deben configurar correctamente todos los equipos de la misma subred, puede ser más difícil de depurar ya que el balanceador solo ve una parte del tráfico, y el servidor debe responder a paquetes dirigidos a la VIP sin contestar a solicitudes ARP, de lo contrario todo el tráfico entrante se dirigiría a ese servidor en vez de al balanceador de carga. Además, al no ver el tráfico de retorno se deben eliminar conexiones activas utilizando un tiempo de inactividad predeterminado.

Una extensión del modo DSR, denominado IP tunnel, permite que el tráfico entre el balanceador de carga y el servidor se puede establecer entre redes remotas.

En DSR solo se cambian las direcciones MAC, mientras que las direcciones IP permanecen igual ya que los servidores y el balanceador se encuentran en la misma red. En el modo túnel, el balanceador encapsula la petición en un túnel IP al servidor. El cliente manda un paquete a la dirección VIP, el balanceador encapsula el paquete original dentro de un paquete con la dirección IP del balanceador como dirección de origen, y de destino la del servidor remoto. El servidor recibe el paquete y responde con la dirección VIP al cliente.

Por lo tanto, este modo tiene los mismos beneficios e inconvenientes de DSR, pero necesita que el balanceador y los servidores sean capaces de incluir y extraer paquetes en un nuevo paquete IP para que funcione el enrutamiento. Este proceso aumenta el cómputo en comparación con el modo DSR.

2.2.2 Conclusión

Dadas las diferentes opciones para configurar el balanceo de carga, la selección debe hacerse en función de los requisitos de servicio. Otra consideración a tener en cuenta es el rendimiento [Arregoces y Portolani, 2003].

En este caso se necesita un balanceo de carga a nivel local, que no necesite cambios en la infraestructura de red, y por tanto una configuración de un brazo. Además, un dispositivo C2 por lo general no tendrá dos interfaces de red. En definitiva, las opciones se reducen a DSR y NAT de un brazo.

En cuanto al rendimiento, la regla general para los balanceadores de carga es que cuantas más tareas de procesamiento requieran, menor será su rendimiento medio. DSR solo ve el tráfico entrante y no realiza reescritura de cabeceras en capas 3 y 4, solo en capa 2. Por el contrario, el modo NAT de un brazo procesa la información de cabecera en las capas 2 y 3, asumiendo que las operaciones de reescritura se realizan en las direcciones IP de origen y destino de los paquetes entrantes y salientes.

Entre las configuraciones de balanceo de carga de capas 2/3/4, se concluye que la opción que mejor se adapta a las limitaciones de un dispositivo C2 es el modo DSR. Se trata de un balanceo de carga asimétrico que reduce el procesamiento en el balanceador porque los paquetes que se envían desde los servidores no pasan por él. Esta característica es particularmente deseable en caso de que el servidor sea el principal productor de tráfico en la conexión. Debido a que el balanceador solo procesa los paquetes entrantes, el rendimiento de red del clúster se verá menos afectado por el balanceador (menos probable que actúe como cuello de botella), y por tanto será mayor que en el modo NAT de un brazo. Esta característica es fundamental ya que un dispositivo C2 tendrá un ancho de banda limitado, y el objetivo debe ser conseguir el mayor rendimiento posible para ese ancho de banda. No obstante, al tratarse de un balanceo de carga asimétrico, el balanceador no podrá aplicar medidas específicas de protección ante ataques DoS[1]. De ahí la necesidad de implementar medidas adicionales, también en los servidores.

2.3 Inundación SYN

Esta sección describe un ataque DoS específico –conocido como inundación SYN– que explota una característica de la implementación del protocolo TCP.

El ataque por inundación SYN llegó a ser conocido en 1996, cuando dos revistas de hackers –2600 y Phrack– publicaron descripciones del ataque junto con el código fuente para llevarlo a cabo.

El primer ataque de inundación SYN a gran escala se produjo contra Panix, el proveedor de servicios de Internet (ISP, por la siglas en inglés de Internet Service Provider) más antiguo y grande del área de

[1] https://supportforums.cisco.com/document/91121/configure-ace-direct-server-return-mode#SYN_Flood_Protection_in_DSR_environment

Nueva York, el 6 de septiembre de 1996[2] y lo dejó fuera de servicio durante una semana. A pesar de que Panix superó el ataque, la nueva amenaza de los ataques DoS se convirtió en una prioridad para los ingenieros y administradores de redes.

2.3.1 El ataque

Para aceptar conexiones, lo primero es crear un socket. Los sockets son un punto de comunicación por el cual un proceso puede emitir o recibir información de un dispositivo de la red. Una vez creado el socket, la disposición para recibir conexiones entrantes se especifica con la llamada listen(). Cuando el servidor llama a la función listen(), el sistema cambia un socket del estado *CLOSED* al estado *LISTEN*, realizando de esa forma un inicio pasivo.

Una vez en el estado *LISTEN*, el servidor tiene que crear e inicializar varias estructuras de datos, entre ellas un espacio de memoria para guardar las conexiones entrantes de ese socket. Como se ha explicado anteriormente, esa estructura de datos se denomina comúnmente TCB. El uso de memoria de un solo TCB depende de las opciones habilitadas en la conexión y del sistema operativo utilizado. Cada TCB se almacenará en una posición de la cola de conexiones TCP, cuyo número máximo dependerá de la configuración y la memoria del sistema.

Cuando se recibe un paquete SYN, se envía de vuelta un paquete SYN-ACK y la conexión entrante pasa por un estado intermedio denominado *SYN-RECEIVED*. De esta forma, un TCB se forma ante la recepción de un paquete SYN, antes de que la conexión se establezca completamente. Solo cuando se recibe el ACK correspondiente el servidor cambia el estado de la conexión a *ESTABLISHED* y se traspasa a la aplicación. Por lo tanto, enviar un gran número de paquetes SYN sin su correspondiente ACK llenará de conexiones en estado *SYN-RECEIVED* la cola de conexiones.

La manera en la que TCP proporciona fiabilidad es que cada extremo debe reconocer cada paquete recibido, indicando el número de secuencia del último paquete recibido correctamente. De igual forma, la ausencia de reconocimiento muestra problemas en las condiciones de red entre el emisor y el receptor.

Para garantizar la transmisión de paquetes, la implementación TCP de cada dispositivo utiliza un mecanismo de retransmisión. El problema es que no hay garantías de que un paquete retransmitido sea recibido. Por lo tanto, el protocolo TCP retransmitirá un paquete un cierto número de veces antes de concluir que hay un problema y terminar la conexión.

Cada sistema es responsable de decidir cuándo y cuántas veces se repetirá la retransmisión. En general, sucesivas retransmisiones utilizan tiempos de espera exponenciales, por lo que los últimos reintentos pueden tomar un tiempo considerable (en los sistemas Linux suele ser de 3 minutos).

De esta forma, una inundación SYN se puede ajustar para utilizar un menor número de paquetes que un ataque de fuerza bruta que simplemente envía un alto volumen de paquetes para saturar el servidor. Conociendo un poco las características del servidor, como el número máximo de estructuras TCB que puede almacenar en la cola de conexiones, y cuánto tiempo se mantiene un TCB en el estado *SYN-RECEIVED* antes de ser descartado, el atacante puede enviar el número exacto de paquetes SYN

[2] http://archive.wired.com/science/discoveries/news/1996/12/1052

que corresponda con el número de estructuras TCB, y repetir este proceso periódicamente a medida que los TCB se descartan con el fin de mantener al servidor ocupado ininterrumpidamente. A este tipo de ataque se le puede denominar como un ataque de baja tasa. En ese punto, se ignorarán todas las solicitudes al puerto TCP, lo que significa que el puerto de destino está inundado. Conviene recordar que los puertos son utilizados por el sistema para identificar procesos de red, y junto con la dirección IP, un puerto TCP proporciona un punto de conexión para la comunicación de red.

En definitiva, el defecto en el protocolo TCP que hace que las inundaciones SYN sean eficaces reside en que para el pequeño coste del envío de un paquete, el servidor se ve obligado a reservar el estado de esa conexión inicial (*SYN-RECEIVED*) en una estructura de memoria específica (TCB). Ante múltiples solicitudes de conexión, el servidor asigna memoria para estas conexiones semiabiertas, y finalmente no tiene suficientes recursos para procesar nuevas solicitudes.

De manera resumida, se pueden distinguir 3 métodos de ataque. El ataque directo utiliza la dirección IP del propio atacante y por tanto será capaz de recibir los paquetes SYN-ACK de respuesta del servidor. Por otra parte, el atacante puede utilizar direcciones de origen falsificadas e inaccesibles. Si las direcciones IP fuesen accesibles, los dispositivos que recibiesen los paquetes SYN-ACK del servidor responderían con un paquete RST ya que no tendrían ese puerto abierto, y por tanto supondría la liberación de la estructura TCB en estado *SYN-RECEIVED* en el servidor y el fin del ataque. Por último, el ataque podría ejecutarse de manera distribuida (DDoS). Este tipo de ataque es el más peligroso ya que múltiples dispositivos atacan al mismo objetivo a la vez.

2.3.2 Estado del arte

La literatura presenta un gran número de trabajos relativos a los ataques SYN. Estos trabajos se pueden clasificar principalmente en dos categorías: a) detectar las inundaciones SYN con mayor precisión y rapidez, y b) mitigar el impacto de los ataques de inundación SYN. En esta sección solo se van a analizar los trabajos relativos a la mitigación, ya que para un dispositivo restringido –como un C2– se considera más prioritario mitigar los efectos del ataque que reducir la tasa de falsos positivos en la detección.

Existen dos clases de soluciones dependiendo del lugar donde se implementen las defensas. La primera clase de soluciones implica el endurecimiento de la implementación TCP en el dispositivo local, incluyendo la alteración de los algoritmos y estructuras de datos utilizadas para el establecimiento de la conexión.

La segunda clase de soluciones consiste en el endurecimiento de la red, ya sea para disminuir la probabilidad de las condiciones previas de ataque (como la propagación de paquetes IP con direcciones de origen falsificadas), o para utilizar dispositivos intermedios que puedan aislar los servidores de paquetes SYN falsos.

A continuación se presentan las soluciones existentes para ambas clases, ya sean soluciones de práctica común o propuestas académicas.

2.3.2.1 Mitigaciones de práctica común

En esta sección se presentan las técnicas comunes de mitigación y sus relaciones costo-beneficio. La mayoría de estas técnicas están documentadas en el RFC 4987 y se aplican en los dispositivo de red y sistemas operativos actuales.

2.3.2.1.1 A nivel de red A nivel de red el método de mitigación más básico es el de aplicar técnicas de filtrado, descritas en el RFC 2827. Cuando los atacantes inyectan paquetes con direcciones de origen falsas, los routers reenvían esos paquetes a su destino sin comprobar la validez de las direcciones de origen de los paquetes. Estos paquetes consumen ancho de banda de red y son a menudo parte de algún ataque. El objetivo del filtrado es que los dispositivos de interconexión no reenvíen paquetes con una dirección IP de origen que no pertenezca a esa red. Esta solución no tiene un coste asociado de cómputo o memoria, simplemente se establecen unas reglas de tráfico y si llegan paquetes que no las cumplen, se descartan esos paquetes. El problema es que estas reglas de filtrado no siempre se aplican, y además su aplicación no depende del usuario, sino de los administradores de red.

Otro método de mitigación a nivel de red es el uso de un dispositivo cortafuegos (firewall) o proxy [Mirzaie et al., 2010]. Estos dispositivos de red actúan como intermediarios entre los clientes remotos y los servidores locales. Por lo tanto se encargan de realizar el establecimiento de la conexión por el servidor. Es decir, el paquete SYN debe pasar por el firewall/proxy, y éste responde con un SYN-ACK. Si recibe un ACK, el firewall/proxy iniciará un establecimiento de la conexión con el servidor, utilizando números de secuencia nuevos. En este modo, los números de secuencia y reconocimiento deben mantenerse sincronizados para las dos conexiones. Esto supone mantener dos conexiones independientes en el firewall/proxy, y por tanto un uso duplicado de cómputo y memoria por cada conexión. Destacar que el firewall/proxy se convertiría en el objetivo del ataque SYN, los servidores detrás del firewall/proxy no recibirían paquetes del atacante, y por tanto podría implementar métodos de mitigación a nivel local que se presentarán en la siguiente sección.

Por último, existen dispositivos denominados *Active Monitor* cuya función es monitorizar e inyectar tráfico en caso necesario. Es decir, este dispositivo tiene la capacidad de enviar paquetes RST para finalizar conexiones con una alta probabilidad de ser falsas. Por lo tanto esta solución también depende de un dispositivo de red específico, y el coste de cómputo y memoria será proporcional al tráfico monitorizado.

2.3.2.1.2 A nivel local A nivel local se pueden aplicar soluciones que se basan en modificar la configuración TCP del dispositivo para hacerlo más resistente a ataques. Entre ellas destacan: a) liberar memoria para nuevas conexiones eliminando conexiones defectuosas o reciclando las que llevan más tiempo sin establecerse, b) limitar el número de paquetes SYN que se aceptan cada cierto tiempo, c) reducir el número de retransmisiones de paquetes SYN-ACK, y por tanto un TCB sin establecer se descartará antes, y d) aumentar la capacidad de la cola de conexiones (más memoria), permitiendo así más solicitudes entrantes. Esta última opción tiene un problema, y es que un atacante que puede generar

un ataque SYN es probable que sea capaz de escalarlo hasta saturar el servidor. Una vez alcanzado el máximo de memoria disponible, la cola no podrá expandirse más, y en un dispositivo C2 ese máximo se alcanzaría relativamente rápido. En cuanto a la opción c), conexiones legítimas expuestas a una congestión de red podrían verse afectadas. Además, solo hay una relación lineal entre la reducción del tiempo en el que caduca un TCB y el correspondiente aumento en la tasa de paquetes que el atacante debe producir para seguir atacando el servidor con el mismo impacto. Por lo tanto, las opciones a) y b) parecen más adecuadas para un dispositivo C2.

Además de modificar la configuración TCP, el RFC 4987 presenta otras soluciones alternativas y más sofisticadas entre las que destacan dos: SYN caché y SYN cookies. Su objetivo es reducir el estado que genera la recepción de un paquete SYN.

Una SYN caché [Lemon, 2002] no es más que una estructura auxiliar de memoria para almacenar un subconjunto de los datos que normalmente se almacenan en un TCB ante la recepción de un paquete SYN. Si se recibe un ACK posterior, estos datos se mueven a la cola de conexiones en un TCB completo, si no, se elimina esa entrada de la SYN caché. En la primera versión de la implementación FreeBSD de Lemon, la entrada SYN caché de una conexión sin establecer ocupa 160 bytes, frente a los 736 bytes para un TCB completo.

Por lo general, los sistemas operativos actuales no utilizan toda la memoria necesaria para una conexión completa hasta que la conexión está totalmente establecida. Por ejemplo, en sistemas Linux se utiliza una estructura de datos denominada tcp_request_sock. Esta estructura solo usa 96 bytes para guardar el estado mínimo necesario[3], mientras que la estructura establecida es de 1616 bytes en un kernel de 64 bits. Por lo tanto, el concepto de reducir el estado almacenado que persigue SYN caché se implementa por defecto en la mayoría de sistemas operativos para minimizar posibles ataques.

Los sistemas operativos Linux y FreeBSD incluyen un mecanismo denominado SYN cookies[4], diseñado e implementado originalmente por Bernstein [1997] entre finales de 1996 (SunOS) y principios de 1997 (Linux). En contraste a SYN caché, no utiliza memoria para un SYN entrante. Cuando la cola de conexiones está llena, el servidor responde a cada paquete SYN con un SYN-ACK que contiene la información básica de la conexión codificada como su número de secuencia inicial (ISN). A este número de secuencia –generado con una función criptográfica– se le denomina *cookie*. Cuando se recibe el paquete ACK que finaliza la fase de establecimiento, el servidor puede verificar la cookie, extraer la información contenida y establecer la conexión (TCB). De ahí que las SYN cookies son un compromiso entre memoria y cómputo, ya que no se usa memoria pero se necesita más cómputo para generar la cookie.

La literatura asume que las SYN cookies son seguras, y los sistemas Linux y FreeBSD activan las SYN cookies una vez detectada la saturación de la SYN caché o la cola de conexiones. Su inconveniente es que no todos los datos de un TCB pueden caber en el número de secuencia de 32 bits, por lo que algunas opciones TCP –necesarias para un mayor rendimiento– no pueden utilizarse. En la actualidad existe un método para codificar las opciones restantes en la opción timestamp, pero requiere que el

[3] https://lwn.net/Articles/277146/
[4] http://cr.yp.to/syncookies.html

cliente implemente esa opción. Otro inconveniente es que cuando el servidor utiliza las SYN cookies, responderá con un solo SYN-ACK a la recepción de un SYN (porque la retransmisión requeriría mantener estado), alterando los procedimientos de sincronización TCP descritos en el RFC 793.

Tras analizar la funcionalidad y limitaciones de SYN caché y SYN cookies, Lemon [2002] concluyó que "cuando las SYN cookies están habilitadas, el sistema no descarta ninguna entrada de la cola de conexiones o de la SYN caché, eligiendo enviar una respuesta que contenga una cookie en su lugar. Sin embargo, en la práctica esto lleva a que la SYN caché esté llena de entradas falsas provocadas por una inundación SYN, obligando a que todas las conexiones legítimas sean tratadas por SYN cookies. Esencialmente el sistema termina comportándose como si no hubiera SYN caché". Teniendo en cuenta que la SYN caché también utiliza una función criptográfica para generar la entrada correspondiente, por un coste similar de cómputo y nulo de memoria las SYN cookies parecen ser más adecuadas para un dispositivo C2.

Una alternativa a SYN cookies son las denominadas RST cookies, aunque pueden causar problemas con ciertos sistemas operativos. Su funcionamiento se basa en que el servidor devuelve un SYN-ACK defectuoso al cliente. Un cliente legítimo debe entonces generar un paquete RST notificando al servidor que el paquete recibido está mal formado. En ese momento, el servidor valida al cliente y aceptará sus siguientes conexiones. A diferencia de las SYN cookies, las RST cookies son un compromiso entre cómputo y latencia, ya que no se usa cómputo para generar una cookie pero se necesita más tiempo (latencia) para establecer la conexión. Además esta alternativa tiene un riesgo, y es que el cliente no vuelva a intentar la conexión.

Por último, en sistemas Windows se implementa un mecanismo de protección denominado *SynAttackProtect*, que a partir de Windows Vista está activado de forma predeterminada y no se puede desactivar. *SynAttackProtect* calcula de manera dinámica los umbrales de cuando se considera que un ataque ha comenzado, basándose en el número de núcleos CPU y memoria disponible.

Este enfoque puede inducir a errores, ya que el sistema puede estar usando más CPU o RAM por un funcionamiento a nivel local o por un tipo de tráfico que no sea un ataque de inundación. Cuando el mecanismo determina la presencia de un ataque, el sistema reduce el número de retransmisiones SYN-ACK (opción *c* descrita anteriormente) y limita el número máximo de conexiones (opción *b*). Además, debido a que TCP pasa al estado de ataque basado en el número de núcleos CPU y la cantidad de memoria disponible, los sistemas con más recursos comenzarán a descartar nuevos intentos de conexión más tarde en comparación con sistemas con menos recursos.

En definitiva, en los sistemas que no implementan las SYN cookies (como Mac o Windows), se debe optar por utilizar más memoria RAM, lo que permite más conexiones entrantes, y esperar que no se produzca un ataque. Por lo tanto estos sistemas tienen una limitación ante ataques de inundación y delegan parte de esa mitigación a dispositivos de red de capas superiores, como routers o firewalls. Merece la pena destacar que la inmensa mayoría de estos dispositivos de red se basan en sistemas operativos Linux, que sí implementan SYN cookies[5].

[5] http://en.wikipedia.org/wiki/List_of_router_and_firewall_distributions

2.3.2.2 Mitigaciones en la literatura

La cantidad de trabajos existentes respecto a ataques SYN hace que una revisión completa sea poco pertinente en tanto que redundante, por lo que esta sección se centrará en presentar algunos trabajos representativos de la literatura.

2.3.2.2.1 A nivel de red Sobre el referente de las SYN cookies, existen varios trabajos basados en utilizar dispositivos intermedios con un alto ancho de banda para proteger servidores web de un agotamiento provocado por tráfico ilegítimo.

Casado et al. [2006] proponen que todo el tráfico hacia y desde un sitio web se enrute a través de un middlebox. El middlebox ofrece dos funciones: i) determinar si un paquete TCP enviado al servidor web pertenece a un flujo legítimo, y ii) filtrar el tráfico de direcciones IP falsas. Se demuestra que esta doble funcionalidad se puede implementar sin mantener estado usando lo que denominan Flow cookies, que no es más que una simple extensión de las SYN cookies, y que se implementa dentro de la marca de tiempo TCP (timestamp) de cada paquete de datos saliente desde el servidor. Por lo tanto, esta propuesta depende de un dispositivo de red específico, el cliente debe implementar la opción TCP timestamp, y el coste de cómputo será igual que el de las SYN cookies, sin hacer uso de memoria para las conexiones entrantes.

Mahimkar et al. [2007] también utilizan un middlebox, pero la función de éste es única: evitar ataques de inundación contra el servidor. Cuando se detecta un ataque de inundación, el middlebox se introduce de manera dinámica en la ruta hacia el servidor y todo el tráfico pasa por él. Las SYN cookies permiten al middlebox no mantener estado. Esta propuesta traslada también la aplicación de las SYN cookies del servidor al dispositivo intermedio con el objetivo de reducir la carga de trabajo del servidor.

En vez de depender de dispositivos intermedios con una funcionalidad dedicada, Safa et al. [2008] proponen un mecanismo de mitigación que se aplica en los routers frontera para determinar si el paquete SYN-ACK entrante es válido o no. Esto se logra mediante el mantenimiento de una tabla de coincidencias de los paquetes SYN y SYN-ACK salientes y entrantes. Si un paquete SYN-ACK entrante no es válido, el router libera la conexión de la víctima (paquete RST) y permite aceptar otras solicitudes de conexión legítimas. En este caso no hace falta un dispositivo de red adicional, su operación es transparente para el cliente, y el coste de cómputo y memoria será proporcional al tráfico monitorizado.

Jiang et al. [2013] proponen un modelo de descarte basado en la confianza. El router frontera monitoriza los flujos de red y calcula sus valores de confianza, que se utilizan para la gestión de colas. A mayor confianza, menos probabilidad de descarte. Al igual que la propuesta anterior, no hace falta un dispositivo de red adicional, su operación es transparente para el cliente, y el coste de cómputo y memoria será proporcional al tráfico monitorizado.

Otra propuesta basada en el descarte la presentan Chang et al. [2010]. Los ataques se neutralizan controlando las tasas de descarte en el nivel de aplicación mediante el uso de una prioridad diferencial de paquetes. Dependiendo del número de descartes y la congestión, se priorizan ciertos flujos. Por lo tanto, la mitigación se realiza a nivel de aplicación en lugar de a nivel de red, lo que reduce la precisión

en la detección a costa de un menor consumo de recursos en el router. Al igual que la propuesta anterior, no hace falta un dispositivo de red adicional, su funcionamiento es también transparente para el cliente, y el coste de cómputo y memoria es proporcional al tráfico de red monitorizado.

2.3.2.2.2 A nivel local Kim et al. [2008] utilizan un mecanismo basado en una lista de clientes previos. En caso de ataque, los paquetes de solicitud de conexión (SYN) se clasifican en dos tipos: un paquete legítimo y un paquete de ataque potencial, dependiendo de si existe la dirección IP de origen en una lista de clientes previos o no. El paquete legítimo se procesa en la cola de conexiones, y el paquete de ataque potencial se procesa con el método SYN cookies. La desventaja de este modelo es que un atacante podría usar una primera conexión legítima para colocar su IP en esa lista y luego saltarse el método de prevención.

Kavisankar y Chellappan [2011] proponen un método sencillo para prevenir ataques SYN sin el computo de las SYN cookies. El mecanismo se basa en que el cliente cambie su tamaño de ventana en el paquete ACK que cierra el establecimiento de la conexión. Esta propuesta no tiene un coste de cómputo pero hay que almacenar la conexión para comprobar que la ventana ha cambiado, y por lo tanto seguiría siendo vulnerable a una inundación.

Con un objetivo parecido, Lee y Thing [2004] propusieron un método de cambio de puerto que se puede utilizar para dispositivos embebidos. Con este método, el número de puerto TCP utilizado por el servidor varía como una función del tiempo y un secreto compartido entre el servidor y el cliente. El método simplifica la detección y filtrado de paquetes falsos. Aunque el coste de cómputo es insignificante, esta propuesta requiere que tanto los clientes como el servidor negocien el secreto e implementen el método de salto de puerto, y por tanto no funcionaría si un dispositivo proporciona un servicio estándar, como por ejemplo un servidor web. Existe también una variación más reciente de este trabajo que propone cambiar no solo el puerto, sino la IP del servidor [Luo et al., 2015]. Este tipo de cambios requerirían una sincronización efectiva entre clientes y el servidor, y podrían causar un conflicto de direcciones IP.

En vez de descartar conexiones, Zuquete [2002] propone modificar uno de los mecanismos de mitigación presentados antes, las SYN cookies, para hacerlas más eficientes. El problema que presentan las SYN cookies es el hecho de tener que descartar opciones TCP si no se utiliza la opción de timestamp. La idea es usar una conexión TCP simultánea con el fin de que los clientes manden las opciones y las cookies al servidor atacado. Esta conexión simultánea presenta un problema: a pesar de que algunos sistemas operativos son capaces de soportarlo, después de cierto tiempo fallan al recibir múltiples paquetes. Por lo tanto, el cliente debe implementar la conexión simultánea para enviar las opciones TCP y el coste de cómputo es el mismo que las SYN cookies.

Crowcroft et al. [2007] dan una vuelta de tuerca a las SYN cookies y proponen un mecanismo que valide que el cliente tiene voluntad de conectarse al servidor. Proponen una prueba de trabajo (*Proof-of-Work*) basada en introducir una latencia en la respuesta del cliente para cerrar la fase de establecimiento (ACK) e incluyen esos parámetros en una SYN cookie modificada con el fin de que el servidor no mantenga estado. Una prueba de trabajo es una técnica que resuelve problemas de equidad o

abuso. Antes de que el proveedor del servicio acepte la conexión, el solicitante tiene que completar un trabajo para demostrar su compromiso. El trabajo debe ser moderadamente duro en la parte solicitante, pero fácil de comprobar para el proveedor. Los clientes pueden tener diferentes capacidades de cómputo, y la selección de una prueba sin conocer esas capacidades puede dañar a clientes legítimos con menos recursos. Los ataques de inundación se caracterizan por solicitudes rápidas, y añadiendo una latencia al cliente para terminar la fase de establecimiento podría hacer los ataques DoS menos atractivos, sin dejar de ser una solución aplicable a cualquier tipo de cliente.

A pesar de ser una buena idea, la propuesta presenta dos problemas: i) clientes con una buena conducta se ven forzados a esperar un tiempo aleatorio para responder al servidor, reduciendo su rendimiento, y ii) el mecanismo encargado de verificar la espera solo detecta que se ha superado el tiempo exigido, lo que podría llevar a ataques de inundación contra el mecanismo. En definitiva, esta propuesta necesita que el cliente implemente esta nueva opción TCP para esperar el tiempo requerido por el servidor, lo que limita su escalabilidad. En cuanto al coste de cómputo, es igual al de las SYN cookies.

Por último, existen múltiples trabajos que aplican políticas de descarte [Ming, 2009]. Al-Duwairi y Manimaran [2006] analizan la reacción del cliente a la pérdida del paquete SYN. La idea es descartar intencionalmente el primer paquete SYN de cada petición de conexión. Un paquete SYN posterior del mismo cliente se acepta solo si se adhiere al tiempo de espera que marca TCP para un reintento de conexión. Si no se adhiere, significa que es un atacante. Este mecanismo es un compromiso entre memoria y latencia. Al utilizar un espacio de memoria mínimo para guardar la IP y el tiempo de recepción del paquete SYN, se necesita más tiempo para establecer la conexión. Esta alternativa tiene un riesgo, y es que el cliente no vuelva a intentar la conexión.

Jamali y Shaker [2014] proponen en cambio que la defensa ante un ataque de inundación se formule como un problema de optimización. Para ello utilizan un algoritmo de enjambre de partículas (PSO) que descarta conexiones en estado *SYN-RECEIVED* cuyo índice de supervivencia no sea óptimo. Esto reduce el volumen de conexiones falsas en la cola de entrada, pero también descarta tráfico legítimo (falsos positivos). Los ataques SYN tienen por objetivo interrumpir conexiones legítimas, por lo que su solución no debe tener el efecto de cerrar innecesariamente conexiones válidas. De lo contrario, solo se ayudaría al atacante en su objetivo. Además, esta propuesta tiene un coste de cómputo significativo para procesar el algoritmo PSO. En un dispositivo C2, descartar entradas mediante algoritmos complejos no parece una solución apropiada para una cola de conexiones tan reducida.

2.3.3 Conclusión

La inundación SYN es el ataque DoS más antiguo que existe, y aún así es un ataque relativamente común hoy en día debido a su facilidad de ejecución.

Tanto las soluciones a nivel de red como las de nivel local funcionan, y por lo general no interfieren cuando se utilizan en combinación. Ya que el objetivo principal de la inundación SYN es saturar la cola de conexiones de un servidor, éste debería aplicar algún tipo de defensa, sin importar sus limitaciones.

En caso de utilizar un balanceo de carga DSR, las soluciones a nivel local cobran mayor importancia si cabe. De esta forma, las técnicas de red se podrían utilizar como segunda línea de defensa.

Tras analizar el estado del arte, se observa que la mayoría de defensas a nivel local se aplican en dispositivos sin las limitaciones descritas en el RFC 7228. En este punto se debe valorar si es conveniente diseñar un nuevo mecanismo de mitigación ajustado a las limitaciones de un dispositivo C2, o por el contrario es más oportuno aplicar soluciones ya existentes.

Dentro del Web of Things Interest Group existe un grupo de trabajo centrado en la seguridad de los dispositivos de recursos limitados que formarán parte de la Web of Things –denominado *Security, Privacy and Resilience*– que defiende una línea de trabajo específica: "el trabajo no se empieza en una página vacía - existen patrones, protocolos (estándar), mecanismos o componentes que pueden ser reutilizados (con o sin adaptación)".

Siguiendo esta línea de trabajo, en vez de diseñar un mecanismo ad hoc para dispositivos C2, se opta por soluciones existentes en sistemas operativos comunes. Las propuestas recogidas en la literatura presentan mecanismos que en muchas ocasiones dependen de la cooperación del cliente, y por tanto son poco escalables, y cuya eficacia no está analizada ni contrastada al mismo nivel que los mecanismos de práctica común.

En concreto se valoran los dos mecanismos –documentados en el RFC 4987– que se aplican cuando la cola de conexiones se llena, algo que debido a la memoria tan reducida de un dispositivo C2 puede ocurrir incluso con un pequeño número de paquetes SYN. El mecanismo SYN cookies –habilitado por defecto en sistemas Linux y FreeBSD– resulta una opción interesante ya que se trata del único mecanismo que no hace uso de memoria para las conexiones entrantes. Su rendimiento debe compararse con la renovación de las conexiones que llevan más tiempo sin establecerse (opción *a)* del apartado 2.3.2.1.2), y determinar así su aplicabilidad en dispositivos C2.

CAPÍTULO

3

Diseño e implementación

Un sistema IoT debería tener las mismas capacidades que el resto de sistemas de Internet, tales como disponibilidad, escalabilidad y seguridad. Implementar y evaluar arquitecturas de balanceo de carga en dispositivos IoT permite avanzar en la consecución de ese objetivo.

En este capítulo se describe el diseño y la implementación de una arquitectura distribuida de dispositivos limitados con cierta tolerancia a ataques. La primera parte comprende el balanceo de carga con configuración DSR (*escalabilidad*), y la segunda parte se centra en la aplicación del mecanismo SYN cookies a dispositivos C2 (*seguridad*).

3.1 Balanceo de carga

Un clúster de balanceo de carga es un tipo de arquitectura formada por un conjunto de dispositivos independientes e interconectados que trabajan como si de uno solo se tratase. Dividiendo la carga de trabajo (conexiones) entre los dispositivos se puede mejorar el rendimiento de un servicio.

La arquitectura básica de un clúster de balanceo TCP está formada por:

- Balanceador de carga: es el dispositivo que recibe las conexiones TCP y las distribuye entre un conjunto de servidores, y por tanto los clientes consideran que el servicio lo proporciona un solo dispositivo.
- Servidores: el conjunto de servidores que se encargan de atender las solicitudes que recibe un servicio TCP.
- Servicio común: es el servicio TCP que proporciona el clúster de servidores. Los servidores deben ofrecer el mismo servicio.

De las configuraciones de balanceo analizadas en el capítulo anterior, DSR es la más eficiente al permitir a los servidores procesar y enviar paquetes directamente a los clientes en lugar de dirigir esos paquetes salientes a través del balanceador. No obstante, los servidores deben estar físicamente conectados al mismo segmento de red que el balanceador.

Esta sección analiza en primer lugar cómo funciona este esquema de balanceo de capa 2. A continuación se presentan los dispositivos C2 utilizados, y cómo se han implementado las funcionalidades y requisitos necesarios de este esquema de balanceo tanto en el balanceador como en los servidores. Se utilizará como referencia la herramienta software que permite gestionar el balanceo de carga en sistemas Linux (*Linux Virtual Server - LVS*).

3.1.1 Análisis del esquema DSR

El hecho de que los paquetes de retorno (del servidor al cliente) no pasen por el balanceador significa que se puede conseguir una mayor escalabilidad, ya que se pueden ir agregando servidores sin la carga adicional en el balanceador para distribuir esos paquetes de retorno. Esto significa también que se puede obtener un mayor rendimiento de red que con los esquemas NAT, especialmente si los datos de retorno son el mayor volumen de tráfico.

Si bien existen ventajas de utilizar este esquema de balanceo, también hay ciertos inconvenientes. El problema más común con el esquema DSR tiene que ver con el protocolo ARP. Este protocolo permite determinar la dirección física (MAC) asociada a una dirección IP para que la capa de enlace de datos pueda proceder con la transmisión de paquetes IP. Para asociar direcciones físicas con las direcciones lógicas (IP), el protocolo ARP interroga a los dispositivos de la red para averiguar sus direcciones físicas y luego crea una tabla de búsqueda con el resultado. Antes de iniciar la comunicación, el dispositivo consulta su tabla de búsqueda. Si la dirección requerida no se encuentra en la tabla, el protocolo ARP envía una solicitud a la red preguntando quién tiene esa dirección IP. Todos los dispositivos en la red comparan esta dirección lógica con la suya. Si alguno de ellos tiene esa dirección, responderá al solicitante indicándole su dirección MAC. El dispositivo almacenará el par de direcciones en la tabla de búsqueda, y a continuación, podrá establecerse la comunicación. El problema de ARP en una configuración DSR es que solo un dispositivo debe responder a una solicitud ARP.

Cuando un cliente se conecta a un clúster, utiliza una conexión TCP a una dirección IP. Al configurar un clúster mediante DSR se debe utilizar una IP virtual (VIP), que no es más que una dirección IP que no corresponde a una interfaz de red física real. Esta dirección VIP sirve como la única dirección externa para todo el clúster. El balanceador de carga recibe los paquetes de entrada a través de esa VIP, que representa el servicio virtual del clúster, reescribe la dirección MAC de destino del paquete a la dirección MAC de uno de los servidores, y vuelve a poner el paquete en la red. Este paquete está sin modificar en la capa de red, y por tanto todos los dispositivos que forman el clúster (balanceador y servidores) deben estar conectados al mismo segmento de red y tener habilitada la VIP, además de sus direcciones IP reales.

Al igual que cualquier otra dirección IP, la dirección VIP del clúster debe estar asociada a una dirección MAC, es decir, a un único dispositivo. Sin embargo, puesto que tanto el balanceador como los servidores tienen configurada la misma VIP, la petición ARP será procesada por todos los dispositivos del clúster. Si cualquiera de los servidores contestase a esa solicitud ARP, la dirección del clúster se

asociaría directamente al primer servidor en responder y por tanto, las solicitudes llegarían a ese servidor, sin pasar por el balanceador, acabando así con el propósito del balanceo de carga.

Para resolver este problema las solicitudes entrantes solo deben asociarse a la dirección MAC del balanceador, que procesará adecuadamente las solicitudes y las enviará al grupo de servidores. La implementación del balanceo de carga en Linux (*LVS*) aplica una configuración adicional para desactivar las respuestas ARP en la interfaz de red (virtual) configurada con la VIP para evitar que los servidores respondan a las solicitudes ARP para esa dirección VIP. De esta forma, el único que responderá a la solicitud ARP para la dirección VIP será el balanceador.

A continuación se muestra un ejemplo de cómo se intercambian los paquetes en un esquema DSR. En este caso el cliente está conectado al mismo segmento de red que el clúster.

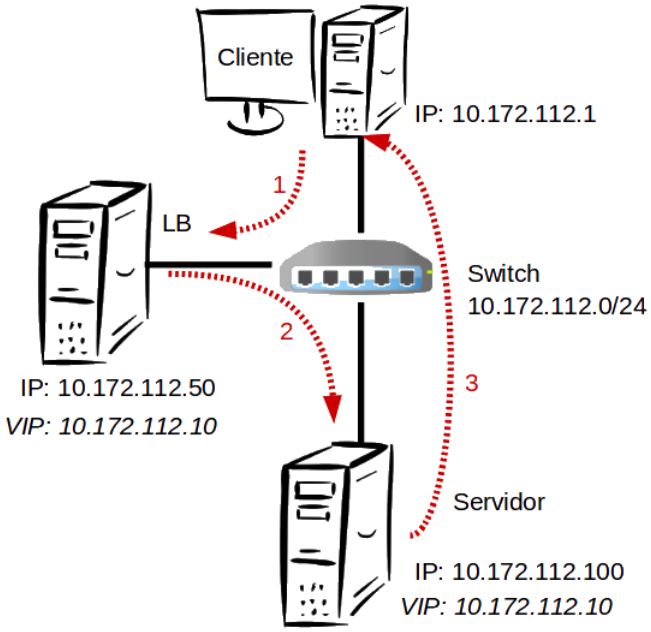


Figura 3.1 Ejemplo DSR

Tabla 3.1 Direcciones IP y MAC

	IP	**MAC**
Cliente	10.172.112.1	78:2b:cb:e4:ad:57
Balanceador	10.172.112.50	00:02:f7:f2:19:3a
Servidor	10.172.112.100	00:02:f7:f2:30:f1
Clúster	10.172.112.10	00:02:f7:f2:19:3a

La tabla 3.2 muestra las direcciones IP y MAC resultantes de un intercambio con el esquema DSR.

Como se puede observar, la configuración DSR realiza un balanceo en tres pasos. En primer lugar, el cliente tiene guardada en su tabla ARP la dirección MAC del balanceador asociada a la VIP, y por tanto

Tabla 3.2 Resultado del balanceo

Tráfico	IP origen	MAC origen	IP destino	MAC destino
Cliente → Balanceador	10.172.112.1	78:2b:cb:e4:ad:57	10.172.112.10	00:02:f7:f2:19:3a
Balanceador → Servidor	10.172.112.1	78:2b:cb:e4:ad:57	10.172.112.10	00:02:f7:f2:30:f1
Servidor → Cliente	10.172.112.10	00:02:f7:f2:30:f1	10.172.112.1	78:2b:cb:e4:ad:57

el paquete llega al balanceador. A continuación, en vez de cambiar la IP de destino por la IP del servidor (10.172.112.100), el balanceador mantiene la misma dirección VIP (10.172.112.10) y solo reescribe la dirección MAC de destino por la del servidor correspondiente. El servidor recibe un paquete con una dirección IP de destino que no es la suya (10.172.112.100), pero como también acepta tráfico para la VIP (10.172.112.10), procesa el paquete. Por último, el servidor tiene que responder al cliente. En lugar de enviar la respuesta a través del balanceador, se proporcionan las direcciones IP y MAC del cliente en el paquete de retorno.

No obstante, si solo se sobreescribe la dirección MAC de destino cuando el balanceador envía el paquete al servidor, el direccionamiento de red de capa 2 no se comporta como debería. Si los equipos del clúster están conectados a una red conmutada (a través de un *switch*) en lugar de a una red de medio compartido (a través de un *hub* o un punto de acceso Wi-Fi), el uso de una misma dirección MAC desde puertos diferentes podría crear un conflicto. Los conmutadores poseen la capacidad de aprender y almacenar las direcciones de red de capa 2 (direcciones MAC) de los dispositivos conectados a cada uno de sus puertos, y por tanto esperan ver direcciones MAC de origen únicas en cada puerto. En el ejemplo anterior, el paquete del cliente entra por un puerto del conmutador, al que se le asocia la dirección MAC del cliente. Cuando el balanceador remite el paquete al servidor, se utiliza la misma MAC de origen, por lo que el conmutador asociará también la dirección MAC a ese puerto. Al enviar el paquete del servidor al cliente, el conmutador podría enviar el paquete a través del puerto incorrecto.

Para solucionar este problema es posible utilizar configuraciones avanzadas en el conmutador, como por ejemplo prevenir el aprendizaje de direcciones MAC. Como resultado, los paquetes de entrada al clúster serían entregados a todos los puertos del conmutador. Para evitar tener que realizar configuraciones adicionales en el conmutador, existe una opción sencilla a nivel de balanceador. El paquete entrará en el conmutador, y se enviará al balanceador de carga. Una vez allí, el balanceador actualizará ambas direcciones MAC, de origen (la suya) y destino (la del servidor), antes de enviar el paquete de nuevo al conmutador [Arregoces y Portolani, 2003]. Un conmutador solo entiende de direcciones MAC, por lo que reenviará el paquete al servidor correcto. Los balanceadores DSR comunes, como los implementados en Linux, funcionan así.

La tabla 3.3 muestra las direcciones IP y MAC resultantes de una comunicación utilizando esta solución.

Si el cliente estuviera en una red externa al clúster, la dirección MAC de destino del último paso se tendría que establecer a la dirección de la puerta de enlace por donde entró el paquete del cliente.

Tabla 3.3 Resultado del balanceo en una red conmutada

Tráfico	IP origen	MAC origen	IP destino	MAC destino
Cliente → Balanceador	10.172.112.1	78:2b:cb:e4:ad:57	10.172.112.10	00:02:f7:f2:19:3a
Balanceador → Servidor	10.172.112.1	00:02:f7:f2:19:3a	10.172.112.10	00:02:f7:f2:30:f1
Servidor → Cliente	10.172.112.10	00:02:f7:f2:30:f1	10.172.112.1	78:2b:cb:e4:ad:57

En los próximos apartados se explicará cómo se han implementado los requisitos funcionales necesarios tanto en el balanceador como en los servidores para desplegar el esquema de balanceo DSR en dispositivos C2.

3.1.2 Recursos utilizados

Antes de entrar en los detalles de la implementación tanto del balanceador como de los servidores que forman el clúster, es necesario explicar las características del dispositivo C2 empleado y la pila TCP/IP utilizada.

3.1.2.1 Dispositivo C2: *LPC1768*

Tanto el balanceador como los servidores se implementan en el LPC1768 de NXP. El LPC1768 es un microcontrolador Cortex-M3 de 32-bits que funciona a 96MHz, tiene 512KB de memoria de código (ROM), y 64KB de memoria de datos (RAM). En cuanto a la memoria RAM, comentar que en realidad se divide en tres bancos: uno de 32KB de propósito general y otros dos de 16KB para los periféricos, como Ethernet. El LPC1768 incluye un controlador de Ethernet 10/100 con interfaz RMII y acceso directo a memoria. De esta forma, para conectar el LPC1768 a la red solo hace falta un conector RJ45. Se ha utilizado la siguiente placa de expansión que incluye multitud de periféricos, entre ellos el conector RJ45.

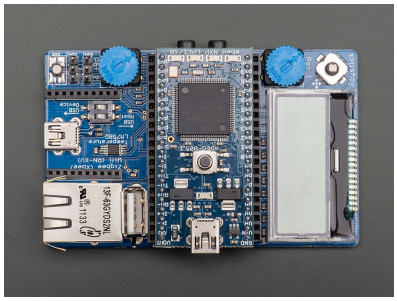


Figura 3.2 Placa de expansión con LPC1768

Si bien es cierto que la capacidad de las memorias son superiores a lo definido en el RFC 7228 para dispositivos C2 (64KB de memoria RAM en vez de 50, y 512KB de ROM en vez de 250), cabe destacar que i) la memoria usada por los programas desarrollados nunca ha superado las especificaciones del RFC 7228 para un dispositivo C2 (36 KB de memoria de código y 43 KB de RAM en el caso del

balanceador, y 62 KB de memoria de código y 47 KB de RAM para el servidor), ii) la capacidad de memoria destinada para procesar paquetes de red es como máximo de 32 KB, y iii) en el RFC 7228 se define textualmente que "existen dispositivos limitados con capacidades significativamente superiores a dispositivos clase 2. Son menos exigentes desde el punto de vista de desarrollo de normas, ya que pueden en gran parte utilizar los protocolos existentes sin cambios. En el presente documento, por tanto, no se hace ningún intento de definir clases más allá de la clase 2". EL RFC 7228 presenta unos valores de memoria orientativos, y los valores del LPC1768 no pueden considerarse significativamente superiores, sobre todo en lo que respecta a memoria RAM donde es similar.

Cabe destacar que el LPC1768 utilizado incorpora por defecto un sistema operativo en tiempo real (RTOS), denominado mbed OS y desarrollado por la propia compañía ARM. Utilizar un RTOS ha sido muy útil especialmente en el servidor, ya que se han podido crear procesos independientes para gestionar las conexiones entrantes.

Por último, mbed OS ofrece facilidades para la comunicación de red a través de la pila TCP/IP lwIP en su versión 1.4.1[1]. A continuación se presenta brevemente esta pila TCP/IP.

3.1.2.2 Pila TCP/IP: *lwIP*

lwIP es la abreviatura de lightweight IP, y como su nombre indica, se trata de una implementación del conjunto de protocolos de Internet que intenta reducir en la medida de lo posible el uso de recursos de memoria y procesamiento. Adam Dunkels desarrolló originalmente lwIP en el Instituto Sueco de Ciencias de la Computación (SICS) con el objetivo de que los sistemas embebidos de recursos limitados –como los microcontroladores– tuvieran acceso a Internet. Para utilizar lwIP solo hace falta tener espacio disponible en memoria RAM de aproximadamente 20KB, y de 40KB en memoria de código.

lwIP soporta los siguientes protocolos:

- ARP (Address Resolution Protocol)
- IP (Internet Protocol) v4 y v6
- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)
- DNS (Domain Name Server)
- SNMP (Simple Network Management Protocol)
- DHCP (Dynamic Host Configuration Protocol)
- ICMP (Internet Control Message Protocol)
- IGMP (Internet Group Management Protocol)

[1] http://git.savannah.gnu.org/cgit/lwip.git/

- PPP (Point to Point Protocol)
- PPPoE (Point to Point Protocol over Ethernet)

lwIP es la pila TCP/IP para microcontroladores con más referencias, se utiliza en muchos productos comerciales, y es la única que sigue siendo mejorada con un foro de usuarios activos. Actualmente es mantenida por un grupo de voluntarios en GNU Savannah, la forja de proyectos de la Free Software Foundation.

Para utilizar lwIP se tienen que realizar ciertas configuraciones, como definir si se utiliza un sistema operativo o no, implementar un controlador de red –el controlador de Ethernet para el LPC1768– y disponer de un entorno de trabajo. La aplicación tiene que inicializar la pila y llamar regularmente a rutinas de mantenimiento. Además, para conseguir un entorno de trabajo se necesita un compilador, algunos scripts de inicialización del hardware y algunas funciones básicas para la depuración. Todo esto lo proporciona mbed OS.

Por otra parte, lwIP ofrece tres API diseñadas para diferentes propósitos:

- API Raw es el núcleo de la API de lwIP. Esta API tiene como objetivo proporcionar el mejor rendimiento usando un tamaño de código mínimo. El principal inconveniente de esta API es que gestiona los eventos asíncronos utilizando retornos de llamada (funciones callback), haciendo más complejo el diseño de la aplicación.
- API Netconn es una API secuencial integrada encima de la API Raw. Permite operaciones con varios procesos (capacidad multi-hilo) y por lo tanto requiere un sistema operativo. Es más fácil de usar que el API Raw a expensas de rendimientos más bajos y un mayor uso de memoria. En este caso, esta API tiene un proceso que es el encargado de recibir la trama Ethernet del controlador y transmitirla a la función de procesado, que hace uso de la API Raw.
- API BSD Socket es una implementación similar al API de sockets Berkeley (Posix/BSD) construida encima de la API Netconn. Se utiliza cuando es necesaria una portabilidad de un sistema que utiliza BSD, y comparte el mismo rendimiento que la API Netconn.

En este caso se ha utilizado la API BSD ya que es necesario trabajar con el sistema operativo y con su capacidad de gestionar varios procesos, además permite generar un código más legible que emula a los sistemas Unix. El API desarrollado para los sockets BSD o Berkeley es el estándar de facto para el desarrollo de aplicaciones de red en la mayoría de sistemas operativos y lenguajes de programación.

3.1.3 Implementación del balanceador

Antes de analizar cómo se han implementado las funciones del balanceador, es necesario explicar la organización de memoria utilizada. Como se ha expuesto en la sección anterior, el microcontrolador LPC1768 tiene 3 bancos de memoria RAM. Uno de propósito general de 32KB, y dos adicionales de 16KB para el resto de periféricos. En primer lugar, se ha comprobado en qué banco se guardan las

estructuras de datos del controlador de Ethernet (EMAC). En concreto, están en el primer banco de memoria adicional de 16KB, y según se describe en la hoja de datos (*datasheet*), los descriptores y las estructuras de datos de recepción pueden estar solo en esos bloques de memoria RAM adicionales.

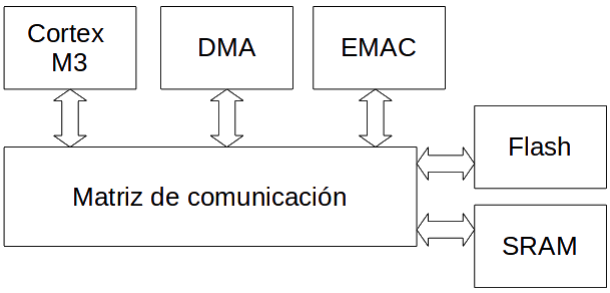


Figura 3.3 Diagrama de bloques conceptual del LPC1768

Para conocer un poco mejor el flujo de un paquete de datos desde memoria hasta que se envía, se presenta una breve explicación sobre el funcionamiento del EMAC. Esta explicación ayudará a entender las limitaciones de memoria de un dispositivo C2 para procesar múltiples conexiones. Durante su funcionamiento, un microcontrolador necesita leer y escribir datos a memoria. Puede leer datos de un periférico y escribirlos en la RAM, o quizá necesite enviar tramas de datos a través de un controlador del medio, como Ethernet. Una vez más se necesita leer esos datos de la memoria RAM y escribir al registro de datos del EMAC. Cuando esta transferencia de datos se hace utilizando el procesador, se pierde una cantidad significativa de tiempo de procesamiento. Para evitar la ocupación de la CPU, los microcontroladores más avanzados tienen un módulo de acceso directo a memoria (DMA por sus siglas en inglés). Como su nombre indica, DMA administra las transferencias de datos entre ubicaciones de memoria sin necesidad de utilizar la CPU.

Los descriptores de transmisión y recepción son utilizados por DMA. En estos descriptores se guarda la dirección de memoria desde donde DMA lee y escribe la trama Ethernet mientras transmite y recibe, respectivamente. Aparte de la ubicación de memoria, otros parámetros importantes que son requeridos por DMA para procesar una trama también se programan en esos descriptores. Una vez que toda la información está programada, se traspasa el control de los descriptores a DMA. Cuando se completa la transmisión o recepción de una trama, DMA actualiza el estado en los descriptores y los asigna de nuevo al controlador EMAC. Así, el flujo de una trama Ethernet sigue este orden:

Transmisión: RAM–>DMA–>MAC–>Capa física
Recepción: Capa física–>MAC–>DMA–>RAM

La implementación para el LPC1768 utiliza dos variables para configurar el número de descriptores a utilizar. LPC_NUM_BUFF_RXDESCS especifica el número de descriptores a utilizar en tiempo de ejecución para recibir paquetes. A más descriptores, se pueden almacenar más datos de Ethernet sin descartar paquetes. Por otro lado, LPC_NUM_BUFF_TXDESCS especifica el número de descriptores a utilizar en tiempo de ejecución para paquetes de transmisión. A más descriptores, la aplicación puede enviar más datos sin que la cola de transmisión se llene.

En lwIP, las estructuras de datos de recepción necesarias para recibir los paquetes están pre-asignadas al tamaño máximo esperado de un paquete Ethernet, y cada una se asigna a un descriptor del controlador de Ethernet. Una vez que la estructura de datos deja las funciones del controlador de Ethernet, el descriptor original que estaba asociado con esa estructura de datos queda libre. El controlador intentará reservar memoria y asignar una nueva estructura de datos para el descriptor antes de devolver el paquete recibido. Si no hay memoria disponible para la nueva estructura de datos, el descriptor permanece sin asignar y se hará un nuevo intento de asignación cuando se reciba el próximo paquete.

Como ya se ha explicado, cuantos más descriptores se configuren, más paquetes se podrán procesar a la vez, pero a coste de más recursos de memoria, ya que cada descriptor deberá tener asociado un espacio de memoria equivalente al tamaño máximo de un paquete Ethernet. Con la configuración por defecto, solo se utilizarían 16KB. El objetivo es aprovechar el máximo tamaño de memoria disponible para gestionar paquetes de red. Por lo tanto, se ha cambiado la configuración de memoria del LPC1768 para crear un único banco de memoria que contenga los dos bancos adicionales. Esta asignación es posible porque los dos bancos de memoria están contiguos. A continuación se ha definido que los datos de Ethernet se gestionen en este nuevo banco de memoria de 32KB.

Una vez explicada la asignación de memoria, se especifica cómo implementar las características funcionales del balanceador. A grandes rasgos, la primera vez que llega un paquete (SYN), se utiliza un algoritmo de planificación para la elección del servidor de destino, y el balanceador guarda la información de la conexión en una estructura de datos específica. Cuando llega un paquete que no tiene activo el flag SYN, se determina si ya existe la conexión en la estructura de datos, y de ser así, se envía el paquete al servidor que está procesando la conexión.

Por lo tanto, desplegar un balanceador de carga con configuración DSR implica una configuración en cuatro pasos:

1. Crear o configurar una interfaz de red con la VIP del clúster.
2. Añadir los servidores que forman el clúster.
3. Implementar la funcionalidad del balanceo DSR.
4. Crear y mantener la estructuras de datos para gestionar una conexión balanceada.

3.1.3.1 Configuración de la VIP

El primer paso es configurar la interfaz de red para aceptar tráfico dirigido al clúster. La pila de red lwIP ha definido un conjunto de prototipos de funciones para interactuar con el hardware de red, entre ellos la creación de una interfaz de red. Para crear una interfaz de red, el usuario debe especificar una dirección IP para la interfaz, su máscara de red y la puerta de enlace. Estos valores se pueden establecer de manera estática o mediante DHCP. Además, se especifica la función que se llamará cuando se reciba un nuevo paquete.

Como se ha explicado en la sección 3.1.1, cada dispositivo del clúster debería tener configurada su propia IP además de la VIP, ambas en la misma subred. En teoría, una interfaz de red solo puede tener

una dirección IP asignada a ella, aunque en realidad se pueden configurar múltiples direcciones IP para la misma interfaz de red. A cada una de estas direcciones se le denomina alias IP. La principal ventaja de utilizar este concepto es que no es necesario tener un adaptador físico conectado a cada IP, se pueden crear múltiples interfaces virtuales con distintas direcciones IP (*alias*) utilizando una sola interfaz de red.

lwIP permite crear varias interfaces de red lógicas. No obstante, la versión de lwIP utilizada (1.4.0) presenta problemas de encaminamiento si existen dos interfaces de red configuradas en la misma subred[2]. El encaminamiento básico IPv4 se basa únicamente en la dirección de destino y no en la dirección de origen. Como tal, si existen dos interfaces de red en la misma subred, el algoritmo de encaminamiento siempre enviará los paquetes a través de la primera coincidencia. Esto podría provocar errores en una configuración DSR.

Por lo tanto, solo se ha configurado una interfaz de red con la dirección IP estática que representa la VIP del clúster, y el número de puerto del servicio virtual (el puerto donde los servidores esperan recibir el tráfico TCP). La IP real se utiliza para labores de gestión del clúster, funcionalidad que no es necesaria para el análisis de rendimiento que se pretende en este trabajo. De esta forma, el balanceador está configurado con la IP del clúster y podrá responder a los paquetes ARP, ya que la funcionalidad ARP para esa interfaz de red está habilitada (#define LWIP_ARP 1). Al recibir una solicitud ARP del cliente dirigida a la dirección VIP, el balanceador responderá con su dirección MAC, por lo que los siguientes paquetes del cliente irán destinados al balanceador. Cabe destacar que el balanceador no guardará en memoria ninguna asociación IP-MAC de los clientes, ésto solo es necesario en los servidores para poder procesar las conexiones del cliente. El balanceador únicamente remitirá los paquetes de entrada al servidor correspondiente.

Esta característica es la principal diferencia respecto a un balanceador de carga DSR convencional. Poder configurar solo la dirección VIP implica que los servidores no deben responder a los paquetes ARP de los clientes, pero de alguna forma deben guardar las direcciones MAC de esos clientes para enviarles paquetes. La sección correspondiente a la implementación de los servidores explicará cómo se ha abordado este problema.

3.1.3.2 Agregar servidores al clúster

Una vez configurada la interfaz de red, es necesario determinar los servidores que formarán parte del clúster. La escalabilidad de un servicio se logra mediante la adición o eliminación transparente de servidores al clúster. En una configuración DSR al uso, el balanceador especificará –de forma manual a través del usuario– las direcciones IP reales de los servidores que se añadirán al clúster. Cuando llegue un paquete SYN, el balanceador determinará el servidor a utilizar a través del algoritmo de planificación. A continuación, buscará la dirección MAC de esa IP en la caché para realizar la transmisión, y si no existe, utilizará su dirección IP real para enviar una solicitud ARP. Con la limitación de una interfaz de red, esta configuración no es posible. Los servidores también tendrán una única dirección IP, y será la VIP del clúster, por lo tanto, en lugar de agregar direcciones IP, se utiliza una lista en la que se guardan

[2] https://savannah.nongnu.org/task/index.php?13397

las direcciones MAC de esos servidores. El balanceador ya no tendrá que hacer ARP a los servidores, simplemente recorrerá esa lista para remitir las conexiones al servidor correspondiente.

3.1.3.3 Funcionalidad del balanceador

Posteriormente, hay que implementar la funcionalidad del balanceador en la rutina encargada de procesar los paquetes Ethernet recibidos. Esta rutina solo procesa los paquetes cuya dirección MAC de destino coincide con la propia dirección MAC del dispositivo, en este caso el balanceador, y por tanto dirigidos al clúster. En esta rutina se utiliza un puntero a la carga útil del paquete, que comienza con un encabezado Ethernet. A continuación se analiza el campo *EtherType* dentro del encabezado, utilizado para indicar qué protocolo se encapsula en la carga útil del paquete Ethernet. Si se trata de un paquete IPv4 (EtherType=0x0800), se retira la cabecera Ethernet y se pasa el paquete resultante a la rutina de procesado de la capa IP. En vez de pasar directamente el paquete a la capa IP, se ha introducido una condición para procesar los paquetes a balancear. Si el campo *Protocol* del encabezado IP –indica el protocolo utilizado en la parte de datos del paquete– es TCP (Protocol=0x06), y el número de puerto de destino de la cabecera TCP corresponde con el puerto del servicio virtual, el paquete se envía a la rutina que se ha creado para realizar el balanceo. En caso contrario, el paquete se envía a la rutina de procesado de la capa IP.

Cuando se recibe un paquete IP en la rutina de balanceo, se utiliza un puntero al encabezado TCP para comprobar si el flag SYN del paquete está activo o no. En caso de un paquete SYN, se comprueba si hay espacio disponible para una nueva entrada en la estructura de datos creada para guardar la información necesaria para gestionar las conexiones balanceadas. En el próximo apartado se explicará esta estructura en detalle. Si no hay espacio libre en la estructura, el paquete se descarta y se hará un nuevo intento con el próximo paquete SYN recibido. Si por el contrario hay espacio disponible, se elige un servidor del clúster (dirección MAC de la lista de servidores antes explicada) para procesar la conexión mediante un algoritmo de planificación. En este caso se utiliza el más común y sencillo, Round-Robin. En su forma matemática, se selecciona el servidor con la ecuación $i = (i + 1) \% n$, donde n es el número de servidores e i indica el servidor seleccionado la última vez (se inicializa con un valor de -1).

3.1.3.4 Estructura de datos para las conexiones balanceadas

Una vez seleccionado el servidor, hay que guardar ciertos datos de la conexión para poder gestionarla. De lo contrario, al recibir un nuevo paquete para la misma conexión no se podría determinar el servidor que la está procesando. En primer lugar, se deben valorar las diferentes estructuras de datos aplicables. Las estructuras de datos más utilizadas son las listas enlazadas, los vectores (o arrays en inglés) y las tablas hash. En esencia, todos ellos se comportan de forma similar a los vectores. La diferencia radica en el mecanismo utilizado para seleccionar la posición a utilizar o si las posiciones se pueden enlazar. A continuación, se analizan brevemente las diferencias entre un array y una tabla hash de un nivel, y se selecciona la estructura más apropiada para implementar en el balanceador.

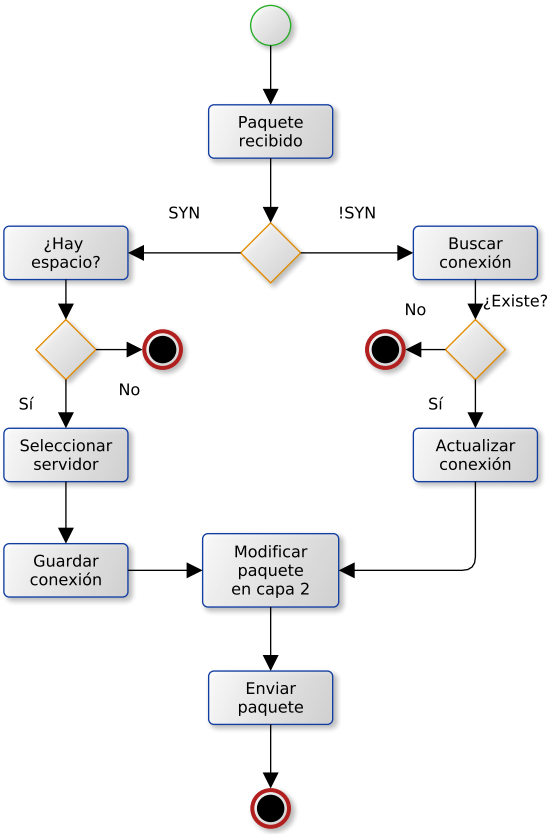


Figura 3.4 Diagrama de flujo del balanceador

Un array es una estructura de datos que permite almacenar un conjunto de datos homogéneo, es decir, todos ellos del mismo tipo. A los datos almacenados en un array se les denomina elementos, y al número de elementos de un array se le denomina tamaño o rango del array. Cada uno de estos elementos puede ser de tipo simple, como caracteres, o de tipo compuesto, como objetos. Para acceder a los elementos individuales de un array se emplea un índice que será un número entero no negativo que indicará la posición del elemento dentro del array. En un array no ordenado, si no se conoce de antemano dónde está el valor deseado, se debe empezar a buscar desde el principio del array comprobando si su contenido coincide con el valor que se busca. Es fácil darse cuenta de que si el valor está al principio del array no se tendrán que recorrer demasiados elementos. Si el valor en cambio está al final, hay que pasar por muchos elementos y por tanto se necesita más tiempo de ejecución para encontrar el valor. En promedio se dice que la cantidad de tiempo que se necesita para encontrar un elemento concreto en una array es proporcional al número de elementos en su conjunto. Cuantos más elementos, más tiempo se tardará en encontrar un elemento específico. Así, la búsqueda en un array se comporta de manera lineal.

Una tabla hash es una estructura de datos que asocia claves con valores. La clave es única para cada elemento de la tabla y es el dato que se utiliza para buscar un determinado valor. A diferencia del array básico, no hay que recorrer la estructura para encontrar el valor. Para almacenar o recuperar un elemento de la tabla hash hay que convertir su clave a un número. Esto se consigue aplicando una función resumen (hash) a la clave del elemento. El número obtenido siempre será el mismo para esa función resumen, y se utilizará como índice de su ubicación en la tabla hash.

Por lo tanto, en una tabla hash las operaciones de búsqueda, inserción y borrado son en el mejor de los casos constantes, ya que se requiere una cantidad constante de tiempo para calcular el hash y luego recuperar ese valor de la posición indicada por ese índice. El problema de la tabla hash es que puede haber colisiones, es decir, que dos cálculos hash apunten a la misma posición. En esos casos hay que tener un método sistemático para colocar ese segundo elemento en la tabla hash. Este proceso se denomina resolución de colisiones, y existen dos métodos básicos para llevarlo a cabo: i) crear una lista enlazada en esa posición del array, y ii) buscar de forma lineal la siguiente posición libre en el array desde la posición calculada por el hash.

El objetivo básico de la tabla hash es minimizar las colisiones, y para eso se debe utilizar un hash que rellene el array de manera uniforme, además de aumentar el tamaño del mismo para minimizar las colisiones, obviamente a un coste mayor de memoria.

En este caso, se ha seleccionado e implementado una tabla hash cuyo método de resolución de colisiones es lineal. Existen dos razones que motivan esta decisión. La primera es que desde el punto de vista del rendimiento, la tabla hash requiere en el mejor de los casos un tiempo constante para encontrar el destino para una clave y guardar sus datos. Además, la función hash asegura que para la misma clave se devuelve la misma posición del array. Los arrays básicos no funcionan con el mismo nivel de eficiencia. En este caso no se conocería el índice donde se encuentra el valor que buscamos, por lo que se debería hacer una búsqueda lineal que llevaría tanto tiempo como el número de elementos en el array. Si el tamaño del array es pequeño, podría no ser un problema. En cambio a medida que el tamaño del array aumente el coste de la búsqueda aumentará de manera proporcional. En definitiva, la tabla hash es más

escalable. La segunda razón para elegir la tabla hash es que la implementación que se ha tomado como referencia (LVS[3]) también utiliza tablas hash. Esta implementación –soportada en Linux– se utilizará en los dos balanceadores de carga con los que se comparará este balanceador C2 en el próximo capítulo.

En cuanto al contenido del array, cada elemento –o entrada– tendrá los siguientes campos:

- Identificador único de la conexión (32 bits)
- Índice del servidor en uso (3 bits)
- Estado de la conexión (2 bits que determinan si la conexión está activa, en proceso de cierre, o inactiva)
- Marca temporal que se guarda cuando se recibe un nuevo paquete (16 bits)

3.1.3.5 Gestión de una conexión balanceada

Como se ha explicado, antes de añadir estos datos es necesario saber la posición del array donde se van a almacenar. Para ello, se ejecuta una función criptográfica sobre los 4 parámetros que identifican la conexión de manera unívoca: direcciones IP de origen y destino, y puertos de origen y destino. Se ha utilizado SipHash [Aumasson y Bernstein, 2012] como función criptográfica, optimizada para entradas de datos pequeñas y rendimiento. A continuación, a este valor se le aplica el operador módulo respecto al número total de posiciones del array. El resultado es la posición del array donde se almacenará la conexión entrante. Solo se pueden sustituir las entradas inactivas (bits de estado a 0). En caso de que la conexión no esté cerrada, se hará una búsqueda lineal de una entrada en estado inactivo desde esa posición.

En cada entrada del array se guardan 32 bits del valor de la función criptográfica como identificador único de la conexión, el índice del servidor que procesará la conexión (índice sobre la lista de direcciones MAC), se establece el primer bit para indicar que es una conexión activa, y también se guarda una marca temporal, en concreto 16 bits que representan el número de *ticks* del dispositivo. Un *tick* es una unidad arbitraria para medir el tiempo interno del sistema. El hecho de utilizar esta marca temporal se debe al funcionamiento del esquema DSR. En este esquema, el balanceador de carga solo ve el tráfico entrante, por lo que obliga a mantener las conexiones activas en la estructura de datos durante un tiempo para evitar descartar conexiones antes de que realmente se cierren. Por lo tanto, en cada elemento del array se guarda una marca temporal que se actualiza cada vez que se recibe un nuevo paquete para la conexión. Esta marca temporal se guarda en milisegundos (unidad utilizada para medir los *ticks* del sistema en el LPC1768), y como se guarda en 16 bits, la resolución del contador es de aproximadamente 65 segundos. Para proceder a la eliminación de entradas caducadas se ha utilizado una rutina periódica que se ejecuta cada 10 segundos y recorre las entradas del array. Si hay entradas sin un nuevo paquete en ese tiempo, el balanceador las marca como inactivas (pone los bits de estado a 0) y reduce el contador de conexiones activas. Este contador sirve para determinar si hay espacio disponible para una nueva entrada. De igual

[3] http://www.linuxvirtualserver.org/

forma, si se recibe un paquete FIN, se marca la entrada en proceso de cierre. Si se recibe un paquete ACK posterior, significa que es el último paquete del cliente que cierra la conexión, y la entrada puede marcarse como cerrada o inactiva. Si este último paquete se perdiese, la función periódica se encargaría de eliminar la entrada.

Una vez guardados todos los campos necesarios en el array, se actualiza el contador de conexiones activas. Por último, se envía el paquete al servidor seleccionado modificando en la capa de enlace de datos la dirección MAC de destino a la dirección MAC del servidor, y utilizando como dirección MAC de origen la propia del balanceador.

En el caso de que el paquete entrante no sea un paquete SYN, se debe comprobar si la conexión existe. Para ello se ejecuta la función criptográfica sobre las direcciones IP y los puertos, y se comprueba el índice del array devuelto por esa función criptográfica. Si la conexión no existe, el paquete se descarta. En cambio, si la conexión está activa y el identificador de la conexión es el mismo que el resultado de la función criptográfica, el paquete se procesa. En este caso se recupera el índice del servidor que está procesando la conexión, se actualiza la marca temporal para ese índice del array, y se envía el paquete a la dirección MAC correspondiente.

3.1.4 Implementación del servidor

El objetivo es implementar un servidor TCP en un dispositivo LPC1768 que acepte y procese las conexiones de los clientes. Como se ha explicado en el apartado 3.1.2.2, se ha utilizado el API BSD de lwIP para implementar el servidor.

Para crear un servidor TCP, se deben ejecutar en orden una serie de funciones proporcionadas por este API:

- socket(): crea un nuevo socket en un dominio concreto, y devuelve un descriptor que se puede utilizar en llamadas de función posteriores. Un socket representa la abstracción de un punto de comunicación por el cual un proceso puede emitir o recibir información, y por tanto permite implementar una arquitectura cliente-servidor.

- bind(): se usa para asociar un socket con una dirección de red. Para un socket de Internet, una dirección de red es la combinación de una dirección IP y un puerto específico.

- listen(): marca el socket al que hace referencia el descriptor como un socket pasivo, es decir, como un socket que se utilizará para aceptar solicitudes de conexión entrantes usando la llamada de función accept().

- accept(): esta función bloquea la ejecución del programa hasta que se reciba una solicitud de conexión entrante para el socket creado. A continuación, se intenta crear una nueva conexión TCP con el cliente remoto. Si tiene éxito, crea un nuevo socket para ese cliente y devuelve el descriptor al que hace referencia para procesar la conexión.

- send()/recv(), write()/read() o sendto()/recvfrom(): estas funciones se utilizan para enviar y recibir datos hacia y desde el otro extremo de la conexión.
- close(): cuando la comunicación con un cliente se termina, se procede al cierre de la misma y se liberan los recursos asignados al socket de cliente.

La figura 3.5 muestra una comunicación cliente-servidor en lwIP y las funciones basadas en el API BSD que se utilizan. Es necesario mencionar que el API de lwIP renombra la función recv() a receive().

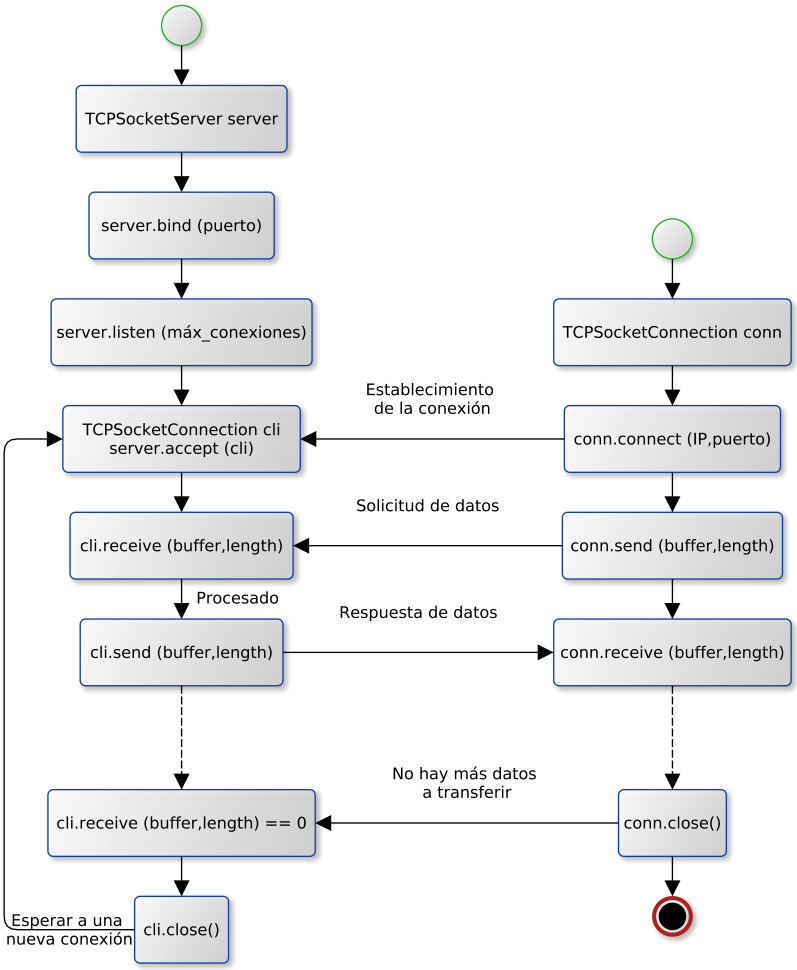


Figura 3.5 Diagrama de flujo de una comunicación TCP utilizando sockets

Para aceptar una conexión TCP, lo primero es configurar la interfaz de red del dispositivo LPC1768. Debido a la limitación de una única interfaz de red, solo se configura la dirección IP estática y la máscara de subred para la VIP del clúster. En cuanto a cómo se ha solucionado el problema ARP, se explicará al final de esta sección.

Una vez configurada la VIP, se implementa el servidor TCP. En primer lugar se crea un objeto TCPSocketServer que representa el socket del servidor (*socket()*). A continuación, se debe asociar el

socket a la interfaz de red y configurar el servidor para que escuche en un puerto específico. El protocolo IP no proporciona números de puerto, son implementados por los protocolos de la capa de transporte, como UDP o TCP. En este caso, la función *bind()* inicializa el socket para el protocolo TCP y configura la estructura de su dirección de socket. Una dirección de socket queda definida por el protocolo, la dirección IP local y el número de puerto de 16 bits. Cuando se crea un socket, se debe especificar la familia de direcciones con la que puede comunicarse el socket. En este caso se establece *AF_INET*, que se utiliza para designar la familia de direcciones del protocolo IPv4. Además, se utiliza el puerto que representa el servicio virtual del clúster, y se establece la dirección del socket a todas las interfaces de red, aunque en este caso solo hay una disponible. A continuación, se empieza a escuchar en ese socket indicando el número máximo de conexiones permitidas (*listen()*). Cuando se recibe una conexión, se acepta (*accept()*), se procesa, y una vez finalizada se vuelve a esperar otra conexión.

3.1.4.1 Diseño de un servidor concurrente

En general, existen dos clases de servidores: los iterativos y los concurrentes. Los primeros solo pueden aceptar y procesar un cliente a la vez, y al acabar con uno empiezan a procesar el siguiente. Los servidores concurrentes por el contrario pueden procesar múltiples clientes a la vez. Para sacar el máximo rendimiento al servidor C2, el objetivo es que sea concurrente.

La técnica más común para crear un servidor concurrente es el uso de hilos [Stevens et al., 2003]. Un hilo de ejecución (*thread* en inglés) es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo, y representa una secuencia de instrucciones ejecutada en paralelo con otras secuencias. En la documentación de lwIP se especifica lo siguiente en lo referente a los hilos: “los hilos de aplicación que utilicen lwIP deben crearse mediante la API sys_thread_new. Internamente esta función hace uso de la función thread_create() del sistema operativo para crear un nuevo hilo. También inicializa las estructuras de tiempo de espera específicos de cada hilo necesarios para la operación de lwIP”. En definitiva, los hilos permiten simplificar el diseño de una aplicación que debe ejecutar tareas específicas en paralelo.

No obstante, se debe tener en cuenta que los procesadores de un núcleo, como el LPC1768, solo pueden ejecutar una instrucción a la vez. Esto significa que el procesador solo puede estar ejecutando una instrucción de uno de los hilos, pero eso no significa necesariamente que la siguiente instrucción que el procesador ejecute sea del mismo hilo. Aquí es donde el sistema operativo y su planificador entran en juego. Es trabajo del planificador determinar cuándo se debe suspender la ejecución de un hilo para dar paso a la ejecución de otro distinto. A esta acción se le llama cambio de contexto (*context switch* en inglés).

En la práctica, una solución común al desarrollar un servidor con hilos es utilizar el patrón productor-consumidor. Este patrón describe dos tipos de hilos, productores y consumidores, que comparten un recurso de tamaño finito. El productor genera y añade elementos al recurso compartido, mientras que el consumidor recupera elementos uno a uno. Como no se pueden crear instancias de un mismo servidor, la tarea del hilo productor (la función main() en este caso) será crear el servidor TCP y esperar nuevas

conexiones (función accept()). Una vez establecida una conexión, se guarda en una cola de conexiones pendientes y se vuelve a esperar otra conexión. Por otra parte, un hilo consumidor recuperará conexiones de la cola y las procesará.

El problema de este patrón reside en que el productor no puede añadir más conexiones que la capacidad de la cola, y que el consumidor no debe intentar recuperar una conexión si la cola está vacía. En definitiva, el productor debe guardar conexiones en la cola y luego notificar a los consumidores que hay conexiones. Por el contrario, un consumidor debe recuperar una conexión de la cola y notificar al productor que se ha vaciado una posición de la cola. Existen diferentes alternativas para abordar este problema utilizando mecanismos de comunicación entre procesos. La comunicación entre procesos (IPC en inglés) es una función básica de los sistemas operativos que permite a los hilos comunicarse y sincronizarse entre sí. Para este problema concreto, los semáforos suelen ser la elección preferente. Un semáforo es una variable especial utilizada para la sincronización entre hilos, de modo que su ejecución se realice de forma ordenada y sin conflictos. Se puede considerar como un contador cuyo valor es el número máximo de hilos que pueden acceder a la sección crítica en un momento dado, y su gestión se realiza a través de 3 funciones:

- init(): inicia el semáforo con el número máximo de accesos permitidos al recurso compartido.
- wait(): si el valor del semáforo es mayor a cero, indica que al menos un hilo puede acceder al recurso. Al hacerlo, el valor del semáforo se reduce en una unidad. Si el valor del semáforo es cero, el hilo queda en espera.
- signal()/post(): cuando el hilo ha terminado de utilizar el recurso, se incrementa el valor del semáforo en 1. Si hay algún hilo esperando su turno para utilizar el recurso, ahora puede acceder.

Conviene destacar que el orden en el que se incrementa o disminuye un semáforo es esencial. Una inadecuada implementación puede terminar en un bloqueo mutuo. Un bloqueo mutuo ocurre en un sistema concurrente cuando un conjunto de hilos quedan en un estado de espera permanente.

En la implementación del servidor se necesitan dos semáforos para la sincronización entre hilos[4]: un semáforo para bloquear a los consumidores cuando la cola está vacía (S1), y otro semáforo para bloquear al productor cuando está llena (S2), es decir, cuando no hay espacio disponible para una nueva entrada. El hilo productor se suspende en la función wait() hasta que haya al menos un espacio disponible en la cola de conexiones. Si lo hay, se bloquea hasta recibir una conexión (función accept()), y después la guarda en la cola, notificando a S1 que hay un elemento en la cola. Por el contrario, los hilos consumidores se suspenden en la función wait() hasta que haya al menos un elemento en la cola de conexiones. Cuando un consumidor recupere una conexión de la cola, notificará al semáforo S2 que se ha liberado un espacio con la función signal(). El semáforo S2 debe tener un valor inicial igual al tamaño de la cola. La cola está vacía inicialmente, y por tanto se puede permitir ese número de accesos. El valor inicial de S1 es cero, porque inicialmente la cola está vacía y no se debe permitir que ningún consumidor acceda a la misma.

[4] https://www.cs.mtu.edu/~shene/NSF-3/e-Book/SEMA/TM-example-buffer.html

La cola de conexiones pendientes es el recurso compartido y debe tener un tamaño finito. En este caso se ha establecido a 4, ya que valores más altos empezaban a causar problemas de memoria. Al limitar el número máximo de conexiones, se puede restringir también el número de hilos consumidores a 4, permitiendo por tanto cuatro conexiones concurrentes en el servidor C2. Por tanto, el hilo productor (función main()) crea 4 hilos consumidores –uno para cada posible cliente– con la función *sys_thread_new* antes de bloquearse esperando a una conexión. Todos estos hilos tienen la misma prioridad para evitar que durante el cambio de contexto hilos con menor prioridad se vean perjudicados en tiempo de CPU. Puesto que cada paso por el semáforo S2 hace que el contador disminuya en uno, cuando la cola de conexiones está llena, el contador del semáforo pasa a valer cero y el productor se bloqueará, rechazando conexiones adicionales. Al bloquear al productor cuando la cola se llena, se evita aceptar conexiones (función accept()) que no podrían guardarse en la cola, y que por tanto tendrían que descartarse. Si bien esta estrategia limita la concurrencia, proporciona latencias más previsibles y evita la sobrecarga de recursos [Stevens et al., 2003].

No obstante, este planteamiento presenta un problema evidente, y es que la cola es compartida por todos los hilos. Si el semáforo permite el acceso a la cola a dos o más hilos, es posible que intenten acceder o modificar los mismos datos de forma concurrente. El resultado de la operación dependerá del orden de ejecución de los hilos. A esta situación se le denomina condición de carrera (*race condition* en inglés). Para evitarla, las operaciones sobre el recurso compartido deben ejecutarse de forma exclusiva. En este caso, el primer hilo que intente tener acceso a la cola debe establecer un indicador de exclusión mutua o *mutex* antes de ejecutar el código correspondiente. Si un mutex está en uso y otro hilo intenta adquirirlo, ese hilo se bloquea hasta que el mutex es liberado por el hilo original. En definitiva, si varios hilos compiten por el mismo mutex, solo uno puede acceder a él. En este caso, tanto el productor como los consumidores deben utilizar el mutex para acceder a la cola.

Listado 3.1 Pseudocódigo del servidor con patrón productor-consumidor

```
//Mutex acceso al buffer
mutex buffer_mutex;
//Número de elementos en el buffer, disponibles para ser leídos
semaphore S1 = 0;
//Número de espacios disponibles en el buffer
semaphore S2 = BUFFER_SIZE;

producer() {
    while (true) {
                wait(S2);
        conn = accept();
            set(buffer_mutex);
                insertBuffer(conn);
            release(buffer_mutex);
        signal(S1);
    }
}

consumer() {
    while (true) {
```

```
        wait(S1);
            set(buffer_mutex);
                conn = getFromBuffer();
            release(buffer_mutex);
        signal(S2);
        processConnection(conn);
    }
}
```

Cabe destacar que las operaciones sobre un semáforo o mutex son atómicas. Una vez que la operación se inicie, continuará hasta el final sin interrupción, evitando de esta forma una condición de carrera al acceder a estos mecanismos de sincronización.

En cuanto a la aplicación de servidor que ejecutan los hilos consumidores, se ha implementado una aplicación básica que permite medir el rendimiento de red de una conexión con un cliente. En concreto, el cliente envía datos y la aplicación de servidor solo responde con reconocimientos de recepción, sin enviar datos de vuelta, es decir, solo ejecuta la función recv(). Esta aplicación se implementa en un bucle infinito. De esta forma los hilos consumidores no tienen que destruirse ni volver a crearse de nuevo al finalizar la conexión con un cliente. Cuando no haya más datos (*connection.receive(buffer,length) == 0*), se cierra la conexión y se espera a que haya otra en la cola de conexiones pendientes. Los clientes y sus datos son procesados con la estructura TCPSocketConnection.

3.1.4.2 Solución al problema ARP con una única interfaz de red

Por último, explicar cómo se ha solucionado el problema ARP en el servidor C2. En la versión de lwIP utilizada, el enrutamiento y la estructura de interfaces se ha simplificado deliberadamente para su aplicación en microcontroladores de bajos recursos. Esto significa que solo se puede tener una dirección IP para cada interfaz de red, y que tener varias interfaces de red en la misma subred puede llevar a errores de funcionamiento, como está documentado en los foros de lwIP y se ha comentado en la sección 3.1.3.1. Por lo tanto, se debe configurar la VIP como la dirección IP de la interfaz de red de los servidores. Pero entonces, ¿cómo se obtienen las direcciones MAC de los clientes sin tener otra dirección IP que permita realizar las solicitudes ARP?

La solución implementada es la siguiente: se deja ARP habilitado para la interfaz de red (#define LWIP_ARP 1), y en el código que procesa los paquetes Ethernet (ethernet_input) se filtra el paquete entrante, y si se trata de un paquete ARP, se procesa pero no se responde. De esta forma, se evita que el servidor responda con su dirección MAC al paquete ARP y rompa el esquema de balanceo, pero a la vez se obtiene la información necesaria del cliente. Es decir, las solicitudes ARP que llegan para la VIP tendrán la dirección IP y MAC del cliente que quiere usar el servicio del clúster. Lo que se consigue es aprender la asociación de direcciones IP-MAC del cliente a través de los paquetes ARP recibidos. Estas asociaciones se guardan en una estructura de datos auxiliar. El servidor que reciba el paquete SYN para ese cliente, remitido por el balanceador, copiará esa asociación IP-MAC a la caché ARP, y el resto de servidores descartarán esa entrada auxiliar pasado un tiempo. La caché ARP es simplemente una

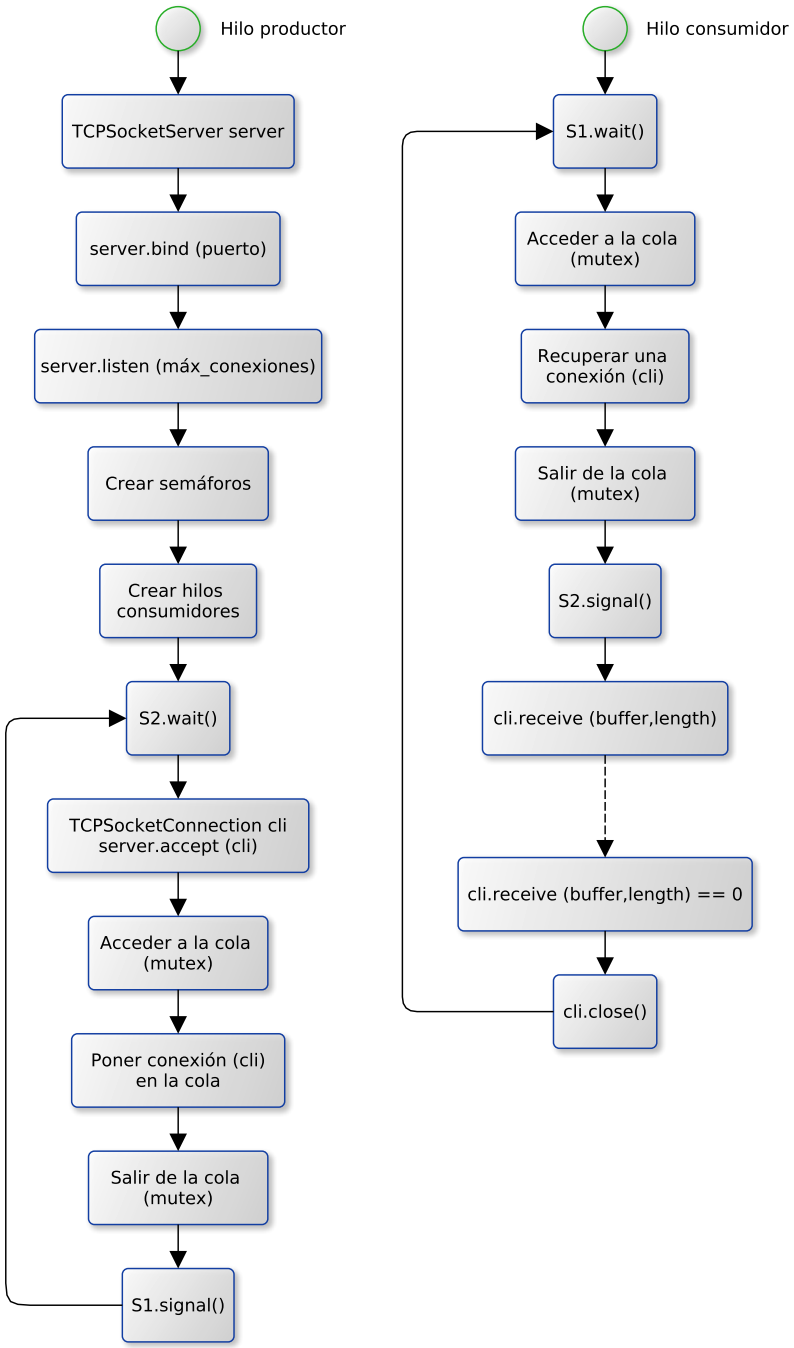


Figura 3.6 Diagrama de flujo de una comunicación TCP con hilos

estructura de datos que guarda las direcciones IP y MAC de conexiones recientes para evitar tener que enviar paquetes ARP por cada paquete IP entrante, lo que sería ineficiente.

Para poder guardar esas direcciones hay que habilitar una configuración en lwIP que permite guardar entradas en la caché de manera estática (#define ETHARP_SUPPORT_
STATIC_ENTRIES 1) utilizando la rutina *etharp_add_static_entry(IP,MAC)*. En definitiva, cuando llega un paquete ARP, en vez de pasárselo a la rutina ARP para que lo procese (etharp_arp_input), se extraen las direcciones IP y MAC del cliente y se guardan de manera provisional. Al recibir un paquete SYN con esa dirección IP, se guarda la entrada en la caché ARP interna. En caso de recibir un paquete SYN con una dirección IP que no esté en la caché ARP, se descartará el paquete. Una opción alternativa a descartar paquetes podría pasar por cambiar momentáneamente la dirección IP de la interfaz de red, mandar la solicitud ARP correspondiente, y al recibir la respuesta volver a configurar la VIP. No obstante, esta opción solo se podría aplicar si el servidor no está procesando conexiones para la VIP en ese momento.

Por otra parte, se ha observado que lwIP se bloquea en ocasiones en el cierre pasivo, más concretamente en el estado LAST-ACK. Conviene recordar que al estado LAST-ACK se llega cuando se recibe un paquete FIN del cliente para cerrar la conexión, pero todavía se necesita confirmación para liberar los recursos y cerrar la conexión. En definitiva, el servidor responderá con su paquete FIN correspondiente y esperará a que llegue un último ACK.

Por lo general, quedarse bloqueado en LAST-ACK significa que la aplicación mantiene el socket abierto incluso cuando el otro extremo ha terminado de enviar datos. Esto puede ocurrir por varias razones. En este caso, la opción más probable es que el balanceador de carga descarte el último paquete ACK que llega del cliente ya que no tiene descriptores de recepción disponibles para procesar el paquete. lwIP cierra una conexión en estado LAST-ACK después de transcurrir dos veces el tiempo definido por el MSL (Maximum Segment Lifetime). EL MSL es el tiempo más largo en milisegundos que se estima que un paquete TCP pueda existir en la red. Durante este tiempo, el servidor mantiene los recursos para esa conexión a medio cerrar.

La solución para evitar un bloqueo prolongado en el servidor es reducir el tiempo de MSL. En lwIP por defecto el MSL es de 60 segundos (#define TCP_MSL 60000), un tiempo excesivo en este contexto. Se ha decidido reducirlo a 1 segundo (#define TCP_MSL 1000), un tiempo de espera menos permisivo con el objetivo de liberar los recursos de esa conexión para procesar una nueva.

3.2 SYN cookies

Como ya se ha explicado en el capítulo 2, el objetivo de las SYN cookies es retrasar el almacenamiento de una solicitud en la cola de conexiones hasta que se haya recibido un ACK. En vez de asignar memoria para un paquete SYN entrante, el estado de la conexión se codifica parcialmente con una función criptográfica, y su resultado (cookie) se devuelve al cliente (SYN-ACK) como el número de secuencia inicial (ISN) del servidor. Dado que TCP requiere que el cliente reenvíe de nuevo ese ISN en el siguiente ACK, el servidor puede verificarlo y, en consecuencia, crear una conexión utilizando la información codificada.

El formato exacto de las cookies se interpreta solo a nivel local, por lo tanto sus parámetros y los procedimientos para su generación y validación varían entre implementaciones. Las SYN cookies están presentes en la actualidad en dos sistemas operativos de tipo Unix: Linux (habilitadas por defecto en varias distribuciones incluyendo Ubuntu y Debian) y FreeBSD [Mahimkar et al., 2007]. Pueden existir más implementaciones, por ejemplo en firewalls o proxys comerciales, pero no están disponibles como código abierto y por tanto no se pueden evaluar. La revisión de los algoritmos de generación y validación de SYN cookies que están disponibles en esos sistemas operativos de tipo Unix permite determinar la viabilidad de su implementación en dispositivos C2.

En primer lugar se revisan las dos implementaciones (Linux y FreeBSD) y se mide su rendimiento. De esta revisión se extraen tres conclusiones: i) en ambas implementaciones el tiempo de validez de una cookie es excesivo para un RTT (Round Trip Time) de una conexión legítima, ii) la implementación de Linux es más robusta pero más lenta que la de FreeBSD, y iii) la implementación de FreeBSD computa cookies para cada paquete SYN entrante. En consecuencia se propone una solución híbrida que aplique lo mejor de cada implementación, y que además reduzca el tiempo de validez de una cookie. Por último, se mide el rendimiento de las tres implementaciones en un dispositivo C2 –bajo un ataque SYN de baja tasa– para determinar si las diferencias son estadísticamente significativas, y por tanto si la propuesta híbrida es adecuada para estos dispositivos.

3.2.1 Revisión de las implementaciones

Aunque la lógica detrás de las SYN cookies es relativamente sencilla, su implementación es más complicada [Zuquete, 2002]. En primer lugar, las cookies deben caber en el espacio definido para el campo ISN (32 bits). En segundo lugar, la generación de cookies debe respetar la recomendación TCP referente a que los valores ISN deben aumentar con el tiempo para reducir la probabilidad de que paquetes con retardo sean aceptados por nuevas conexiones (RFC 793). En tercer lugar, las cookies deben ser impredecibles para evitar su falsificación. Y en último lugar, las cookies deben contener parte de las opciones TCP enviadas por los clientes en los paquetes SYN.

A continuación se muestra cómo se implementan estos aspectos en Linux y FreeBSD.

3.2.1.1 Linux

Las SYN cookies fueron incluidas por primera vez en Linux con un parche en su versión 2.1.44. Como se ha explicado en el capítulo anterior, TCP utiliza números de secuencia de 32 bits para realizar un seguimiento de los datos que se han enviado. El problema de las SYN cookies es que estos 32 bits no son suficientes para guardar toda la información de una conexión.

El número de secuencia construido por las SYN cookies originales se basaba en las siguientes partes:

- una marca de tiempo con incremento lento (el tiempo del sistema desplazado a la derecha 6 posiciones, lo que da una resolución de 64 segundos) (5 bits)
- el valor del tamaño máximo de segmento (MSS) codificado (3 bits)

- el resultado de una función secreta de cifrado, calculada en base a las direcciones IP y números de puerto del cliente y servidor (24 bits)

Esta implementación codificaba el contador y el valor MSS en los primeros 8 bits del número de secuencia, dejando así los últimos 24 bits para el valor generado por la función criptográfica [Smith y Matrawy, 2008]. Esta distribución hacía que la dificultad de falsificar una cookie se redujera a 2^{24}, es decir, el atacante debía adivinar 24 bits en lugar de 32. En la actualidad, Linux añade otro secreto seleccionado por el servidor para dificultar la falsificación. A continuación se muestran los algoritmos para generar y validar SYN cookies en la versión 4.8 de Linux:

Tabla 3.4 Parámetros de la implementación Linux

Parámetro	**Significado**
K_1, K_2	Claves secretas
IP_s, IP_d	Direcciones IP de origen y destino
$Port_s$, $Port_d$	Puertos de origen y destino
ISN_s, ISN_d	Números de secuencia inicial de origen y destino
ACK	Número de reconocimiento
SEQ	Número de secuencia
MSS	2 bits que codifican el MSS del cliente
count	contador de minutos de 32 bits
hash()	función criptográfica de 32 bits (SHA-1)

Generación de una cookie:

$$H_1 = hash(K_1, IP_s, IP_d, Port_s, Port_d) \tag{3.1}$$

$$H_2 = hash(K_2, count, IP_s, IP_d, Port_s, Port_d) \tag{3.2}$$

$$ISN_d = H_1 + ISN_s + (count * 2^{24}) + (H_2 + MSS) \text{ mód } 2^{24} \tag{3.3}$$

A continuación, se muestra cómo se verifican las cookies cuando se recibe un paquete ACK:

$$ISN_d = ACK - 1 \tag{3.4}$$

$$ISN_s = SEQ - 1 \tag{3.5}$$

$$count_{cookie} = (ISN_d - H_1 - ISN_s)/2^{24} \tag{3.6}$$

$$count_{\Delta} = count_{current} - count_{cookie} \tag{3.7}$$

$$MSS_{cookie} = (ISN_d - H_1 - ISN_s) \text{ mód } 2^{24} - H_2 \text{ mód } 2^{24} \tag{3.8}$$

Existen dos controles de integridad, y si alguno de ellos falla, la cookie se rechaza. En Linux, las SYN cookies utilizan un contador que se incrementa cada minuto. Una cookie solo es válida si la diferencia entre el valor del contador actual y el codificado es menor que 2 (MAX_SYNCOOKIE_AGE). Por lo tanto, una cookie es válida como máximo durante 2 minutos. El segundo control de integridad evalúa si el valor de MSS está dentro del rango de 2 bits (0-3). Si la cookie no es válida, el valor MSS

estará fuera de ese rango. De lo contrario, la cookie se considerará válida y se creará una nueva conexión en el estado ESTABLISHED.

Por lo tanto, hay 2 valores de contador válidos y 4 posibles valores de MSS. Cada una de estas combinaciones da como resultado un ISN válido (cookie) que será aceptado por el servidor. Esto significa que el número de paquetes necesarios para falsificar una conexión se puede reducir a $2^{32}/8$ durante una inundación SYN[5].

Al recibir un paquete ACK, el sistema operativo comprueba si la conexión existe en la cola SYN. Si no existe, comprueba si ha habido un desbordamiento reciente de la cola para ese socket que explique el envío de cookies (tcp_synq_no_recent_overflow). En definitiva, el sistema operativo no quiere aceptar cookies si no ha habido una inundación reciente. No obstante, si se envían cookies y luego se sale de este modo, se seguirán aceptando cookies si se reciben hasta 2 minutos después del final de la inundación SYN (TCP_SYNCOOKIE_VALID). El siguiente código muestra este comportamiento. Es necesario señalar que la función time_after(a,b) devuelve *true* si el tiempo definido por *a* es posterior a *b*.

Listado 3.2 Comprobación inundación SYN

```
#define MAX_SYNCOOKIE_AGE       2
#define TCP_SYNCOOKIE_PERIOD    (60 * HZ)
#define TCP_SYNCOOKIE_VALID     (MAX_SYNCOOKIE_AGE * TCP_SYNCOOKIE_PERIOD)
...
if (tcp_synq_no_recent_overflow(sk))
   goto out;
...
static inline bool tcp_synq_no_recent_overflow(const struct sock *sk)
{
   unsigned long last_overflow = tcp_sk(sk)->rx_opt.ts_recent_stamp;
   return time_after(jiffies, last_overflow + TCP_SYNCOOKIE_VALID);
}
```

Debido al funcionamiento de TCP, las conexiones no pueden cerrarse inmediatamente ya que los paquetes pueden llegar fuera de orden o ser recibidos después de que la conexión se haya cerrado. El estado TIME_WAIT evita que paquetes con retraso sean aceptados por una conexión posterior. El equipo que inicia el cierre activo (primer paquete FIN) es el que entra en el estado TIME_WAIT, donde la conexión está todavía activa. Por el contrario, un cierre pasivo permite deshacerse rápidamente de la conexión. Los servidores tienden a evitar este estado para liberar recursos y poder así atender más peticiones. Faber et al. [1999] explican que "*los servidores que tienen muchas conexiones TCP en estado TIME_WAIT sufren una degradación en su rendimiento, y pueden colapsar.*".

Los usuarios de Linux pueden ejecutar el comando $netstat - nt$ cuando navegan por Internet y ver cómo las conexiones en estado TIME_WAIT generalmente están en su ordenador, y no en el servidor. A continuación se puede ver un ejemplo donde la dirección local es 10.48.1.46 (el puerto 80 es HTTP y el puerto 443 es HTTPS).

Listado 3.3 Resultado netstat

[5] http://www.jakoblell.com/blog/2013/08/13/quick-blind-tcp-connection-spoofing-with-syn-cookies/

```
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp        0      0 10.48.1.46:55838      185.43.181.169:80      TIME_WAIT
tcp        0      0 10.48.1.46:55865      216.58.211.238:80      ESTABLISHED
tcp        0      0 10.48.1.46:34580      74.125.168.238:443     TIME_WAIT
tcp        0      0 10.48.1.46:42353      216.58.210.163:443     ESTABLISHED
tcp        0      0 10.48.1.46:34579      74.125.168.238:443     TIME_WAIT
```

Por lo tanto, si la conexión se cierra en el servidor (estado CLOSED), la misma combinación de dirección de origen y puerto estará disponible de nuevo. Al recibir un ACK con una cookie repetida, el servidor debe asumir que es un tercer paquete válido de una fase de establecimiento. El destino nunca será capaz de determinar si corresponde a una conexión cerrada, ya que el estado para esa conexión se ha eliminado. Por lo tanto, un cliente podría reutilizar cookies obtenidas de manera legítima o ilegítima para ejecutar un establecimiento de conexión con un solo paquete. Cabe destacar que aunque el estándar TCP asume que el establecimiento se completa antes de enviar los datos, es posible añadirlos al último paquete ACK de esta fase. Esto significa que el establecimiento de una conexión TCP con datos de carga útil (como una solicitud HTTP) puede reducirse a enviar un solo paquete.

En este escenario, las SYN cookies permiten al cliente no esperar un SYN-ACK la mayor parte del tiempo. Con la cola de conexión tan limitada de los dispositivos C2, un cliente podría enviar periódicamente el mínimo número de paquetes SYN para llenar la cola de conexiones y forzarlo a responder con SYN cookies. Entonces podría abrir una nueva conexión, guardar la cookie recibida, y reutilizarla para acelerar la apertura de sucesivas conexiones durante 2 minutos. Este procedimiento tendría el mismo resultado que el mecanismo TCP Fast Open [Radhakrishnan et al., 2011], pero sin el consentimiento explícito del servidor.

3.2.1.2 FreeBSD

FreeBSD utiliza una SYN caché [Lemon, 2002] para almacenar un subconjunto de la información de la conexión. Cuando se realiza el establecimiento de la conexión, el sistema operativo crea una conexión completa con las opciones almacenadas en la entrada de la SYN caché, que luego libera. Las SYN cookies se utilizan cuando la caché está llena.

A continuación, se muestran los algoritmos para generar y validar SYN cookies en FreeBSD 11.0 (revisión 308048):

Generación de una cookie:

$$h = hash(K_P, IP_s, IP_d, Port_s, Port_d, ISN_s, options) \tag{3.9}$$

$$ISN_d = h \,\&\, \sim 0xff \tag{3.10}$$

$$ISN_d| = options \oplus (h \gg 24) \tag{3.11}$$

A continuación, se muestra cómo se verifican las cookies cuando se recibe un paquete ACK:

Tabla 3.5 Parámetros de la implementación FreeBSD

Parámetro	**Significado**
IP_s, IP_d	Direcciones IP de origen y destino
$Port_s$, $Port_d$	Puertos de origen y destino
ISN_s, ISN_d	Números de secuencia inicial de origen y destino
ACK	Número de reconocimiento
SEQ	Número de secuencia
MMM	3 bits que codifican el MSS del cliente
WWW	3 bits que codifican el escalado de ventana del cliente
S	1 bit que define si el acuse de recibo selectivo está habilitado
P	1 bit que define cuál de las dos claves está en uso
options	8 bits (WWWMMMSP)
K_P	Clave secreta en uso
hash()	función criptográfica de 32 bits (SipHash)

$$ISN_d = ACK - 1 \tag{3.12}$$

$$ISN_s = SEQ - 1 \tag{3.13}$$

$$options = (ISN_d \& 0xff) \oplus (ISN_d \gg 24) \tag{3.14}$$

$$h = hash(K_P, IP_s, IP_d, Port_s, Port_d, ISN_s, options) \tag{3.15}$$

$$(ISN_d \,\&\sim 0xff) \tag{3.16}$$

$$(h \,\&\sim 0xff) \tag{3.17}$$

En primer lugar, se invierte el avance de los números de secuencia. A continuación se extraen las opciones codificadas (8 bits), entre ellas cuál de las dos claves se utilizó para producir la cookie. En lugar de un contador codificado en la cookie, el algoritmo utiliza dos claves que se alternan, y la propia cookie codifica qué clave se utilizó para producirla. Cada 15 segundos se actualiza una de las claves con un nuevo valor aleatorio. Finalmente, si la función criptográfica local coincide con el valor recibido del ACK, la cookie se considera válida y se establece la conexión. Como solo se usan 24 bits para la función criptográfica, si un atacante adivina esos bits, el sistema operativo aceptará el paquete ACK. Esto significa que el número promedio de paquetes necesarios para falsificar una conexión se puede reducir a 2^{24}, pero a diferencia de Linux, este intento de falsificación puede suceder incluso si no se recibe una inundación SYN.

Si las SYN cookies están habilitadas, FreeBSD generará una cookie para cada paquete SYN recibido [McKusick et al., 2014]. Cuando el sistema operativo recibe un ACK para un socket en el estado LISTEN, busca si la conexión está en la SYN caché. Si no hay entrada en la caché, comprueba si el ACK contiene una cookie válida. Este funcionamiento sigue la propuesta que hizo Lemon en su implementación de SYN caché [Lemon, 2002]: "*un enfoque alternativo que puede resultar factible es utilizar una SYN cookie como la ISN para todas las conexiones, en lugar de utilizar un número aleatorio como es el caso*

de SYN caché. Esto permitiría que el mecanismo de sustitución de las entradas dentro del SYN caché funcionase con normalidad, ya que el ACK de respuesta podría ser aceptado por cumplir la verificación de SYN cookie, o coincidiendo con una entrada SYN caché existente".

Como el sistema operativo no comprueba si se ha producido un desbordamiento reciente de la SYN caché antes de aceptar cookies, es posible inundar el servidor con ACKs en un intento de crear una conexión. Esto puede proporcionar una forma de evitar los firewalls convencionales que filtran los paquetes entrantes con el bit SYN activo. Además, el coste de validar cookies puede ser sustancial en comparación con el procesamiento de paquetes ACK regulares.

En cuanto a la reutilización de cookies, ya que se generan para cada paquete SYN recibido y la función de validación no aplica ningún criterio temporal para dejar de aceptarlas, un cliente podría guardar y reutilizar cookies para conseguir un establecimiento de conexión con un solo paquete. A pesar de que las cookies son válidas como máximo durante 30 segundos, cuatro veces menos que Linux, el cliente no tiene que desbordar la SYN caché para reutilizarlas.

3.2.1.3 Tiempo de ejecución

En esta sección se realiza un análisis preliminar en un dispositivo de recursos no limitados para determinar qué implementación tiene un menor tiempo de ejecución en procesar cookies. El dispositivo de recursos no limitados es un ordenador Intel Core i5-560M con 4 núcleos a 2,67 GHz, 6 GB de RAM DDR3 y funcionado sobre Ubuntu 14.04.

Los sistemas modernos Linux proporcionan varios relojes y funciones de temporización con diferentes tipos de resolución. En este caso, se utiliza la función *clock_gettime()*, ya que proporciona marcas de tiempo desde la fuente de reloj especificada con resolución de nanosegundos[6]. Esta función de temporización permite elegir la fuente de reloj de entre las disponibles en el sistema. Con el fin de evitar el sesgo del reloj del sistema u otros problemas derivados de los ajustes de tiempo, se utiliza CLOCK_MONOTONIC que garantiza valores monotónicamente crecientes entre dos medidas sucesivas. Por último, la estructura *timespec* utilizada en clock_gettime almacena la hora actual, guardando los segundos y nanosegundos por separado en contadores de 32 bits.

Listado 3.4 Tiempo de ejecución

```
struct timespec tsi, tsf;

/* Determine clock_gettime overhead */
clock_gettime(CLOCK_MONOTONIC, &tsi);
clock_gettime(CLOCK_MONOTONIC, &tsf);
long elaps_ns_clock_gettime = tsf.tv_nsec - tsi.tv_nsec;

/* Determine loop overhead */
clock_gettime(CLOCK_MONOTONIC, &tsi);
for (i=0; i<ITERATIONS; i++);
clock_gettime(CLOCK_MONOTONIC, &tsf);
long elaps_ns_for = tsf.tv_nsec - tsi.tv_nsec;
```

[6] https://linux.die.net/man/3/clock_gettime

```
/* Determine cookie generation overhead */
clock_gettime(CLOCK_MONOTONIC, &tsi);
for(i=0;i<ITERATIONS;i++){
        cookie_generate(Saddr, Daddr, Sport, Dport, ISNcli, MSS);
}
clock_gettime(CLOCK_MONOTONIC, &tsf);
long elaps_ns = tsf.tv_nsec - tsi.tv_nsec;
elaps_ns = (elaps_ns-elaps_ns_for-elaps_ns_clock_gettime)/ITERATIONS;

/* Determine cookie lookup overhead */
clock_gettime(CLOCK_MONOTONIC, &tsi);
for(i=0;i<ITERATIONS;i++){
        cookie_lookup(Saddr, Daddr, Sport, Dport, ISNcli, cookie);
}
clock_gettime(CLOCK_MONOTONIC, &tsf);
elaps_ns = tsf.tv_nsec - tsi.tv_nsec;
elaps_ns = (elaps_ns-elaps_ns_for-elaps_ns_clock_gettime)/ITERATIONS;
```

Este programa determina el tiempo que tardan en ejecutarse una gran cantidad de iteraciones de la generación y validación de cookies. Estas iteraciones permiten una mayor precisión, y de ahí se puede determinar el tiempo para una sola operación. Para aumentar la precisión de los resultados, se mide el tiempo de un bucle for vacío y de la función clock_gettime, y se restan del tiempo que tarda en completarse la operación.

La siguiente tabla muestra un resumen de los resultados de este estudio:

Tabla 3.6 Tiempo de ejecución en nanosegundos

	Linux	**FreeBSD**
Generación de cookie (Media)	6582,13	475,53
Generación de cookie (σ)	44,38	6,51
Validación de cookie (Media)	5649,22	471,47
Validación de cookie (σ)	17,29	4,97

El análisis de rendimiento muestra que la implementación de Linux podría ser más adecuada para dispositivos de recursos no limitados debido al tiempo de cómputo para dos funciones SHA-1 por cada solicitud de conexión (SYN). La implementación de FreeBSD en cambio parece más adecuada para dispositivos de recursos limitados, ya que solo utiliza un cómputo SipHash [Aumasson y Bernstein, 2012], una función resumen o criptográfica más eficiente para entradas cortas. Sin embargo, depende de una renovación periódica de las claves utilizadas por la función resumen (uso adicional de CPU). Por lo tanto, una solución híbrida que aplicase la función resumen de FreeBSD y utilizase un contador codificado en la cookie en vez de una renovación periódica de las claves sería más apropiada para un dispositivo C2. Además, se observa la necesidad de reducir el tiempo de validez de una cookie a valores adecuados para un RTT normal, lo que reduciría la probabilidad de ataques de repetición con cookies anteriores.

3.2.2 Propuesta híbrida

Un ataque de repetición es una forma de ataque de red en el cual una transmisión de datos válida es maliciosamente repetida. Este ataque puede ser ejecutado por un atacante que intercepta la información y la reproduce, o por el propio autor de la transmisión con el objetivo de reproducir el efecto.

En general se puede afirmar que los ataques de repetición no funcionan bien con TCP, ya que cada conexión depende de dos números de secuencia de 32 bits generados de manera aleatoria por el cliente y el servidor. Para que un ataque de repetición funcione, el atacante debe adivinar el número de secuencia generado por el servidor. Sin embargo, si se aplican las SYN cookies, el número de secuencia del servidor no se calcula de forma aleatoria cada vez que se recibe un paquete SYN. Para la combinación de direcciones de origen y destino, y puertos, el número de secuencia generado (cookie) será el mismo durante un tiempo.

De la revisión anterior se observa que el tiempo de expiración de una cookie es de varias decenas de segundos. El RTT entre un SYN-ACK y el ACK correspondiente del cliente suele ser inferior a 500ms para más del 90 % de las conexiones [Korczynski et al., 2011]. Pruebas experimentales en Internet demuestran esta afirmación, con un RTT medio entre el SYN-ACK y el ACK de 145ms [Sessini y Mahanti, 2006].

Esto significa que tanto Linux como FreeBSD utilizan un tiempo de expiración demasiado largo para un establecimiento de conexión normal. Por otro lado, FreeBSD no comprueba si ha habido un desbordamiento reciente en el socket, requiere una rutina periódica para actualizar las claves criptográficas, y reduce el esfuerzo requerido para adivinar un ISN. Por lo tanto, se utiliza la implementación de Linux como referencia para la propuesta híbrida. El objetivo es reducir el tiempo de expiración y utilizar la misma función criptográfica que FreeBSD para mejorar el rendimiento.

En Linux, las SYN cookies caducan al codificar un contador que se incrementa cada minuto. Al generar una cookie, el contador carga el número de *ticks* que se han producido desde que arrancó el sistema y, a continuación, convierte ese valor a minutos. En lugar de usar una conversión a minutos, se propone una conversión a segundos para minimizar el tiempo de expiración.

A pesar de que el contador es de 32 bits, la forma en que se codifica en la cookie (8 bits superiores) hace que rote cada 256 minutos. Esto significa que si el resto de los parámetros son iguales, el valor de una cookie generada cada 256 minutos será el mismo. Si se codifica el contador de segundos de la misma manera, rotará cada 256 segundos, demasiado rápido. Por ello, se utilizan 14 bits para que rote de manera similar (16384 segundos = 273 minutos).

A continuación, se muestran los algoritmos de generación y validación propuestos:

Generación de una cookie:

$$H_1 = hash(K_1, IP_s, IP_d, Port_s, Port_d) \tag{3.18}$$

$$H_2 = hash(K_2, count, IP_s, IP_d, Port_s, Port_d) \tag{3.19}$$

$$ISN_d = H_1 + ISN_s + (count * 2^{18}) + (H_2 + MSS) \text{ mód } 2^{18} \tag{3.20}$$

Tabla 3.7 Parámetros de la implementación híbrida

Parámetro	**Significado**
K_1, K_2	Claves secretas
IP_s, IP_d	Direcciones IP de origen y destino
$Port_s$, $Port_d$	Puertos de origen y destino
ISN_s, ISN_d	Números de secuencia inicial de origen y destino
ACK	Número de reconocimiento
SEQ	Número de secuencia
MSS	2 bits que codifican el MSS del cliente
count	contador de segundos de 32 bits
hash()	función criptográfica de 32 bits (SipHash)

Verificación de una cookie:

$$ISN_d = ACK - 1 \tag{3.21}$$

$$ISN_s = SEQ - 1 \tag{3.22}$$

$$count_{cookie} = (ISN_d - H_1 - ISN_s)/2^{18} \tag{3.23}$$

$$count_{\Delta} = count_{current} - count_{cookie} \tag{3.24}$$

$$MSS_{cookie} = (ISN_d - H_1 - ISN_s) \text{ mód } 2^{18} - H_2 \text{ mód } 2^{18} \tag{3.25}$$

Para mantener la coherencia con la implementación de Linux, una cookie solo será válida si la diferencia entre el valor actual del contador y el codificado es menor que 2 (MAX_SYNCOOKIE_AGE), es decir, se aceptan cookies de hasta 2 segundos. Además, como también se admiten 4 posibles valores de MSS, el servidor acepta igualmente 8 combinaciones válidas. Finalmente, la variable TCP_SYNCOOKIE_VALID también se reduce a 2 segundos, es decir, se dejan de aceptar cookies si se reciben 2 segundos después del final del episodio de inundación SYN. Si este tiempo de expiración es demasiado restrictivo para el RTT de la red, se pueden incrementar ligeramente estas dos variables.

En resumen, una cookie producida con este algoritmo híbrido es válida como máximo durante dos segundos, lo que hace que la reutilización de cookies sea mucho más difícil. El próximo paso es implementar estos algoritmos en un dispositivo C2 y evaluar su rendimiento.

3.2.3 Implementación en un dispositivo C2

En este apartado se explica cómo se han implementado las cookies en el dispositivo LPC1768 y la pila lwIP. Para ver exactamente dónde se deben implementar los algoritmos de generación y validación de las SYN cookies, es necesario analizar cómo procesa el receptor TCP de la pila lwIP un paquete entrante.

En primer lugar, lwIP utiliza una única cola de conexión, cuyo tamaño está determinado por el argumento MEMP_NUM_TCP_PCB (establece el número máximo de conexiones TCP activas que la pila soportará). Cada conexión tendrá asociada una estructura tcp_pcb que contiene la información sobre

el estado de la conexión. Esta estructura de datos, o simplemente PCB (Protocol Control Block), es el equivalente al TCB del que se habló en el capítulo 2.

La función encargada de procesar los paquetes TCP entrantes es *tcp_input()*, en el fichero tcp_in.c, la cuál recibe como parámetros el contenido del paquete y la interfaz de red por donde se ha recibido. Esta función verifica la cabecera TCP, demultiplexa el paquete y lo pasa a la función *tcp_process()*, que implementa la máquina de estados finitos de TCP.

Cuando se recibe un paquete para un socket en el estado LISTEN, se llama a la función *tcp_listen_input()*, también en el fichero tcp_in.c.

Si se trata de un paquete SYN, lwIP intenta asignar y crear una nueva estructura TCP mediante la llamada a la función *tcp_alloc()*. Esta función intentará reservar la memoria necesaria para crear una estructura tcp_pcb, y si tiene éxito configura esa estructura de datos con la información del paquete SYN entrante, como número de secuencia y puerto de origen. Esa estructura de datos pasa a formar parte de la denominada cola de conexiones, en el estado SYN-RECEIVED. Por último, la función responde a la dirección IP de origen con un paquete SYN-ACK que permite continuar con el establecimiento de la conexión. Cuando se recibe el ACK correspondiente, la conexión cambia su estado a ESTABLISHED y se puede traspasar a la aplicación.

Si por el contrario no hay espacio cuando se recibe un paquete SYN, la respuesta dependerá de la configuración utilizada para la cola de conexiones. lwIP permite configurar la cola de conexiones en dos modos: backlog o cola de prioridad. El backlog es un tipo de cola que sigue el modelo implementado en sistemas Unix: cada paquete SYN crea una estructura de datos hasta que ya no haya memoria para crear más estructuras. En ese momento se descartan los paquetes SYN entrantes hasta que las estructuras presentes en la cola de conexiones vayan expirando, y por tanto se vaya liberando memoria. En caso de un ataque de inundación SYN, la estructura no expirará hasta que el número de reintentos SYN-ACK se haya cumplido, lo que se traduce en bastantes segundos para liberar una entrada (el número de segundos dependerá de la configuración de los argumentos de lwIP). Es fácil darse cuenta de que este modo no es útil en un dispositivo C2 con una memoria tan limitada, y más en caso de ataque ya que como se ha comentado en el capítulo 2, unos pocos paquetes enviados estratégicamente en tiempo pueden saturar la cola de conexiones permanentemente. Por lo tanto, si la cola de conexiones está llena, la función *tcp_listen_input()* devuelve error pero no se lo notifica al cliente, es decir, no envía un paquete RST, simplemente descarta el paquete SYN. El cliente deberá volver a intentar establecer la conexión y esperar que haya espacio en memoria.

La otra opción de gestión de la cola de conexiones consiste en una cola de prioridad, habilitada por defecto en lwIP. En este caso, si al llegar una solicitud la cola está llena, la función *tcp_alloc()* primero buscará si existen estructuras tcp_pcb en estado TIME_WAIT para eliminar. Si no hay, buscará la que tenga menor prioridad y sea más antigua. Este último caso es habitual si todas las conexiones tienen la misma prioridad (por defecto en lwIP), y por tanto sigue una estrategia LRU (Least Recently Used), es decir, la entrada más antigua se va sustituyendo cuando llega una nueva solicitud de conexión.

En definitiva, este modo se comporta mejor en caso de un ataque SYN, ya que forzaría al atacante a enviar más paquetes SYN e incluso en ese caso una solicitud legítima tendría opciones de entrar en

la cola, mientras que con el backlog no sería así. Este concepto se explicará de nuevo en el próximo capítulo para justificar por qué se ha comparado el rendimiento de las SYN cookies con la cola de prioridad y su política de reciclado de conexiones.

Por último, para los paquetes entrantes con el flag ACK activo, la función *tcp_listen_input()* responde con un RST ya que si no existe la conexión, no puede haber un ACK para esa conexión.

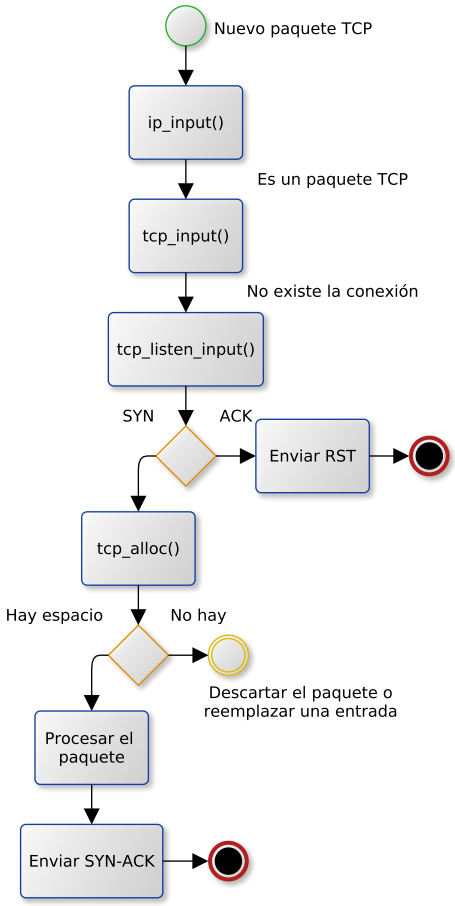


Figura 3.7 Procesamiento de las conexiones sin establecer en lwIP

Una vez analizado el procesamiento de las conexiones sin establecer, implementar las SYN cookies es sencillo. Si el paquete entrante es un SYN, no se intenta crear una estructura de datos en la cola de conexiones ni guardar la información del paquete en ella, simplemente se aplica el algoritmo de generación de SYN cookies y se responde al cliente con un SYN-ACK en el que se incluye como ISN esa cookie.

En cuanto a la validación, se utiliza la estructura condicional de la función *tcp_listen_input()* para paquetes ACK. En vez de responder con un RST, se aplica el algoritmo de validación de cookies, y si el resultado es correcto, se intenta crear una estructura de datos tcp_pcb con *tcp_alloc()*. Si hay espacio disponible para esta nueva estructura, se guarda la información correspondiente del paquete ACK y se

traspasa la conexión –ya en estado ESTABLISHED– a la aplicación. En caso contrario, se descarta la conexión y se responde con un paquete RST.

Por último, señalar que en este caso la funcionalidad ARP está habilitada (#define LWIP_ARP 1) para que el cliente sea capaz de comunicarse con el LPC1768 directamente.

3.2.4 Análisis de rendimiento

En esta sección se mide el efecto que tienen los algoritmos de generación de cookies (Linux, FreeBSD y la propuesta híbrida) en el rendimiento de red del LPC1768 con el objetivo de determinar si la propuesta híbrida es adecuada para los experimentos que se presentarán en el próximo capítulo.

El entorno de pruebas consta de los siguientes equipos:

- Equipo atacante: actúa como la fuente de la inundación SYN. El equipo es un Intel Pentium 4 de 3,4 GHz con 3 GB de RAM y Ubuntu 12.04.
- Equipo cliente: accede al servicio TCP de la víctima. El equipo es un Intel Core i5-560M de 2,67 GHz con 6 GB de RAM DDR3 y Ubuntu 14.04.
- Equipo víctima: implementa un servidor TCP y es el destino de la inundación SYN. El equipo es el dispositivo C2 (LPC1768).

Un conmutador de red 100Base-T proporciona la conectividad de red necesaria entre todos los equipos. Para generar el ataque de inundación SYN, se utiliza la herramienta *hping*. Para medir el rendimiento de red, se utiliza Iperf para crear un flujo de datos TCP entre los equipos cliente y víctima. La salida de esta herramienta contiene un informe de tiempo con la cantidad de datos transferidos y el rendimiento medido. Los resultados del rendimiento se recogen para su posterior análisis.

Finalmente, hay que seleccionar la tasa de inundación SYN. Se pueden clasificar los ataques SYN en dos categorías: ataques de alta y baja tasa [Nashat et al., 2008]. Se ha decidido establecer la tasa de paquetes SYN a 5000μs (200 paquetes/segundo), un ataque que podría clasificarse de baja tasa en entornos de ordenadores, pero para un dispositivo C2 y su cola de conexiones limitada se trata de una tasa significativa para determinar qué implementación de SYN cookies funciona mejor.

La tabla 3.8 muestra un resumen de los resultados del experimento.

Tabla 3.8 Rendimiento en Mbps

	Media	**Desviación estándar**
Sin ataque	33,72	0,042
Cookies FreeBSD	31,94	0,094
Cookies Linux	31,67	0,073
Cookies híbridas	31,91	0,081

Como se puede observar, hay una pérdida de rendimiento cuando el dispositivo está bajo un ataque SYN. Los resultados muestran que los algoritmos utilizados para el cálculo de cookies tienen diferentes

medias. Para evaluar si estas diferencias son significativas se utilizarán el valor p y el tamaño del efecto (d de Cohen) [Cohen, 1988].

El valor p es una medida estadística que se usa para determinar si los resultados se encuentran dentro del rango normal de valores para los eventos observados. Si el valor p de un conjunto de datos está por debajo de un valor de referencia (por lo general 0,05), se rechaza la hipótesis nula, es decir, las variables del experimento sí tienen un efecto significativo en los resultados.

Por otra parte, el tamaño del efecto es una forma sistemática de cuantificar el tamaño de la diferencia entre dos grupos. Si los dos grupos tienen el mismo número de muestras, el tamaño del efecto se calcula restando las medias y dividiendo el resultado por la desviación estándar agrupada. Cohen utilizó los valores 0,2, 0,5 y 0,8 para representar la importancia relativa de los tamaños del efecto. Por otra parte, aunque no se toma en cuenta el tamaño de la muestra, si éste es pequeño (generalmente por debajo de 20), el tamaño del efecto puede estar algo sesgado [Lakens, 2013]. Se han obtenido 30 muestras por cada grupo (tabla 3.8) y el intervalo de confianza (CI en inglés) es del 95 %.

Entre las implementaciones de SYN cookies (N = 30), hubo una diferencia estadísticamente significativa entre Linux (M = 31,67, SD = 0,073) y FreeBSD (M = 31,94, SD = 0,094), t (58) = 12,4256, p $\leq$ 0,05, 95 % CI [-0,31350, -0,22650]. Por lo tanto, se rechaza la hipótesis nula de igualdad de medias para el rendimiento de red entre las implementaciones de Linux y FreeBSD. Además, el tamaño del efecto (d = 3,2) sugiere que la relevancia del efecto es muy alta. De la misma manera, se obtuvo una diferencia estadísticamente significativa entre Linux (M = 31,67, SD = 0,073) y la propuesta híbrida (M = 31,91, SD = 0,081), t (58) = 12,0554, p $\leq$ 0,05, 95 % CI [-0.27985, -0.20015]. Por consiguiente, se rechaza la hipótesis nula de igualdad de medias entre esas implementaciones. El tamaño del efecto (d = 3,11) también sugiere una relevancia práctica muy alta. Por último, no hubo diferencias estadísticamente significativas entre FreeBSD (M = 31,94, SD = 0,094) y la propuesta híbrida (M = 31,91, SD = 0,081), t (58) = 1,3242 p $\geq$ 0,05, 95 % CI [-0,01535, 0,07535]. Por lo tanto, no se puede rechazar la hipótesis nula. Además, el tamaño del efecto (d = 0,33) sugiere una relevancia práctica entre baja y moderada.

El análisis de rendimiento demuestra que la implementación de Linux podría ser más adecuada para dispositivos de recursos no limitados debido al tiempo de cómputo involucrado en el cálculo de dos funciones SHA-1 por paquete SYN. Por otro lado, FreeBSD y la implementación híbrida parecen más adecuadas para dispositivos de recursos limitados (como los dispositivos C2) porque utilizan SipHash, un algoritmo de cifrado optimizado para el rendimiento. En este caso, la función criptográfica es la principal responsable de la variación estadística.

En resumen, la propuesta híbrida hace que la reutilización de cookies sea más difícil, mientras que el uso de dos cómputos SipHash no es estadísticamente significativo respecto a utilizar uno (FreeBSD). Esta implementación híbrida se utilizará en el estudio experimental que se presentará en el próximo capítulo.

3.3 Conclusión

La función principal de un balanceador de carga es distribuir la carga de tráfico entre los servidores que forman el clúster con el fin de aumentar la escalabilidad del servicio. En los esquemas de balanceo tradicionales, cada servidor responde al balanceador de carga con los datos requeridos, y luego el balanceador es el encargado de reenviar esa respuesta al cliente. Esta solución presenta un inconveniente, y es que el tráfico de vuelta tiene que viajar por el balanceador de carga, y por tanto el rendimiento de red sufre en consecuencia [Bourke, 2001]. De ahí la necesidad de DSR, que modifica el flujo del tráfico al permitir que el servidor responda directamente al cliente. Aunque este esquema es el más apropiado para clústeres de dispositivos C2, ya que alivia al balanceador de gestionar una mayor carga de tráfico, el hecho de que el tráfico de retorno no pase por el balanceador también impide que éste pueda aplicar mecanismos de seguridad avanzados ante ataques SYN.

El balanceador de carga podría limitar las conexiones por dirección IP o por el número de paquetes SYN entrantes, pero para conseguir una protección global del clúster es necesario aplicar también defensas específicas en los servidores. Tras analizar las dos implementaciones de SYN cookies existentes en sistemas operativos de tipo Unix, se demuestra que la implementación de Linux es más lenta pero más robusta que la de FreeBSD. De esta forma, se propone una implementación híbrida que combina el rendimiento de FreeBSD y la robustez de Linux, y se aplica a dispositivos C2. El análisis estadístico confirma que el diseño de cualquier pieza particular de software para un dispositivo de recursos limitados debe ser cuidadosamente diseñado por el impacto no trivial que los cómputos, como las funciones criptográficas, tienen en su rendimiento de red.

Por último, merece la pena destacar que las implementaciones que se han realizado (balanceador, servidores, y SYN cookies) son independientes de la plataforma utilizada (LPC1768), y por tanto podrían trasladarse a otros dispositivos C2. Las modificaciones solo se han desarrollado a nivel de lwIP. En consecuencia, para migrar el código a otra plataforma simplemente habría que utilizar la misma versión de lwIP y ajustar las llamadas al sistema operativo subyacente, como la creación de semáforos o hilos de ejecución.

CAPÍTULO

4

Experimentación y resultados

Este capítulo describe el banco de pruebas y la metodología de experimentación utilizada para obtener medidas reales del rendimiento de red en un clúster de dispositivos C2 con tres balanceadores de carga diferentes. En primer lugar, conviene recordar la hipótesis planteada en el capítulo 1: "*Un clúster homogéneo de dispositivos de recursos limitados puede distribuir y procesar conexiones TCP concurrentes sin reducir su rendimiento de red respecto a balanceadores de carga de recursos no limitados*". Los resultados de la experimentación determinarán si la hipótesis de investigación es aceptada o rechazada.

De la hipótesis y el objetivo principal de investigación pueden distinguirse dos experimentos independientes. Un esquema de balanceo DSR es el más indicado en un clúster de dispositivos C2 ya que solo procesa los paquetes del cliente (menor uso de recursos), al contrario que el resto de esquemas de balanceo en los que el tráfico en la dirección del servidor al cliente también pasa por el balanceador. En este caso hay que validar que un dispositivo C2 puede balancear el tráfico en un esquema DSR sin reducir el rendimiento de red de manera significativa respecto a balanceadores de carga con capacidades superiores a las definidas en el RFC 7228. La primera parte del capítulo abordará esta experimentación para buscar factores que expliquen los resultados obtenidos.

Por otra parte, un esquema DSR no puede mitigar eficazmente ataques SYN en el balanceador ya que el tráfico de retorno no pasa por él. La segunda parte del capítulo determinará si una estrategia de SYN cookies en los servidores C2 aporta mejores resultados que una estrategia de reemplazo de conexiones semiabiertas.

4.1 Balanceo de carga

En esta sección se describe una configuración que crea un entorno de red aislado que proporciona una capacidad de control de los factores importantes de un sistema de balanceo de carga. Las conexiones se generan desde el equipo cliente y se dirigen a un balanceador de carga (más adelante se describen en detalle las características de los 3 balanceadores usados), cuya función será distribuir el tráfico de red a los servidores C2 que forman el clúster, también conectados al mismo segmento de red.

Esta configuración y sus posteriores experimentos permiten evaluar el rendimiento de cada balanceador a medida que crece la demanda del servicio ofrecido. Los resultados obtenidos demostrarán si un dispositivo C2 puede actuar como balanceador de carga para el tráfico generado en el experimento, y si su rendimiento es comparable al de los otros 2 balanceadores, cuyas capacidades de cómputo y almacenamiento son muy superiores.

Al igual que en el capítulo 3, se ha utilizado el rendimiento de red (*network throughput* en inglés) como métrica de evaluación. En redes de telecomunicación, el rendimiento de red es la métrica más utilizada para determinar la calidad de una conexión de red [Darst y Ramanathan, 1999], y mide la tasa efectiva de datos transmitidos en un período de tiempo determinado. Los experimentos medirán el rendimiento de red en la dirección del cliente al servidor. El cliente tratará de enviar paquetes TCP al servidor lo más rápido posible durante un período determinado. Por lo tanto, la velocidad de envío la determinará la capacidad del servidor C2 de ir reconociendo esos paquetes entrantes.

4.1.1 Planteamiento del problema

Antes de comenzar el diseño de la fase de experimentación es necesario plantear el problema que se quiere estudiar. Este experimento pretende aportar información en relación a la siguiente pregunta: ¿en qué medida influyen los balanceadores de carga disponibles, incluido el dispositivo C2, en el rendimiento de red del servicio proporcionado por el clúster de dispositivos C2? Para responder a esta pregunta es necesario:

- Definir los elementos hardware y software empleados.
- Diseñar una red aislada que permita desplegar el clúster de dispositivos C2 y el cliente en el mismo segmento de red.
- Definir un método para generar múltiples conexiones simultáneas desde el cliente.
- Investigar la degradación del servicio ofrecido por el clúster ante el aumento de la demanda.
- Investigar cómo diferentes balanceadores de carga degradan o mejoran el rendimiento de red.
- Investigar si el número de clientes o servidores afecta de manera significativa al rendimiento de red del clúster.

A modo de avance, es preciso destacar que la experimentación constará de dos fases, y su razón se explicará en la sección 4.1.4.

4.1.2 Configuración del experimento: fase I

Esta sección presenta la configuración detallada de la primera fase del experimento. En primer lugar, se presenta la configuración hardware y de red. A continuación, se procede a explicar el software necesario para llevar a cabo el experimento.

4.1.2.1 Configuración hardware

En este apartado se detallan los equipos y el conmutador de red que constituyen la parte hardware del experimento. Los equipos utilizados son: i) tres balanceadores de carga, ii) cinco servidores que forman el clúster, y iii) el equipo cliente que accede al servicio TCP proporcionado por el clúster.

A continuación, se presenta un resumen que cubre las características más importantes de los equipos mencionados.

1. Balanceador de carga I:
    - Sistema operativo: mbed OS
    - Procesador: ARM Cortex M3
    - Núcleos: 1
    - Velocidad procesador: 96MHz
    - Modelo: LPC1768
    - Memoria RAM: 64kB (32kB para Ethernet)
    - Controlador de acceso al medio: Ethernet 10/100 Mbps
2. Balanceador de carga II:
    - Sistema operativo: Raspbian con RT Preempt
    - Procesador: ARM11
    - Núcleos: 1
    - Velocidad procesador: 700MHz
    - Modelo: Raspberry Pi B
    - Memoria RAM: 512MB
    - Controlador de acceso al medio: Ethernet 10/100 Mbps
3. Balanceador de carga III:
    - Sistema operativo: Ubuntu 12.04
    - Procesador: Intel Pentium 4
    - Núcleos: 1
    - Velocidad procesador: 3.4GHz
    - Modelo: Dell Dimension 5150
    - Memoria RAM: 3GB DDR2 SDRAM
    - Tarjeta de red: Integrada Ethernet 10/100 Mbps
4. Servidores:

- Sistema operativo: mbed OS
- Procesador: ARM Cortex M3
- Núcleos: 1
- Velocidad procesador: 96MHz
- Modelo: LPC1768
- Memoria RAM: 64kB (32kB para Ethernet)
- Controlador de acceso al medio: Ethernet 10/100 Mbps

5. Cliente:

   - Sistema operativo: Ubuntu 14.04
   - Procesador: Core i5 430M
   - Núcleos: 4
   - Velocidad procesador: 2.26GHz
   - Modelo: Dell Vostro 3300
   - Memoria RAM: 6GB DDR3 SDRAM
   - Tarjeta de red integrada: Ethernet 10/100/1000 Mbps

6. Conmutador de red:

   - Modelo: D-Link DES-1008F
   - Gestionable: No
   - Velocidad: 10/100 Mbps (10/100BaseT)
   - Puertos: 8 puertos RJ45
   - Auto-Negociación: Velocidad, dúplex y control de flujo

En cuanto a la Raspberry Pi, se utiliza una implementación de Raspbian con el núcleo en tiempo real. El parche RT Preempt ofrece límites deterministas en el retardo y en la variación del mismo, similar a mbed OS que también es un sistema operativo en tiempo real. Es compatible con los modelos de Raspberry Pi 2B, B+, A+, B, y A, y se basa en la versión de Raspbian *2015-02-16-raspbian-wheezy* con el núcleo predeterminado sustituido a *3.18.9-rt5-v7+*, además de algunas configuraciones adicionales.

El núcleo predeterminado en la versión 3.18 de Raspbian está configurado con la opción PREEMPT. Esto significa que el núcleo de Linux solo permitirá a un proceso adelantarse a otro bajo ciertas circunstancias:

- Cuando la CPU está ejecutando código en modo de usuario.
- Cuando el código del núcleo regresa de una llamada al sistema o de una interrupción al espacio de usuario.

- Cuando el código del núcleo se bloquea en un mutex (recordar que un mecanismo de exclusión mutua se usa para evitar el ingreso a secciones críticas por más de un hilo a la vez), o de manera explícita pasa el control a otro proceso.

Si el código del núcleo se ejecuta cuando algún evento tiene lugar, el proceso no puede adelantarse al código de núcleo en ejecución hasta que el código del núcleo pase de forma explícita el control. En el peor de los casos la latencia es de cientos de microsegundos. El parche RT Preempt reduce ese tiempo a decenas de microsegundos ya que reestructura las funciones críticas del núcleo para evitar bloqueos, bucles o tiempos de espera prolongados.

En definitiva, se comparan tres balanceadores cuyas capacidades aumentan progresivamente. Desde un microcontrolador de recursos limitados que utiliza un sistema operativo en tiempo real, a un ordenador denominado de placa única corriendo una versión de Linux en tiempo real, para acabar con un ordenador de sobremesa con mayor velocidad de CPU pero sin implementar el parche Linux de tiempo real.

4.1.2.2 Configuración de red

A continuación se describe la configuración de red para conectar los equipos. Los equipos se conectan mediante un cable de red RJ45 a uno de los puertos del conmutador de red. El conmutador de red proporciona la conectividad de red necesaria entre todos los equipos, formando así una red de área local (LAN) entre los equipos conectados. Conviene señalar que solo uno de los balanceadores estará conectado en cada configuración, es decir, al mismo tiempo estarán conectados el equipo cliente, un balanceador, y hasta 5 servidores C2. Una vez conectados al conmutador, se deben configurar todos los equipos con direcciones IP y máscaras que pertenezcan a la misma subred, a fin de que los diferentes equipos en esa subred puedan comunicarse entre sí. En este caso se utiliza la subred 10.48.1.x. Es necesario recordar que los servidores C2 estarán configurados con una misma dirección IP, al igual que el balanceador C2.

Antes de realizar los experimentos, se comprueba la conectividad de red en todos los equipos utilizando la herramienta *ping* para enviar mensajes ICMP entre los equipos.

4.1.2.3 Configuración software

La configuración software de los dispositivos clase 2, tanto el balanceador como los servidores, ha quedado explicada en el capítulo anterior. De esta forma, se analizará en primer lugar el software utilizado para balancear tráfico en los equipos Linux (balanceadores de carga II y III).

El Servidor Virtual de Linux (LVS por sus siglas en inglés) es un servidor virtual construido sobre un clúster de servidores reales (equipos físicos), con el balanceador de carga ejecutándose en el sistema operativo Linux. Como ya se ha comentado en repetidas ocasiones, la arquitectura del clúster de servidores es totalmente transparente para los clientes finales. Es decir, para los clientes existirá un único servidor virtual de alto rendimiento (el balanceador), que se encargará de distribuir la carga entre los servidores reales.

En Ubuntu, LVS viene integrado en el núcleo desde la versión 6.06, y por tanto no es necesario recompilar el núcleo para utilizarlo, basta con instalar la herramienta *ipvsadm* para configurar el balanceo de carga. Esta herramienta de línea de comandos permite administrar y tener control total de LVS en tiempo de ejecución, como añadir, editar y eliminar servidores reales, cambiar los algoritmos de balanceo, o exportar e importar la configuración activa.

En la configuración DSR, la dirección virtual (VIP) es compartida por el balanceador y los servidores. De esta manera el balanceador recibe las peticiones y las envía a los servidores que procesan esas peticiones y dan servicio directamente a los clientes. Por lo tanto, el balanceador debe tener configurada la IP virtual y habilitado el reenvío IP. De esta forma, si el tráfico llega a través de una interfaz que coincida con una subred de otra interfaz de red, permite reenviar el tráfico a la otra interfaz de red.

A continuación se muestra cómo configurar la IP virtual y el balanceador de carga:

Listado 4.1 Configuración balanceador

```
#configurar la interfaz de red virtual con la dirección VIP
ifconfig eth0:0 10.48.1.32 netmask 255.255.255.0
#visualizar el archivo de configuración del núcleo
nano /etc/sysctl.conf
#editar el parámetro de configuración
net.ipv4.ip_forward = 1
#utilizar la nueva configuración
sysctl -p
#configurar el LVS
ipvsadm -C
ipvsadm -A -t 10.48.1.32:45678 -s rr
ipvsadm -a -t 10.48.1.32:45678 -r 10.48.1.35:45678
ipvsadm -a -t 10.48.1.32:45678 -r 10.48.1.36:45678
```

donde:

- -C limpia la tabla del LVS. Como ya se explicó en el capítulo anterior, el balanceador debe configurar una tabla o estructura de datos donde almacenar las conexiones activas. Este comando se ejecuta antes de iniciar una nueva configuración.
- -A permite agregar un servicio virtual. Además, hay que especificar si es un servicio UDP o TCP, la dirección virtual y el algoritmo de balanceo.
- -t se utiliza para indicar que se proporcionará un servicio TCP. A continuación se debe especificar la dirección IP virtual del servicio, pudiendo también especificar el número de puerto del mismo, de la forma *IP:puerto*.
- -s se utiliza para especificar el algoritmo de balanceo a utilizar. Entre los valores admitidos, el más utilizado es rr (Round-Robin), que es precisamente el que se utiliza.
- -a permite agregar un servidor real a un servicio virtual. Se debe especificar la dirección virtual del servicio al que se agrega, y su dirección real.

- -r se utiliza para especificar la dirección IP de un servidor real, y opcionalmente puede agregarse el número de puerto.

Por supuesto, existen más comandos que permiten configuraciones más avanzadas, como especificar el peso de un servidor real, pero para el experimento es necesario utilizar una configuración que corresponda con la implementación hecha en el dispositivo C2.

Como se ha visto, al agregar un servidor real (-a) se debe especificar la dirección IP real del servidor. De esta forma, cuando llegue un paquete al balanceador, éste seleccionará un servidor, mirará la tabla ARP local, y si no existe una entrada, enviará una petición ARP a esa dirección IP especificada con el comando -r. Esto es un problema ya que como se explicó en el capítulo anterior, los servidores C2 tienen todos la misma IP, la VIP del clúster. Entonces, ¿cómo se pueden asociar esas IPs a los servidores C2? La solución pasa por utilizar direcciones IP no asignadas en la subred, y asociar de manera local en el balanceador esas direcciones IP a las direcciones MAC de los servidores C2.

Cuando se trata con situaciones anormales inducidas por ARP, es posible añadir entradas estáticas en la tabla ARP local. Cuando una dirección MAC de destino se encuentra en la tabla ARP, no hay necesidad de enviar peticiones ARP, es decir, no sería necesario enviar peticiones ARP a las IPs agregadas con el comando -r. A continuación, se muestra cómo se añade en Linux una entrada ARP estática a la tabla ARP local. Para ello se utilizan las dos direcciones IP de los servidores reales que se han usado en el ejemplo anterior de configuración del balanceador.

Listado 4.2 Configuración ARP estática

```
sudo arp -s 10.48.1.35 00:02:f7:f2:30:f1
sudo arp -s 10.48.1.36 00:02:f7:f2:3c:76
```

Este comando almacena en la tabla ARP local las direcciones MAC asociadas a los servidores del clúster, y les asigna direcciones IP que en realidad no existen en la red. Una vez configurada una entrada ARP estática, se puede comprobar el contenido de la tabla ARP con el siguiente comando:

Listado 4.3 Verificación tabla ARP

```
arp -a -n

? (10.48.1.35) at 00:02:f7:f2:30:f1 [ether] PERM on eth0
? (10.48.1.36) at 00:02:f7:f2:3c:76 [ether] PERM on eth0
```

Como se puede ver, las entradas ARP configuradas se muestran correctamente, marcadas como permanentes en la tabla ARP. Para eliminar estas entradas permanentes se puede utilizar el comando -d, o bien reiniciar o apagar el equipo.

Por último, en cuanto al software del equipo cliente, se utiliza el programa Iperf para medir el rendimiento de red entre el equipo cliente y el clúster. Iperf es una herramienta de línea de comandos que se utiliza en el diagnóstico de redes informáticas. Iperf fue desarrollado por el Distributed Applications Support Team (DAST) en el National Laboratory for Applied Network Research (NLANR) y está escrito en C++. Esta herramienta mide el rendimiento de red máximo que un servidor puede soportar. Cuando

se utiliza en su modo TCP, Iperf mide el rendimiento de carga útil, es decir, el número de bytes de datos que es capaz de enviar en un período de tiempo.

A continuación se muestra un ejemplo de un comando Iperf desde un cliente TCP:

Listado 4.4 Iperf

```
iperf -c 10.48.1.32 -p 45678 -t 30 -f m
```

donde:

- -c indica el modo cliente y la dirección destino IP (en este caso 10.48.1.32, la dirección VIP del clúster).
- -p indica el puerto del servicio al que conectarse.
- -t indica el tiempo en segundos de la conexión. La duración por defecto es de 10 segundos si no se especifica un valor diferente con este argumento.
- -f indica el formato en el que se notifica el rendimiento de red, en este caso m indica Mbits.

La salida de un comando Iperf contiene un informe con la cantidad de datos transmitidos y el rendimiento medio de la conexión (en este caso Mbits/s). La inserción de marcas de tiempo en la salida es opcional (parámetro -i).

4.1.3 Definición del experimento: fase I

En esta sección se describe la metodología utilizada para medir el rendimiento de red de un clúster de dispositivos C2. En el experimento se manipulan tres variables independientes (tipo de balanceador, número de clientes y número de servidores) para determinar su efecto en la variable dependiente (rendimiento de red). Como se ha explicado anteriormente, se define rendimiento de red como el número de bytes que se envían de manera correcta por unidad de tiempo. En este caso solo se mide el rendimiento experimentado por el cliente.

Tal y como ya se ha descrito, se dispone de tres balanceadores de carga con capacidades muy diferentes (variable estadística definida como *LB*). La idea es experimentar y comparar el funcionamiento de esos tres balanceadores aumentando gradualmente el número de servidores y clientes.

Como su nombre indica, un clúster está compuesto por un grupo de servidores conectados que trabajan juntos, lo que significa que tiene que estar formado al menos por dos servidores. En este caso se dispone de cinco servidores C2, por lo que el número de servidores (variable estadística definida como *núm_servidores*) empezará en 2 (mínimo) y se irá agregando un servidor más al clúster hasta llegar a 5 (máximo).

Por último, como ya se explicó en el capítulo anterior, cada servidor C2 desarrollado acepta hasta 4 conexiones TCP simultáneas. Validar una aplicación o servicio en un entorno concurrente es crucial si alguna vez el clúster va a estar funcionando con más de un usuario. A la hora de medir el rendimiento

de servidores, los desarrolladores utilizan las denominadas "*pruebas de estrés*" o "*pruebas de carga*", que básicamente miden el rendimiento de red con el mayor número de conexiones concurrentes que esperan recibir durante el pico de uso del servidor. Se denominan conexiones simultáneas o concurrentes a conexiones que se producen al mismo tiempo. De esta forma, para medir cómo se comporta el balanceador ante conexiones concurrentes se irán incrementando progresivamente el número de conexiones (variable estadística definida como *núm_clientes*) hasta llegar al número máximo de conexiones que un servidor es capaz de procesar. Conviene señalar que se ejecutará una conexión por cada servidor, es decir, si hay 2 servidores, se ejecutan 2 conexiones concurrentes desde el cliente que el balanceador repartirá para que cada servidor procese una conexión. Por tanto, en el caso de 2 servidores, el límite serán 8 conexiones (2 servidores x 4 conexiones concurrentes).

A continuación se muestran las configuraciones del experimento por cada balanceador:

- 2 servidores: 2, 4, 6 y 8 conexiones concurrentes.
- 3 servidores: 3, 6, 9 y 12 conexiones concurrentes.
- 4 servidores: 4, 8, 12 y 16 conexiones concurrentes.
- 5 servidores: 5, 10, 15 y 20 conexiones concurrentes.

Una vez presentadas las diferentes configuraciones del experimento, es preciso definir cómo se generan las conexiones desde el mismo equipo cliente. Por defecto, Iperf ejecuta una sola prueba de rendimiento de red utilizando una única conexión. Sin embargo, en este caso es necesario: i) ejecutar el mismo experimento múltiples veces, y ii) ejecutar conexiones en paralelo.

En Iperf se pueden ejecutar conexiones concurrentes para simular más de un usuario con el parámetro -P, seguido por el número de conexiones a crear. En cuanto a ejecutar el experimento varias veces, se ha desarrollado un script Bash para automatizar la ejecución de los experimentos. Un script Bash es un archivo de texto plano que contiene una serie de comandos ejecutables por un sistema operativo de tipo Unix. El script asume que hay un solo servidor en funcionamiento (dirección IP y puerto destino del clúster) y ejecuta múltiples clientes en paralelo (diferentes puertos de origen). Entre cada ejecución, el script entra en un estado de inactividad por un período breve.

Por defecto, el script utiliza los siguientes parámetros:

- Dirección IP del clúster (-c), en este caso 10.48.1.32.
- Número de puerto del servidor (-p), en este caso 45678.
- Número de instancias del cliente (-P), dependiente de las configuraciones antes presentadas.
- Duración de cada prueba (-t), en este caso 30 segundos.
- Tiempo de inactividad entre las pruebas, en este caso 1 segundo.
- Número de pruebas a ejecutar.

A continuación, se debe determinar el número de elementos que formarán la muestra del experimento (argumento *Número de pruebas a ejecutar*). Corder y Foreman [2009] destacan que "*el tamaño mínimo de la muestra para el uso de una prueba estadística paramétrica varía entre los textos. Por ejemplo, Pett [1997] y Salkind [2004] observaron que la mayoría de los investigadores sugieren N > 30. Warner [2008] alentó a considerar N > 20 como mínimo y N > 10 por grupo como mínimo absoluto*". Hogg y Tanis [2006] también proponen que el tamaño debería ser "*mayor que 25 o 30*".

Ninguna cantidad de muestras proporcionará una certeza absoluta, pero por lo general se afirma que para que una muestra sea representativa debe contener las características relevantes del experimento estudiado. Se considera que 30 muestras por experimento (N) son suficientes para reducir el efecto de la variación no controlada, y por tanto ser representativas. A partir de los datos obtenidos se realizan las pruebas estadísticas que permitan generalizar los resultados con una mínima probabilidad de error ($\alpha \leq 5\,\%$).

Por último, y para facilitar el análisis estadístico, los resultados se guardan en un archivo de texto. Una vez obtenido el fichero con todas las pruebas realizadas, se procesa con un script en Python para convertirlo en un fichero CSV (de las siglas en inglés Comma Separated Values). Este archivo contiene los datos para el análisis estadístico que se presenta a continuación.

4.1.4 Resultados y análisis: fase I

En primer lugar, conviene señalar que para el análisis de los resultados se ha utilizado el programa SPSS de IBM, que se compone de un conjunto de herramientas para el tratamiento de datos y el cálculo de una amplia variedad de estadísticas. La decisión de utilizar esta herramienta se basa en que es una opción ampliamente utilizada en la comunidad científica, de fácil uso, y permite importar datos en CSV para su análisis.

En cuanto al tipo de análisis, se realiza un ANOVA factorial (*LB* x *núm_servidores* x *núm_clientes*). En estadística, el análisis de la varianza (ANOVA) permite determinar si las medias de dos o más grupos tienen diferencias significativas, o por el contrario puede suponerse que no difieren.

Los modelos factoriales permiten evaluar el efecto individual y conjunto de dos o más factores sobre la misma variable dependiente. En un modelo de tres factores, los efectos de interés son siete: los tres efectos principales o individuales, los tres efectos de las interacciones dobles (uno por cada interacción entre cada dos factores), y el efecto de la interacción triple (entre los tres factores). En este caso, el ANOVA factorial permite analizar si el rendimiento de red (variable dependiente) de un clúster formado por dispositivos C2 es diferente utilizando tres tipos de balanceadores de carga (efecto del primer factor) y, al mismo tiempo, si el número de servidores y clientes tienen un impacto significativo en los resultados (efectos del segundo factor). Asimismo, se puede determinar si la interacción entre los factores afecta al rendimiento de red.

Así pues, para cada efecto existe una hipótesis, y la estrategia para contrastarla consiste en obtener un estadístico, llamado F, que refleja el grado de parecido existente entre las medias que se están comparando. Cuanto más diferentes sean las medias, mayor será el valor de F. Si el nivel de significación

(Sig.) asociado al estadístico F es menor que 0,05, se rechaza la hipótesis de igualdad de medias y se concluye que las medias comparadas son diferentes. En caso contrario, no se puede rechazar la hipótesis de igualdad y por tanto no es posible afirmar que los grupos comparados difieran en sus promedios. Cabe destacar que se usa el valor 0,05 (α) ya que en las publicaciones científicas es el nivel de significación más utilizado (equivalente a decir que se acepta un error inferior al 5 %).

En un ANOVA factorial existe una hipótesis nula por cada factor y por cada posible interacción. La hipótesis nula referida a un factor afirma que las medias de los grupos para ese factor son iguales. De igual forma, la hipótesis referida al efecto de una interacción afirma que tal efecto es nulo.

Un análisis ANOVA requiere el cumplimiento de los siguientes supuestos:

- Las distribuciones de probabilidad de la variable dependiente, para cada factor en este caso, deben ser normales (*normalidad*).
- La varianza entre los grupos debe ser aproximadamente igual (*homogeneidad*).
- Las muestras sobre las que se aplican los tratamientos deben ser independientes entre sí (*independencia*).

Estos supuestos pueden ser probados utilizando herramientas estadísticas. El supuesto de homogeneidad de varianza se puede validar mediante la prueba de Levene, y la normalidad de la distribución usando gráficos o pruebas como la de Shapiro-Wilk. El supuesto de independencia puede determinarse a partir del diseño del estudio, que deberá ratificar que el valor de una muestra no esté relacionado con cualquier otra muestra. En este caso, se cumple el supuesto de independencia ya que los experimentos han sido diseñados y ejecutados de tal forma que las muestras obtenidas no se veían influidas por una medida anterior, y por tanto, son independientes entre sí.

En cuanto a los supuestos de homogeneidad y normalidad, si el diseño del experimento se ha hecho de forma apropiada, no habrá dos conjuntos de datos que tengan la misma varianza, por lo que con un número de muestras suficientemente grande es muy probable que la prueba de Levene falle. En la sección 4.2 se analizará este supuesto para el caso de las SYN cookies. Por otra parte, cuando se toma un gran número de muestras aleatorias de una población, las medias de esas muestras presentan una distribución aproximadamente normal incluso cuando la población no es normal. Estudios de simulación han demostrado que la tasa de falsos positivos no se ve muy afectada por la violación de estos supuestos [Lix et al., 1996]. En definitiva, un ANOVA soporta violaciones a los supuestos de homogeneidad y normalidad, y por tanto no se analizarán en esta sección.

Es preciso destacar también que en este caso se ha realizado un análisis exploratorio, y no confirmatorio, por lo que cumplir estos dos supuestos no es imprescindible. En un análisis exploratorio se busca deducir el comportamiento global de los datos. De esta forma, en vez de utilizar hipótesis preestablecidas, se analizan los datos y después se intenta mejorar la calidad del análisis poniendo a prueba las hipótesis resultantes por medio del análisis confirmatorio. El objetivo de este experimento es identificar qué parámetros parecen afectar más al rendimiento de red en un clúster de dispositivos C2.

Para llevar a cabo el ANOVA factorial con SPSS se utilizarán las especificaciones del Modelo Lineal General Univariante. A continuación se muestran los resultados obtenidos:

Tabla 4.1 Factores inter-sujetos

Factor	Valor	N
LB	Balanceador de carga I	4200
	Balanceador de carga II	4200
	Balanceador de carga III	4200
núm_servidores	2	1800
	3	2700
	4	3600
	5	4500
núm_clientes	2	180
	3	270
	4	720
	5	450
	6	1080
	8	1440
	9	810
	10	900
	12	2160
	15	1350
	16	1440
	20	1800

La tabla 4.1 muestra el nombre de las variables independientes (factores), sus valores o identificadores, y el número de muestras que hay en cada grupo. Hay 4200 muestras por cada balanceador (*LB*), y se observa cómo se distribuyen esas muestras con diferentes configuraciones de número de servidores y clientes.

Por otra parte, la tabla 4.2 contiene la información referida a los siete efectos del modelo de tres factores respecto a la variable dependiente (rendimiento de red). Para este análisis es necesario fijarse en las dos últimas columnas de la tabla.

La fila *Modelo corregido* tiene en cuenta todos los efectos del modelo, es decir, los siete efectos de interés y el de intersección. El nivel de significación (0,000) determina que el modelo explica una parte significativa de la variación observada en la variable dependiente. El coeficiente de determinación (R^2 = 0,987), que se obtiene dividiendo la suma de cuadrados de la fila *Modelo corregido* entre la suma de cuadrados de la fila *Total corregido*, indica que los tres efectos incluidos en el modelo (*LB*, *núm_servidores* y *núm_clientes*) explican el 98,7 % de la variación en la variable dependiente. Con este dato se puede determinar que casi toda la varianza se debe al modelo.

La fila *Intersección* informa sobre la constante del modelo. Las tres filas siguientes recogen los efectos principales, es decir, los efectos individuales de los tres factores incluidos en el modelo: *LB*,

Tabla 4.2 Efectos inter-sujetos

Fuente	Suma de cuadrados	gl	Media cuadrática	F	Sig.	η_p^2
Modelo corregido	300924,888[a]	47	6402,657	20439,653	0,000	0,987
Intersección	1198137,882	1	1198137,882	3824899,869	0,000	0,997
LB	35805,544	2	17902,772	57152,279	0,000	0,901
núm_servidores	2935,779	2	1467,890	4686,048	0,000	0,427
núm_clientes	195433,674	10	19543,367	62389,667	0,000	0,980
LB * núm_servidores	3711,844	4	927,961	2962,395	0,000	0,486
LB * núm_clientes	34728,261	20	1736,413	5543,274	0,000	0,898
núm_servidores * núm_clientes	1,587	2	0,794	2,533	0,079	0,000
LB * núm_servidores * núm_clientes	528,086	4	132,021	421,461	0,000	0,118
Error	3931,875	12552	0,313			
Total	1394393,997	12600				
Total corregido	304856,762	12599				

R^2 = 0,987 (R^2 corregido = 0,987)

núm_servidores y *núm_clientes*. Los niveles de significación (Sig.) indican que los grupos definidos por los tres factores tienen rendimientos medios de red significativamente diferentes (Sig. = 0,000 < 0,05). El estadístico *eta cuadrado parcial* ($\eta_p{}^2$) es una medida del tamaño del efecto en un ANOVA. En concreto, es una estimación de la proporción de variación de la variable dependiente que está explicada por cada efecto. Según este estadístico, el parámetro más significativo es el número de clientes (0,980), seguido del balanceador utilizado (0,901).

Las siguientes filas contienen información sobre el efecto de las interacciones entre esos factores. Los estadísticos F correspondientes llevan asociados un nivel de significación de 0,000, lo que significa que esas interacciones tienen también un efecto significativo sobre el rendimiento de red. La única excepción es la interacción núm_servidores-núm_clientes, cuyo nivel de significación (Sig. = 0,079 > 0,05) y tamaño del efecto (η_p^2 = 0,000 < 0,01) indican que las diferencias en el rendimiento de red que se dan con un número determinado de servidores no se pueden considerar significativas respecto al número de clientes concurrentes.

La fila *Error* ofrece información relacionada con el error del ANOVA. A continuación, la fila *Total* muestra la suma de los cuadrados de la variable dependiente. Por último, la fila *Total corregido* recoge la variación debida a cada efecto teniendo en cuenta el error.

A la vista de los resultados, el número de clientes –o conexiones concurrentes– es el factor más significativo del ANOVA factorial respecto al rendimiento de red del clúster. Teniendo en cuenta que las conexiones se ejecutan desde un mismo equipo cliente, y por tanto desde la misma interfaz de red, es necesario realizar un segundo experimento en el que cada conexión se ejecute desde un equipo cliente diferente, con el objetivo de eliminar la posibilidad de que generar múltiples conexiones desde un único equipo cliente pueda influir en las medidas del experimento.

4.1.5 Configuración del experimento: fase II

Esta sección presenta la configuración de la segunda fase del experimento en los mismos términos que la fase I. En primer lugar se presenta la configuración hardware y de red, y a continuación se explica el software de utilidad necesario para ejecutar el experimento.

4.1.5.1 Configuración hardware

En esta parte se detallan los equipos y los conmutadores de red que constituyen la parte hardware del experimento. Se utilizan varios equipos en el banco de pruebas: i) los tres balanceadores de carga, ii) los cinco servidores que forman el clúster, y iii) los 20 equipos cliente que accederán al servicio TCP proporcionado por el clúster.

A continuación, se presenta un resumen que cubre las características más importantes de los equipos mencionados.

1. Balanceador de carga I:
    - Sistema operativo: mbed OS
    - Procesador: ARM Cortex M3
    - Núcleos: 1
    - Velocidad procesador: 96MHz
    - Modelo: LPC1768
    - Memoria RAM: 64kB (32kB para Ethernet)
    - Controlador de acceso al medio: Ethernet 10/100 Mbps
2. Balanceador de carga II:
    - Sistema operativo: Raspbian con RT Preempt
    - Procesador: ARM11
    - Núcleos: 1
    - Velocidad procesador: 700MHz
    - Modelo: Raspberry Pi B
    - Memoria RAM: 512MB
    - Controlador de acceso al medio: Ethernet 10/100 Mbps
3. Balanceador de carga III:
    - Sistema operativo: Ubuntu 14.04
    - Procesador: Core i5-4570

- Núcleos: 4
- Velocidad procesador: 3.2GHz
- Modelo: Lenovo ThinkCentre M93p
- Memoria RAM: 4GB DDR3 SDRAM
- Tarjeta de red integrada: Ethernet 10/100 Mbps

4. Servidores:

   - Sistema operativo: mbed OS
   - Procesador: ARM Cortex M3
   - Núcleos: 1
   - Velocidad procesador: 96MHz
   - Modelo: LPC1768
   - Memoria RAM: 64kB (32kB para Ethernet)
   - Controlador de acceso al medio: Ethernet 10/100 Mbps

5. Clientes:

   - Sistema operativo: Ubuntu 14.04
   - Procesador: Core i5-4570
   - Núcleos: 4
   - Velocidad procesador: 3.2GHz
   - Modelo: Lenovo ThinkCentre M93p
   - Memoria RAM: 4GB DDR3 SDRAM
   - Tarjeta de red integrada: Ethernet 10/100 Mbps

6. Conmutadores de red:

   - Modelo: Cisco Catalyst 2950
   - Gestionable: Sí
   - Velocidad: 10/100 Mbps (10/100BaseT)
   - Puertos: 24 puertos RJ45
   - Auto-Negociación: Velocidad, dúplex y control de flujo

Destacar que el balanceador III es un equipo con mayores capacidades que el utilizado en la fase I, y además es el mismo modelo que los equipos cliente. Por último, este balanceador también se utilizará para las labores de configuración y gestión del experimento que se explicarán más adelante.

4.1.5.2 Configuración de red

El experimento se lleva a cabo en el laboratorio 112 de la Facultad de Ingeniería de la Universidad de Deusto. Este laboratorio dispone de los equipos cliente antes definidos y una red desplegada que permite la comunicación entre los equipos (intranet) y hacia fuera (Internet). La idea inicial era realizar el experimento sobre la misma red del laboratorio. Por eso se realizaron unas pruebas iniciales con Iperf para medir el rendimiento de red entre dos equipos del laboratorio. Los resultados no fueron buenos, con un rendimiento medio menor a 10Mbps y gran varianza. Conectados los dos equipos a través de un conmutador no gestionable (definido anteriormente en 4.1.2.1) se conseguían rendimientos medios de 94Mbps. Quedaba claro que la red del laboratorio daba un rendimiento muy bajo y poco estable, debido posiblemente a que el conmutador de red que conectaba los equipos del laboratorio 112 también daba servicio a otros laboratorios con alumnos generando tráfico.

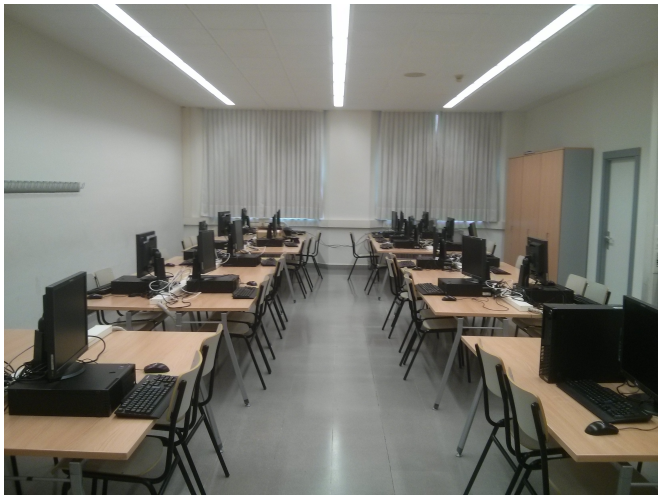


Figura 4.1 Laboratorio 112

Ya que no se podía confiar en la coherencia de los resultados, se decidió crear un entorno controlado y aislado para realizar los experimentos. El conmutador de red disponible tenía 24 puertos, y el experimento requería hasta 27 equipos (20 clientes, 1 balanceador, 5 servidores, y 1 equipo para gestionar el experimento). Por lo tanto hay que utilizar dos conmutadores para crear la red local. Debido a la disposición del laboratorio, se eligieron dos filas de diez ordenadores cuyas conexiones de red acababan en la pared. En ese lugar se colocaron los conmutadores y se conectaron todos los equipos cliente. Un cable cruzado conectaba los dos conmutadores y se configuraron los conmutadores para que todos los puertos estuvieran en la misma red. En ese momento había diez equipos cliente conectados en cada conmutador. El equipo de gestión restante y los dispositivos del clúster se conectaron todos a uno de los conmutadores.

Antes de empezar con el experimento se hizo una prueba de una conexión de un cliente a un servidor C2, sin balanceador, y se obtuvo un rendimiento de red superior a 20Mbps. Si se iba a tener hasta 5 servidores funcionando a la vez, cabía la posibilidad de que se necesitaran más de 100Mbps, la capacidad

Figura 4.2 Despliegue del clúster (conmutadores)

máxima del puerto que conectaba los dos conmutadores. Por eso se decidió sobredimensionar y utilizar un canal agregado de 2 puertos para conectar los conmutadores y evitar así un posible cuello de botella en la conexión entre conmutadores.

El concepto básico de un canal agregado (o EtherChannel en conmutadores Cisco) es que varios puertos físicos se combinan en un enlace lógico. Esto proporciona beneficios que incluyen el aumento de la capacidad, en este caso habría una capacidad teórica máxima de 200Mbps, y además se añade redundancia, es decir, el canal agregado puede sobrevivir a la pérdida de un puerto.

Para configurar un canal agregado de capa 2 hay que configurar las interfaces Ethernet del conmutador donde estarán conectados los puertos del enlace. Para ello se usa el comando de configuración de interfaz *channel-group*, que crea la interfaz lógica para esos puertos. A continuación se muestra cómo se configura un canal agregado para los puertos 1 y 2:

Listado 4.5 EtherChannel

```
Cat2950# configure terminal
Cat2950(config)# interface fa0/1
Cat2950(config-if)# channel-group 1 mode on
Cat2950(config-if)# exit
Cat2950(config)# interface fa0/2
Cat2950(config-if)# channel-group 1 mode on
Cat2950(config-if)# exit
```

Una vez configurado el canal agregado, una consideración a tener en cuenta es el tipo de balanceo de carga aplicado. Al tener más de un puerto configurado en el canal agregado, el conmutador debe decidir por qué puerto envía el paquete. EtherChannel proporciona balanceo de carga por paquete, no por conexión, y no se asigna dinámicamente, sino que se envía el paquete utilizando un algoritmo determinista. El conmutador decide el puerto por el resultado de una función que toma como entrada uno o más campos de cada paquete. Los campos que se evalúan dependen del tipo de conmutador. Los

Catalyst 2950 utilizados tienen dos métodos para distribuir la carga en el canal agregado, y son por dirección MAC de origen o de destino. Como en este caso el canal agregado está formado por dos puertos, se utiliza el último bit de la dirección MAC para seleccionar el puerto por el que se envía el paquete.

Con el objetivo de que el balanceo fuese lo más equitativo posible, se analizan las direcciones MAC de los equipos conectados a los dos conmutadores para determinar qué balanceo aplicar. En primer lugar se analizó el conmutador que solo tenía equipos cliente. Hay que tener en cuenta que el tráfico de salida de este conmutador tendrá como destino el clúster, ubicado en el otro conmutador de red. El balanceador de carga del clúster tiene obviamente una única dirección MAC, por lo que si se configurase el balanceo del conmutador por dirección MAC de destino, todos los paquetes del conmutador se enviarían por el mismo puerto del canal agregado. Una vez descartada esta opción, solo queda evaluar el balanceo por dirección MAC de origen. En este caso las direcciones MAC pares e impares (último bit a 0 y 1 respectivamente) se reparten equitativamente, es decir, hay 5 de cada tipo. Si todos los equipos hubiesen sido pares o impares, todos los paquetes utilizarían el mismo puerto de salida. En ese caso hubiera sido necesario modificar las direcciones MAC de origen para que los dos puertos se utilizasen de manera equitativa durante el experimento.

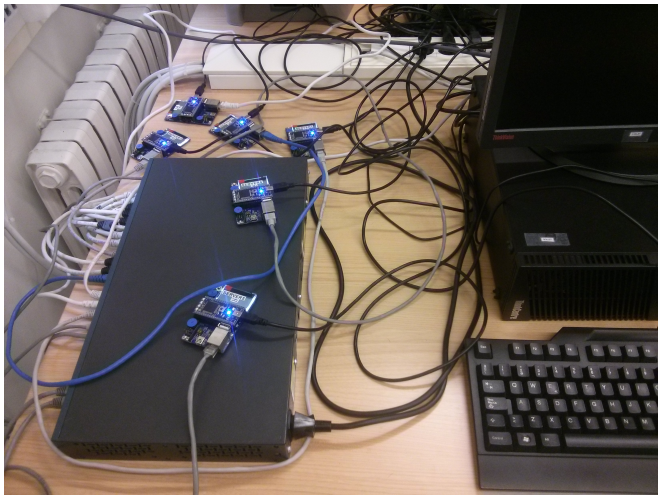


Figura 4.3 Despliegue del clúster (dispositivos C2)

Respecto al conmutador que contiene el clúster, sí se puede configurar el método de balanceo por dirección MAC de destino, ya que permite que el canal agregado se utilice de manera equitativa con los equipos cliente del otro conmutador.

Por último, se comprueba la conectividad de red de todos los equipos utilizando la herramienta *ping* para enviar mensajes ICMP entre los dos conmutadores.

4.1.5.3 Configuración software

El software utilizado es por lo general el mismo que en la fase I, explicado en la sección 4.1.2.3. La diferencia radica en que las conexiones se generan desde diferentes equipos cliente, y para que una conexión sea concurrente debe generarse al mismo tiempo que otras. Esto significa que es necesario utilizar software que permita sincronizar en red los equipos cliente.

El protocolo NTP (Network Time Protocol) es un protocolo diseñado para sincronizar con precisión el reloj de un equipo utilizando servidores de tiempo. Normalmente, los equipos Linux sincronizan el reloj periódicamente con un servidor de tiempo externo, ya que los relojes de los equipos son imperfectos y se desplazan durante el día. Por otra parte, la deriva de los relojes también es diferente entre equipos, de tal manera que al final del día pueden existir diferencias significativas entre los diferentes equipos de una red local, lo que puede interferir con ciertas operaciones que requieran sincronización.

Este enfoque de sincronizar los relojes con un servidor externo puede replicarse para una red local. El objetivo es que todos los clientes estén sincronizados con la misma hora, no que esa hora sea la correcta. Lo primero es instalar el servicio NTP en un equipo de la red local, que actuará como servidor "externo" para el resto de equipos. El equipo seleccionado será el balanceador de carga III, que actuará de balanceador cuando el experimento lo requiera pero también se encargará de operar como servidor NTP para sincronizar los equipos antes de ejecutar el experimento. Una vez configurado el servicio NTP (*/etc/ntp.conf*) y ejecutado como servicio, el resto de equipos de la red local pueden sincronizarse con él. El comando *ntpdate* puede utilizarse para sincronizar la fecha y hora local mediante una solicitud al servidor NTP (*ntpdate -u* seguido de la dirección IP del servidor NTP). De esta forma, las diferencias en los relojes entre los equipos de la red local se reducen al mínimo, lo que permite ejecutar conexiones concurrentes entre diferentes equipos. A continuación, se debe programar la hora exacta para generar las conexiones al clúster desde los equipos cliente.

En los sistemas operativos tipo Unix, el comando *at* se usa para programar comandos o scripts que se ejecutarán una única vez y en un momento determinado. A diferencia del comando *cron*, su resultado no es persistente, es decir, si se reinicia o apaga el equipo se perderá la tarea programada. En primer lugar, se ejecuta el comando *at* seguido de la hora programada (por ejemplo, at 11:45). A continuación, el terminal permite escribir el comando, script, o serie de comandos y/o scripts para ejecutar en ese tiempo programado. El comando *atq* permite visualizar las tareas en cola esperando ejecución.

Por último, para generar y medir el rendimiento de una conexión al clúster se vuelve a utilizar Iperf como en la fase I, y por tanto, cada equipo cliente debe tener esta herramienta instalada.

4.1.6 Definición del experimento: fase II

La metodología para medir el rendimiento de red del clúster de dispositivos C2 es la misma que en la fase I. La diferencia es que en vez de ejecutar conexiones concurrentes desde el mismo equipo cliente, se utilizan múltiples equipos que ejecutarán una única conexión, pero lo harán al mismo tiempo. Este factor impide hacer las ejecuciones del experimento de manera manual desde los equipos cliente. Por ello, se

utiliza uno de los equipos conectados a la red local, en este caso el balanceador III, para configurar y ejecutar el experimento de manera remota.

En esta tarea juega un papel fundamental el protocolo SSH (Secure Shell). Este protocolo de comunicación ofrece un acceso a una línea de comandos del equipo remoto sobre una conexión cifrada. La sintaxis del comando SSH es simple, ya que solo hay que especificar el nombre de usuario y la dirección IP del equipo al que se desea conectarse (por ejemplo, *ssh user@10.48.1.1*), y si la contraseña introducida es correcta, se puede acceder a la consola del equipo remoto con los privilegios que tenga la cuenta utilizada.

A primera vista, se identifican dos limitaciones para el experimento. En primer lugar, se necesita abrir una conexión SSH por cada equipo remoto al cual se accede, y además se utiliza esa conexión para realizar tareas que se irán repitiendo por cada equipo. En cuanto a la segunda limitación, una conexión SSH requiere introducir la contraseña del equipo remoto. *Expect* es una herramienta de automatización para convertir este proceso interactivo en una tarea completamente no interactiva. Básicamente, *Expect* emula la comunicación bidireccional con un usuario.

Para automatizar el experimento se han creado sencillos scripts que permiten acceder rápidamente y de manera secuencial a los equipos del experimento mediante SSH. En estos scripts se utiliza *Expect* para lanzar la conexión SSH (comando *spawn*) y se espera a una respuesta concreta del equipo remoto (comando *expect*). Cuando se recibe esa respuesta, el equipo local puede enviar una respuesta u otro comando al equipo remoto (comando *send*). Además, *Expect* permite incluir argumentos de entrada y luego usar esos argumentos como variables en el script.

A continuación, se muestran los scripts desarrollados en su orden de ejecución:

- Instalación y configuración: la funcionalidad del script es preparar los equipos cliente para el experimento. El script recibe como argumento por la línea de comandos el último byte o número de la dirección IP del equipo cliente. Una vez completa la dirección IP de destino, automatiza la conexión SSH de manera que no sea necesario indicar la contraseña. Cuando la sesión interactiva se establece, se escriben los comandos en el equipo remoto. En primer lugar, se inicia una instalación de los paquetes software requeridos para el experimento (Iperf, at y ntpdate), y una vez instalados, se sincroniza el reloj del sistema con el servidor NTP (el mismo equipo desde el que se está lanzando el script) para que todos los equipos de la red local compartan la misma hora (ntpdate -u).

- Conexión Iperf: este script define el comando Iperf para crear una conexión con el clúster. Contiene la dirección IP del clúster, el puerto del servicio, y la ruta donde guardar el resultado en un archivo de texto.

- Copia del archivo Iperf: este script se utiliza para copiar el script anterior en los equipos cliente. Para copiar archivos o directorios entre un equipo local y un equipo remoto a través de SSH se utiliza el comando *scp*. Se usarán los mismos datos de conexión que se vienen usando para conectarse por SSH, pero utilizando la siguiente sintaxis: *scp ruta_origen/archivo_origen usua-*

rio@IP_remota: ruta_destino. Este script también recibe como argumento el último número de la dirección IP del equipo cliente para establecer la conexión SSH.

- Programación de la ejecución: la funcionalidad del script es programar la hora de ejecución del script Iperf para que los equipos seleccionados generen conexiones concurrentes al clúster. El script recibe dos argumentos: el último número de la dirección IP y la hora deseada. Utilizando la opción -f se consigue que el comando *at* programe la ejecución del script Iperf a la hora correspondiente al valor especificado como argumento.

- Copia de resultados Iperf: este script se utiliza para copiar los resultados de los equipos remotos al equipo local. En este caso la sintaxis a utilizar es la siguiente: *scp usuario@IP_remota: ruta_origen/archivo_origen ruta_destino*. Al igual que en los casos anteriores, se utiliza *Expect* para automatizar la conexión SSH.

Como se puede ver, todos los scripts reciben como argumento el último byte o número de la dirección IP del equipo cliente. Para seleccionar los equipos cliente que el experimento demande en cada momento, se utiliza un bucle *for* sobre un conjunto de números que representará en cada momento el último byte de la dirección IP de los equipos cliente correspondientes. Esos equipos cliente serán los que ejecuten las instrucciones definidas por el script.

Por último, una vez recuperados los datos de los equipos remotos, se generan los ficheros CSV y se realiza el análisis estadístico correspondiente.

4.1.7 Resultados y análisis: fase II

Al igual que en la fase I, existen 3 niveles para el factor *LB*, 4 niveles para el factor *núm_servidores* y 12 niveles para el factor *núm_clientes*. Los experimentos en los que se consideran en conjunto los efectos de más de un factor y sus posibles combinaciones se denominan "experimentos factoriales" y pueden ser analizados mediante un ANOVA factorial. Al igual que en la sección 4.1.4, se asume que los supuestos del ANOVA se cumplen y se utilizan las especificaciones del Modelo Lineal General Univariante.

A continuación se muestran los resultados obtenidos:

En primer lugar, la tabla 4.3 muestra los factores del experimento, sus valores, y el tamaño de muestras que hay en cada grupo. Como se puede observar, el número de muestras es igual al de la fase I (tabla 4.1).

La tabla 4.4 recoge los resultados de todas las combinaciones de nivel de los factores inter-sujetos respecto a la variable dependiente (rendimiento de red). Las dos primeras filas contienen los valores para el modelo de regresión. Además, la fila *Modelo corregido* tiene en cuenta todos los efectos del modelo, es decir, los siete efectos de interés y el de intersección. El nivel de significación (0,000) determina que el modelo explica una parte significativa de la variación observada en la variable dependiente. A continuación, se analiza el coeficiente de determinación del modelo o R^2. Este coeficiente indica la cantidad de variabilidad explicada por el modelo. Cuanto mayor sea, más predecible es la variable dependiente en función de las variables independientes o factores. En este modelo, el R^2 corregido es

Tabla 4.3 Factores inter-sujetos

Factor	Valor	N
LB	Balanceador de carga I	4200
	Balanceador de carga II	4200
	Balanceador de carga III	4200
núm_servidores	2	1800
	3	2700
	4	3600
	5	4500
núm_clientes	2	180
	3	270
	4	720
	5	450
	6	1080
	8	1440
	9	810
	10	900
	12	2160
	15	1350
	16	1440
	20	1800

Tabla 4.4 Efectos inter-sujetos

Fuente	Suma de cuadrados	gl	Media cuadrática	F	Sig.	η_p^2
Modelo corregido	$117859,802^a$	47	2507,655	21179,827	0,000	0,988
Intersección	860040,913	1	860040,913	7263963,974	0,000	0,998
LB	18354,949	2	9177,475	77513,573	0,000	0,925
núm_servidores	405,947	2	202,973	1714,327	0,000	0,215
núm_clientes	63089,810	10	6308,981	53286,082	0,000	0,977
LB * núm_servidores	1117,892	4	279,473	2360,449	0,000	0,429
LB * núm_clientes	9958,807	20	497,940	4205,638	0,000	0,870
núm_servidores * núm_clientes	411,651	2	205,826	1738,418	0,000	0,217
LB * núm_servidores * núm_clientes	231,706	4	57,927	489,252	0,000	0,135
Error	1486,135	12552	0,118			
Total	1121045,521	12600				
Total corregido	119345,937	12599				

R^2 = 0,988 (R^2 corregido = 0,988)

0,988, lo que significa que el 98,8 % de la variabilidad del rendimiento de red se puede explicar con los tres factores incluidos en el modelo (*LB*, *núm_servidores* y *núm_clientes*). Como en la fase I, se verifica que se han seleccionado los factores más significativos para el ANOVA.

Las tres filas siguientes recogen los efectos principales, es decir, los efectos individuales de los tres factores incluidos en el modelo: *LB*, *núm_servidores* y *núm_clientes*. Al igual que en la fase I, los niveles de significación (Sig.) indican que los grupos definidos por los tres factores tienen rendimientos medios de red significativamente diferentes (Sig. = 0,000 < 0,05), y que el parámetro más significativo sigue siendo el número de clientes ($\eta_p{}^2 = 0{,}977$), seguido del tipo de balanceador utilizado ($\eta_p{}^2 = 0{,}925$).

Las siguientes filas contienen información sobre el efecto de las interacciones entre los factores. Todos los estadísticos F correspondientes llevan asociados un nivel de significación de 0,000, lo que significa que las interacciones tienen un efecto significativo sobre el rendimiento de red. A diferencia de la fase I, la interacción núm_servidores-núm_clientes es también significativa respecto al rendimiento de red (Sig. = 0,000 < 0,05), con un tamaño del efecto grande ($\eta_p^2 = 0{,}217 > 0{,}14$).

La tabla 4.5 muestra la media, la desviación estándar y el tamaño de muestras para cada combinación de los factores utilizados. A primera vista, se observa que existen diferencias entre las medias, y los estadísticos F correspondientes a los factores y a sus interacciones (tabla 4.4) permiten rechazar la hipótesis general de que los promedios comparados son iguales.

Para identificar dónde se encuentran esas diferencias se utiliza un tipo particular de contrastes denominados comparaciones múltiples (*post hoc*).

Existen diferentes procedimientos dependiendo de si se asumen las varianzas como iguales o no. Para un ANOVA factorial se seleccionan los métodos *post hoc* que asumen varianzas poblacionales iguales, y entre ellos se ha optado por uno de los procedimientos más comunes: Bonferroni. El método de Bonferroni está basado en la distribución T de Student y en la desigualdad de Bonferroni. Este método controla la tasa de error dividiendo el nivel de significación entre el número de comparaciones llevadas a cabo.

A continuación se muestran los resultados obtenidos para los efectos individuales de los tres factores:

La tabla 4.6 muestra el resultado obtenido para el factor *LB*. Las dos primeras columnas muestran los grupos, balanceadores en este caso, que se están comparando. Las filas de la tabla muestran todas las posibles combinaciones, las diferencias entre los rendimientos de red medios de cada combinación, el error típico de esas diferencias y el nivel de significación asociado a cada diferencia.

En las últimas columnas se observan los límites del intervalo de confianza, cuya función es mostrar entre qué limites se encuentra la verdadera diferencia entre las medias de los grupos. Estos intervalos también permiten tomar decisiones sobre si dos medias difieren o no significativamente (si el intervalo contiene el cero, no difieren significativamente). En este caso ninguno de los intervalos contiene el cero, por lo que en principio se podría asumir que las medias de todos los grupos difieren significativamente. No obstante, se debe tener en cuenta que el intervalo se obtiene individualmente para cada diferencia, sin establecer control sobre la tasa de error, y por tanto es más aconsejable utilizar el valor de significación (Sig.) para decidir sobre la hipótesis de igualdad de medias.

Tabla 4.5 Estadísticos descriptivos del procedimiento ANOVA

LB	núm_servidores	núm_clientes	Media	Desviación estándar	N
Balanceador de carga I	2	2	17,412	0,5918	60
		4	17,008	0,4578	120
		6	12,846	0,7605	180
		8	10,095	1,0411	240
	3	3	17,557	0,5176	90
		6	15,895	0,2260	180
		9	11,255	0,3923	270
		12	8,643	0,2649	360
	4	4	17,212	0,3767	120
		8	12,498	0,2453	240
		12	8,800	0,2417	360
		16	6,626	0,2985	480
	5	5	17,048	0,4120	150
		10	10,555	0,2097	300
		15	7,028	0,2852	450
		20	5,454	0,2709	600
Balanceador de carga II	2	2	11,790	0,2312	60
		4	11,249	0,0907	120
		6	9,987	0,1788	180
		8	8,257	0,3557	240
	3	3	10,898	0,1754	90
		6	8,690	0,1086	180
		9	7,607	0,1771	270
		12	7,004	0,2100	360
	4	4	10,161	0,1508	120
		8	8,253	0,3128	240
		12	7,150	0,1623	360
		16	6,181	0,1505	480
	5	5	9,307	0,0879	150
		10	7,480	0,1968	300
		15	6,591	0,1814	450
		20	5,218	0,1458	600
Balanceador de carga III	2	2	14,123	0,3711	60
		4	13,373	0,2933	120
		6	12,464	0,2124	180
		8	9,376	1,2833	240
	3	3	13,982	0,4403	90
		6	12,967	0,2726	180
		9	10,848	0,1544	270
		12	8,501	0,1778	360
	4	4	13,674	0,2980	120
		8	11,931	0,1480	240
		12	8,593	0,1793	360
		16	6,633	0,1943	480
	5	5	13,415	0,2529	150
		10	10,400	0,1883	300
		15	7,070	0,2725	450
		20	5,505	0,2631	600

Tabla 4.6 Comparaciones múltiples *post hoc*

(I) LB	(J) LB	Diferencia medias (I-J)	Error típico	Sig.	Intervalo de confianza al 95 % Límite inferior	Límite superior
Balanceador carga I	Balanceador carga II	2,512*	0,0075	0,000	2,494	2,530
	Balanceador carga III	0,728*	0,0075	0,000	0,710	0,746
Balanceador carga II	Balanceador carga I	-2,512*	0,0075	0,000	-2,530	-2,494
	Balanceador carga III	-1,783*	0,0075	0,000	-1,801	-1,765
Balanceador carga III	Balanceador carga I	-0,728*	0,0075	0,000	-0,746	-0,710
	Balanceador carga II	1,783*	0,0075	0,000	1,765	1,801

La hipótesis nula que se pone a prueba con cada combinación es que las medias comparadas son iguales. Cuando las medias de los balanceadores comparados difieren al nivel establecido (0,05) se acepta que existe una diferencia estadística significativa con un nivel de confianza del 95 % entre las medias de esos dos balanceadores, y esa combinación aparece marcada en la tabla con un asterisco.

De acuerdo con el procedimiento de Bonferroni, se concluye que todos los balanceadores difieren significativamente en sus medias (Sig. = 0,000 en las tres comparaciones). Además, la diferencia entre las medias permite determinar que el balanceador de carga I tiene un rendimiento medio mayor que el balanceador de carga III (diferencia entre medias positiva), y a su vez éste último tiene un rendimiento medio mayor que el balanceador de carga II. En definitiva, para el experimento planteado se demuestra que el dispositivo C2 tiene un rendimiento medio de red mayor que los otros dos balanceadores de características muy superiores.

Tabla 4.7 Subgrupos homogéneos

			Subgrupo para α=0,05		
	LB	N	1	2	3
Scheffé	Balanceador carga II	4200	7,485		
	Balanceador carga III	4200		9,268	
	Balanceador carga I	4200			9,996
	Sig.		1,000	1,000	1,000

La tabla 4.7 muestra una clasificación basada en el procedimiento de Scheffé del resultado obtenido con las comparaciones múltiples. Conviene destacar que esta clasificación no está disponible para todos los procedimientos *post hoc*. Los grupos cuyas medias no difieren significativamente entre sí están agrupados en el mismo subconjunto, y los grupos cuyas medias difieren forman parte de subconjuntos diferentes. Los grupos se enumeran en orden ascendente de medias. En este caso, se observa que los grupos no son homogéneos, es decir, en cada subgrupo está incluido un solo balanceador, lo que significa que la media del rendimiento de red para cada balanceador difiere significativamente de la media de los otros balanceadores, y que obviamente no difieren de sí mismos (Sig. = 1,000).

La tabla 4.8 muestra el resultado obtenido para el factor *núm_servidores*. De acuerdo con el procedimiento de Bonferroni, se concluye que el rendimiento de red promedio respecto al número de servidores que forman el clúster difiere de forma significativa (Sig. = 0,000). Además, la diferencia entre

Tabla 4.8 Comparaciones múltiples *post hoc*

(I) Número servidores	(J) Número servidores	Diferencia medias (I-J)	Error típico	Sig.	Intervalo de confianza al 95 % Límite inferior	Límite superior
2	3	1,338*	0,0105	0,000	1,310	1,365
	4	2,853*	0,0099	0,000	2,827	2,879
	5	3,999*	0,0096	0,000	3,974	4,024
3	2	-1,338*	0,0105	0,000	-1,365	-1,310
	4	1,515*	0,0088	0,000	1,492	1,539
	5	2,661*	0,0084	0,000	2,639	2,684
4	2	-2,853*	0,0099	0,000	-2,879	-2,827
	3	-1,515*	0,0088	0,000	-1,539	-1,492
	5	1,146*	0,0077	0,000	1,126	1,166
5	2	-3,999*	0,0096	0,000	-4,024	-3,974
	3	-2,661*	0,0084	0,000	-2,684	-2,639
	4	-1,146*	0,0077	0,000	-1,166	-1,126

las medias permite determinar que a medida que se incrementa el número de servidores que forman el clúster, el rendimiento medio se reduce.

A medida que se distribuye el servicio entre más servidores, el rendimiento por servidor tiende a disminuir. Sin embargo, como hay más servidores, la mejora se aprecia en conjunto. Si se presta atención a la tabla 4.5, se observa una mejora sublineal del rendimiento a medida que se incrementa el número de servidores en uso. Cuando se hace un balanceo de carga, el crecimiento del clúster no tiene una relación lineal con el rendimiento de red [Dehmer et al., 2016], es decir, cinco servidores no ofrecen cinco veces el rendimiento de un único servidor. No obstante, los resultados del experimento demuestran que añadir servidores adicionales se traduce en un mayor rendimiento global, y por tanto se puede determinar que el clúster de dispositivos C2 es escalable.

Tabla 4.9 Subgrupos homogéneos

	Número servidores	N	Subgrupo para α=0,05 1	2	3	4
Scheffé	5	4500	7,447			
	4	3600		8,593		
	3	2700			10,109	
	2	1800				11,446
	Sig.		1,000	1,000	1,000	1,000

La tabla 4.9 muestra la clasificación de los grupos basada en el grado de parecido existente entre sus medias. Como los tamaños de los grupos son desiguales, se utiliza la media armónica de los tamaños de los grupos, en este caso 2805,195. Al igual que con el factor *LB*, los grupos no son homogéneos, y por tanto en cada columna solo está incluido el grupo que representa el número concreto de servidores que forman el clúster. En este caso, el rendimiento por servidor se reduce en un factor aproximado de 1,15 al agregar un nuevo servidor al clúster.

Tabla 4.10 Comparaciones múltiples *post hoc*

(I) Núm. clientes	(J) Núm. clientes	Dif. medias (I-J)	Err.	Sig.	Confianza al 95 % Límite inf.	Confianza al 95 % Límite sup.
	3	0,296*	0,0331	0,000	0,185	0,408
	4	0,662*	0,0287	0,000	0,566	0,759
	5	1,185*	0,0303	0,000	1,083	1,287
	6	2,300*	0,0277	0,000	2,207	2,394
	8	4,373*	0,0272	0,000	4,282	4,465
2	9	4,538*	0,0284	0,000	4,443	4,634
	10	4,963*	0,0281	0,000	4,869	5,058
	12	6,327*	0,0267	0,000	6,237	6,416
	15	7,545*	0,0273	0,000	7,453	7,637
	16	7,962*	0,0272	0,000	7,870	8,053
	20	9,049*	0,0269	0,000	8,958	9,140
	2	-0,296*	0,0331	0,000	-0,408	-0,185
	4	0,366*	0,0246	0,000	0,284	0,449
	5	0,889*	0,0265	0,000	0,800	0,978
	6	2,004*	0,0234	0,000	1,925	2,083
	8	4,077*	0,0228	0,000	4,000	4,154
3	9	4,242*	0,0242	0,000	4,161	4,324
	10	4,667*	0,0239	0,000	4,587	4,748
	12	6,030*	0,0222	0,000	5,956	6,105
	15	7,249*	0,0229	0,000	7,172	7,326
	16	7,666*	0,0228	0,000	7,589	7,742
	20	8,753*	0,0225	0,000	8,677	8,829
	2	-0,662*	0,0287	0,000	-0,759	-0,566
	3	-0,366*	0,0246	0,000	-0,449	-0,284
	5	0,523*	0,0207	0,000	0,453	0,592
	6	1,638*	0,0166	0,000	1,582	1,694
	8	3,711*	0,0157	0,000	3,658	3,764
4	9	3,876*	0,0176	0,000	3,817	3,935
	10	4,301*	0,0172	0,000	4,243	4,359
	12	5,664*	0,0148	0,000	5,614	5,714
	15	6,883*	0,0159	0,000	6,829	6,936
	16	7,299*	0,0157	0,000	7,246	7,352
	20	8,387*	0,0152	0,000	8,336	8,438
	2	-1,185*	0,0303	0,000	-1,287	-1,083
	3	-0,889*	0,0265	0,000	-0,978	-0,800
	4	-0,523*	0,0207	0,000	-0,592	-0,453
	6	1,115*	0,0193	0,000	1,050	1,180
	8	3,188*	0,0186	0,000	3,126	3,251
5	9	3,353*	0,0202	0,000	3,285	3,421
	10	3,778*	0,0199	0,000	3,711	3,845
	12	5,141*	0,0178	0,000	5,081	5,201
	15	6,360*	0,0187	0,000	6,297	6,423
	16	6,777*	0,0186	0,000	6,714	6,839
	20	7,864*	0,0181	0,000	7,803	7,925
	2	-2,300*	0,0277	0,000	-2,394	-2,207
	3	-2,004*	0,0234	0,000	-2,083	-1,925
	4	-1,638*	0,0166	0,000	-1,694	-1,582
	5	-1,115*	0,0193	0,000	-1,180	-1,050
	8	2,073*	0,0139	0,000	2,026	2,120
6	9	2,238*	0,0160	0,000	2,184	2,292
	10	2,663*	0,0155	0,000	2,611	2,715
	12	4,026*	0,0128	0,000	3,983	4,069
	15	5,245*	0,0140	0,000	5,198	5,292
	16	5,661*	0,0139	0,000	5,615	5,708
	20	6,749*	0,0132	0,000	6,704	6,793
	2	-4,373*	0,0272	0,000	-4,465	-4,282
	3	-4,077*	0,0228	0,000	-4,154	-4,000
	4	-3,711*	0,0157	0,000	-3,764	-3,658
8						

Tabla 4.10 – continuación

(I) Núm. clientes	(J) Núm. clientes	Dif. medias (I-J)	Err.	Sig.	Confianza al 95 % Límite inf.	Límite sup.
	5	-3,188*	0,0186	0,000	-3,251	-3,126
	6	-2,073*	0,0139	0,000	-2,120	-2,026
	9	0,165*	0,0151	0,000	0,114	0,216
	10	0,590*	0,0146	0,000	0,541	0,639
	12	1,953*	0,0117	0,000	1,914	1,993
	15	3,172*	0,0130	0,000	3,128	3,216
	16	3,588*	0,0128	0,000	3,545	3,632
	20	4,676*	0,0122	0,000	4,635	4,717
	2	-4,538*	0,0284	0,000	-4,634	-4,443
	3	-4,242*	0,0242	0,000	-4,324	-4,161
	4	-3,876*	0,0176	0,000	-3,935	-3,817
	5	-3,353*	0,0202	0,000	-3,421	-3,285
	6	-2,238*	0,0160	0,000	-2,292	-2,184
9	8	-0,165*	0,0151	0,000	-0,216	-0,114
	10	0,425*	0,0167	0,000	0,369	0,481
	12	1,788*	0,0142	0,000	1,740	1,836
	15	3,007*	0,0153	0,000	2,955	3,058
	16	3,423*	0,0151	0,000	3,372	3,474
	20	4,511*	0,0146	0,000	4,462	4,560
	2	-4,963*	0,0281	0,000	-5,058	-4,869
	3	-4,667*	0,0239	0,000	-4,748	-4,587
	4	-4,301*	0,0172	0,000	-4,359	-4,243
	5	-3,778*	0,0199	0,000	-3,845	-3,711
	6	-2,663*	0,0155	0,000	-2,715	-2,611
10	8	-0,590*	0,0146	0,000	-0,639	-0,541
	9	-0,425*	0,0167	0,000	-0,481	-0,369
	12	1,363*	0,0137	0,000	1,317	1,409
	15	2,582*	0,0148	0,000	2,532	2,632
	16	2,998*	0,0146	0,000	2,949	3,048
	20	4,086*	0,0140	0,000	4,038	4,133
	2	-6,327*	0,0267	0,000	-6,416	-6,237
	3	-6,030*	0,0222	0,000	-6,105	-5,956
	4	-5,664*	0,0148	0,000	-5,714	-5,614
	5	-5,141*	0,0178	0,000	-5,201	-5,081
	6	-4,026*	0,0128	0,000	-4,069	-3,983
12	8	-1,953*	0,0117	0,000	-1,993	-1,914
	9	-1,788*	0,0142	0,000	-1,836	-1,740
	10	-1,363*	0,0137	0,000	-1,409	-1,317
	15	1,219*	0,0119	0,000	1,179	1,259
	16	1,635*	0,0117	0,000	1,596	1,675
	20	2,723*	0,0110	0,000	2,686	2,760
	2	-7,545*	0,0273	0,000	-7,637	-7,453
	3	-7,249*	0,0229	0,000	-7,326	-7,172
	4	-6,883*	0,0159	0,000	-6,936	-6,829
	5	-6,360*	0,0187	0,000	-6,423	-6,297
	6	-5,245*	0,0140	0,000	-5,292	-5,198
15	8	-3,172*	0,0130	0,000	-3,216	-3,128
	9	-3,007*	0,0153	0,000	-3,058	-2,955
	10	-2,582*	0,0148	0,000	-2,632	-2,532
	12	-1,219*	0,0119	0,000	-1,259	-1,179
	16	0,416*	0,0130	0,000	0,373	0,460
	20	1,504*	0,0124	0,000	1,462	1,546
	2	-7,962*	0,0272	0,000	-8,053	-7,870
	3	-7,666*	0,0228	0,000	-7,742	-7,589
	4	-7,299*	0,0157	0,000	-7,352	-7,246
	5	-6,777*	0,0186	0,000	-6,839	-6,714
	6	-5,661*	0,0139	0,000	-5,708	-5,615
16	8	-3,588*	0,0128	0,000	-3,632	-3,545

Tabla 4.10 – continuación

(I) Núm. clientes	(J) Núm. clientes	Dif. medias (I-J)	Err.	Sig.	Confianza al 95 % Límite inf.	Límite sup.
	9	-3,423*	0,0151	0,000	-3,474	-3,372
	10	-2,998*	0,0146	0,000	-3,048	-2,949
	12	-1,635*	0,0117	0,000	-1,675	-1,596
	15	-0,416*	0,0130	0,000	-0,460	-0,373
	20	1,087*	0,0122	0,000	1,046	1,128
20	2	-9,049*	0,0269	0,000	-9,140	-8,958
	3	-8,753*	0,0225	0,000	-8,829	-8,677
	4	-8,387*	0,0152	0,000	-8,438	-8,336
	5	-7,864*	0,0181	0,000	-7,925	-7,803
	6	-6,749*	0,0132	0,000	-6,793	-6,704
	8	-4,676*	0,0122	0,000	-4,717	-4,635
	9	-4,511*	0,0146	0,000	-4,560	-4,462
	10	-4,086*	0,0140	0,000	-4,133	-4,038
	12	-2,723*	0,0110	0,000	-2,760	-2,686
	15	-1,504*	0,0124	0,000	-1,546	-1,462
	16	-1,087*	0,0122	0,000	-1,128	-1,046

La tabla 4.10 muestra el resultado obtenido para el factor *núm_clientes*. De acuerdo con el procedimiento de Bonferroni, se aprecia que hay diferencias significativas (Sig. = 0,000) en las medias respecto al número de conexiones concurrentes establecidas. Además, si se presta atención a la columna que muestra la diferencia entre las medias, se puede concluir que a medida que se incrementa el número de conexiones, el rendimiento medio de red se reduce. Con una capacidad limitada para trabajar, el número y duración de las conexiones que utilizan ese ancho de banda tienen un impacto en el rendimiento de red. A mayor número de clientes concurrentes, se intentará repartir lo mejor posible el ancho de banda disponible entre esas conexiones. Esto se traduce en una reducción del rendimiento de red por conexión.

Tabla 4.11 Subgrupos homogéneos

			Subgrupo para α=0,05											
	Número clientes	N	1	2	3	4	5	6	7	8	9	10	11	12
	20	1800	5,393											
	16	1440		6,480										
	15	1350			6,896									
	12	2160				8,115								
	10	900					9,478							
	9	810						9,903						
Scheffé	8	1440							10,068					
	6	1080								12,141				
	5	450									13,257			
	4	720										13,779		
	3	270											14,146	
	2	180												14,442
	Sig.		1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000

Por último, la tabla 4.11 muestra por columnas los subgrupos de medias iguales. Como los tamaños de los grupos son desiguales, se utiliza la media armónica, en este caso 622,080. De la prueba de Scheffé se observa que todos los grupos han sido clasificados en columnas diferentes, y por tanto difieren significativamente entre sí (grupos no homogéneos).

Las comparaciones múltiples *post hoc* proporcionan la información necesaria para poder interpretar los efectos principales o individuales. Con esta información ya se han podido extraer ciertas conclusiones, como que el balanceador C2 funciona mejor para el clúster desplegado que los otros dos balanceadores más potentes. No obstante, estas comparaciones múltiples no permiten analizar el efecto de las interacciones de los factores. La interpretación correcta de una interacción requiere la ayuda de un gráfico. En lugar de mostrar el resultado de las posibles interacciones entre dos factores, solo se analizará la interacción triple del experimento. Para representar esa interacción triple se utiliza un gráfico de líneas para cada interacción doble en cada nivel del factor *núm_servidores* (grupo de figuras 4.4).

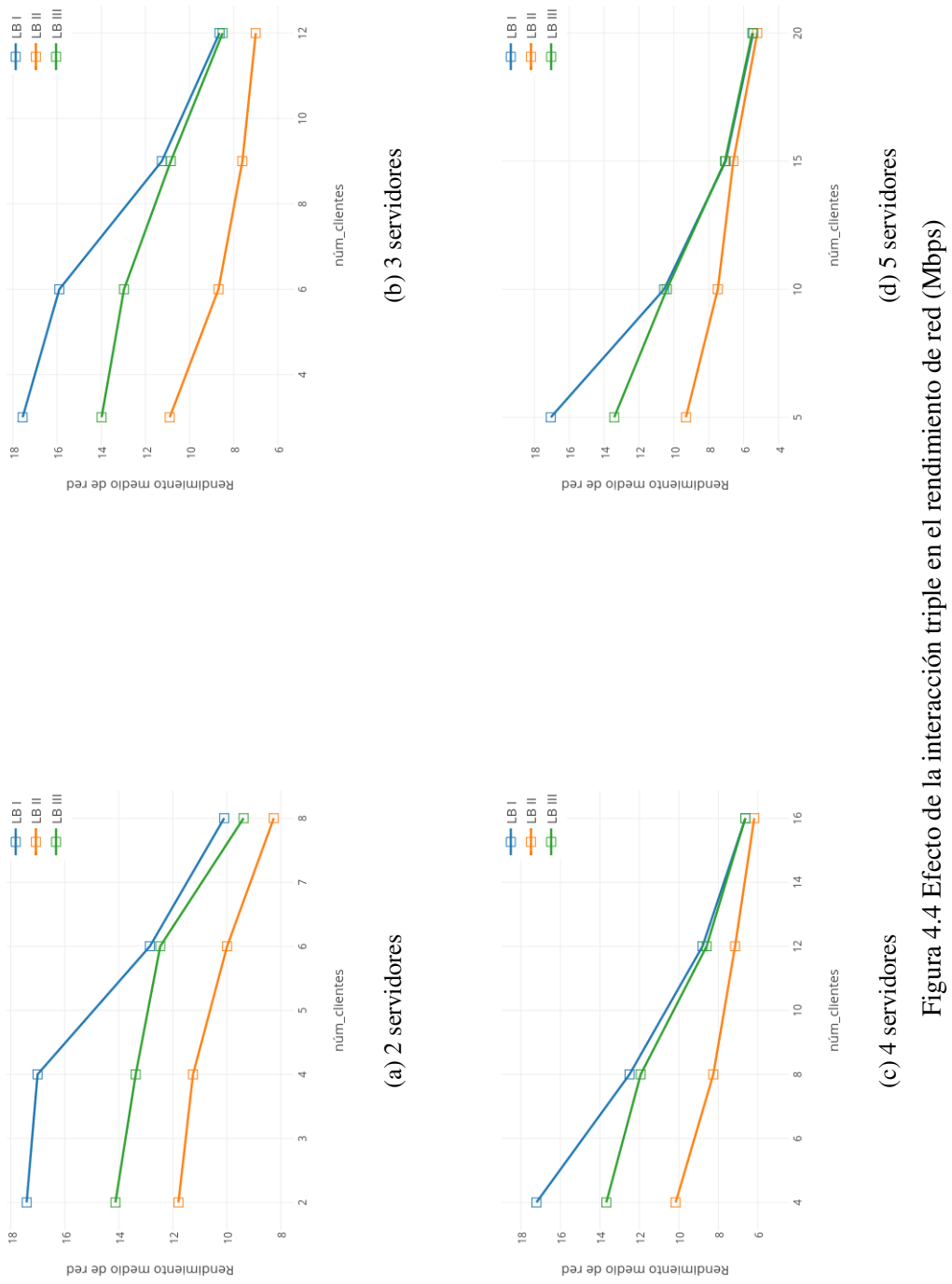
El efecto de la interacción triple de los factores permite determinar que el rendimiento medio del clúster es mayor con el dispositivo C2 (denominado en las gráficas como *LB I*). No obstante, al aumentar el número de clientes, la diferencia entre el dispositivo C2 y el balanceador III (definido en las gráficas como *LB III*) tiende a reducirse. Esa diferencia se hace mínima en las dos últimas gráficas, al alcanzar el máximo número de conexiones concurrentes permitidas. Por otra parte, el balanceador II (definido en las gráficas como *LB II*) es el menos eficiente de forma consistente, a excepción de cuando se alcanza el máximo número de conexiones para 5 servidores, en ese caso el rendimiento es muy similar al de los otros balanceadores.

4.1.8 Conclusión

Esta sección pretende demostrar que es posible prestar servicios TCP balanceados en dispositivos de recursos limitados. Para ello se ha analizado la formación de un pequeño clúster de servidores C2, y en consecuencia se ha comparado su rendimiento de red con tres tipos de balanceadores en un experimento real.

En general, se puede afirmar que el hardware apropiado depende de la aplicación y sus requisitos. El dispositivo C2 (LCP1768) tiene unos recursos mucho más limitados que los otros dos balanceadores. Sin embargo, ¿es adecuado para gestionar grupos pequeños de servidores que comparten su limitación de recursos?

Para contestar esta pregunta se ha realizado un ANOVA factorial sobre los datos recogidos en el experimento. Hay que ser prudentes a la hora de extraer conclusiones, puesto que como se ha señalado previamente, este análisis debe usarse solamente con una finalidad exploratoria. En cualquier caso, sus resultados pueden emplearse para valorar la influencia que los diferentes factores analizados tienen en el rendimiento de red del clúster. A la vista de los resultados, se puede determinar que el número de clientes es el factor que afecta más al rendimiento de red, y que el balanceador C2 es estadísticamente más eficiente que los otros dos balanceadores, aunque su rendimiento se iguala a medida que aumenta el número de clientes. Esto puede deberse a que se está alcanzando la capacidad máxima del clúster de servidores C2, y por tanto el balanceador utilizado deja de ser un factor determinante. Incluso en el supuesto de que el clúster tuviera más capacidad, el número de descriptores disponibles en el controlador de acceso al medio (EMAC) del balanceador C2 también podría ser un factor limitante. En estos descriptores se define la dirección de la posición de memoria desde donde el controlador lee



(a) 2 servidores (b) 3 servidores

(c) 4 servidores (d) 5 servidores

Figura 4.4 Efecto de la interacción triple en el rendimiento de red (Mbps)

y escribe la trama Ethernet para su transmisión y recepción. Para el LCP1768, los descriptores y sus estructuras de datos asociadas deben estar ubicados solamente en la memoria RAM periférica (32kB máx.), y por tanto son muy escasos. En este caso, y para llegar a un equilibrio entre rendimiento y uso de recursos, se han utilizado cinco descriptores para transmisión y otros cinco para recepción. En caso de múltiples clientes (muchos paquetes entrantes), el controlador podría descartar paquetes y limitar el rendimiento del balanceador. Mathis et al. [1997] propusieron un modelo para el rendimiento de TCP que tiene en cuenta algunas características del enlace, como el tamaño máximo del segmento (MSS), el tiempo de ida y vuelta (RTT) y la pérdida de paquetes. Por lo tanto, un número reducido de descriptores puede provocar la pérdida de paquetes y un menor rendimiento.

No obstante, se puede concluir que un dispositivo C2 es capaz de proporcionar servicios más propios de otro tipo de equipos de red más avanzados. En este caso se ha demostrado que un dispositivo C2 puede actuar como balanceador de carga para clústeres de hasta 5 servidores C2 y 20 clientes concurrentes sin reducir su rendimiento de red en comparación con balanceadores de características muy superiores a las definidas en el RFC 7228. Por lo tanto, se confirma la hipótesis de investigación planteada.

Por último, y aunque estos factores no han formado parte del experimento, es interesante señalar que los dispositivos C2 podrían reducir también el coste de ciertos clústeres y su consumo de energía. El estado del arte ya presenta trabajos de investigación que utilizan pequeños clústeres de dispositivos embebidos como servidores web, y comparan su consumo de energía con servidores más tradicionales y potentes [Mohaghegh y Aboelaze, 2012]. En general, el coste total de un sistema construido utilizando un microprocesador es mayor que el de un microcontrolador, debido en parte a la necesidad de más componentes externos. De forma excepcional el LCP1768 es más caro que la Raspberry Pi, principalmente porque se produce en menores cantidades. En cuanto al consumo de energía, un microcontrolador funciona a una menor frecuencia, tiene una menor disipación de energía y además necesita un menor número de componentes externos. En este caso concreto, la Raspberry necesita unos 2W mientras que el LPC1768 solo 1W.

4.2 SYN cookies

Esta sección presenta el experimento que evalúa en un dispositivo C2 dos técnicas de mitigación frente a inundaciones SYN que figuran en el RFC 4987, y que se aplican cuando la cola de conexiones se llena:

- Reciclar conexiones semiabiertas: los nuevos paquetes SYN sustituyen las entradas más antiguas de la cola de conexiones en estado SYN-RECEIVED.
- SYN cookies: este mecanismo retrasa el almacenamiento en la cola hasta que se haya recibido un ACK válido.

En primer lugar, se describe una configuración que crea un entorno de red aislado que proporciona una capacidad de control de los factores importantes de un ataque de inundación SYN de baja tasa. El

ataque SYN se genera desde un equipo en esta red, y se dirige a un dispositivo C2, también conectado al mismo segmento de red.

Esta configuración permite evaluar el servicio ofrecido por el dispositivo C2, ya que la estrategia de asignación de estado por la pila TCP para cada paquete SYN recibido dicta la degradación del rendimiento de red que resulta de una inundación SYN. Los resultados obtenidos demostrarán si una estrategia de SYN cookies es apropiada para dispositivos C2, o por el contrario sustituir conexiones semiabiertas es más apropiado.

Al igual que en el experimento anterior, se ha utilizado el rendimiento de red en la dirección del cliente al servidor como la métrica de evaluación.

Por último, la validación del rendimiento de red ante ataques SYN de baja tasa se hace solo con el servidor C2 porque el objetivo es determinar qué configuración en la cola de conexiones penaliza menos. Es decir, es un estudio que compara dos técnicas de mitigación del servidor, y en la configuración de balanceo elegida, el balanceador penalizaría de la misma forma para las dos estrategias de mitigación analizadas. En resumen, para evitar que interacciones entre variables modifiquen el resultado y desvíen las conclusiones se ha analizado la contribución de seguridad por separado, sin el balanceador. Además, el balanceador de carga trabaja en la capa 2 del modelo OSI, mientras que las estrategias de mitigación ante ataques SYN actúan en la capa 4. El modelo de referencia OSI para la creación de redes está diseñado en torno a siete capas. La arquitectura de seguridad del modelo OSI también está diseñado en torno a esas siete capas, lo que refleja que cada capa del modelo tiene diferentes requisitos de seguridad, y por tanto deben evaluarse de manera independiente.

4.2.1 Planteamiento del problema

Antes de comenzar la fase de experimentación, se plantea el problema que se quiere estudiar. El presente experimento pretende aportar información en relación a la siguiente pregunta: ¿en qué medida las diferentes estrategias de mitigación ante ataques SYN repercuten en el rendimiento de red de un dispositivo C2? Para responder a esta pregunta es necesario:

- Definir los elementos hardware y software necesarios para el experimento.
- Diseñar una red aislada que permita ejecutar un ataque SYN y medir el rendimiento de red.
- Definir un método para generar paquetes SYN para provocar un ataque DoS.
- Investigar la degradación del servicio ofrecido por el dispositivo C2 durante un ataque SYN de baja tasa.
- Investigar si diferentes tamaños de la cola de conexiones del dispositivo C2 degradan o mejoran el rendimiento de red.
- Investigar si una estrategia de SYN cookies, en vez de sustituir entradas en la cola de conexiones, degrada o mejora el rendimiento de red.

4.2.2 Configuración del experimento

Esta sección presenta la configuración detallada del experimento. En primer lugar se presentan la configuración hardware y de red, y después se procede a explicar el software de utilidad necesario, como el generador de paquetes para crear el ataque SYN.

4.2.2.1 Configuración hardware

En este apartado se definen los equipos y el conmutador de red que constituyen la parte hardware del experimento. Se utilizan tres equipos en el banco de pruebas: i) el equipo atacante, que actúa como fuente de los paquetes de inundación SYN, ii) el equipo víctima, que actúa como receptor de los paquetes SYN del equipo atacante y además presta el servicio TCP, y iii) el equipo cliente, que accede al servicio TCP del equipo víctima.

A continuación se presenta un resumen que cubre las características más importantes de los equipos mencionados.

1. Atacante:
    - Sistema operativo: Ubuntu 12.04
    - Procesador: Intel Pentium 4
    - Núcleos: 1
    - Velocidad procesador: 3.4GHz
    - Modelo: Dell Dimension 5150
    - Memoria RAM: 3GB DDR2 SDRAM
    - Tarjeta de red integrada: Ethernet 10/100 Mbps
2. Víctima/servidor:
    - Sistema operativo: mbed OS
    - Procesador: ARM Cortex M3
    - Núcleos: 1
    - Velocidad procesador: 96MHz
    - Modelo: LPC1768
    - Memoria RAM: 64kB (32kB para Ethernet)
    - Controlador de acceso al medio: Ethernet 10/100 Mbps
3. Cliente:
    - Sistema operativo: Ubuntu 14.04
    - Procesador: Core i5 430M

- Núcleos: 4
- Velocidad procesador: 2.26GHz
- Modelo: Dell Vostro 3300
- Memoria RAM: 6GB DDR3 SDRAM
- Tarjeta de red integrada: Ethernet 10/100/1000 Mbps

4. Conmutador de red:

- Modelo: D-Link DES-1008F
- Gestionable: No
- Velocidad: 10/100 Mbps (10/100BaseT)
- Puertos: 8 puertos RJ45
- Auto-Negociación: Velocidad, dúplex y control de flujo

4.2.2.2 Configuración de red

El conmutador de red proporciona la conectividad de red necesaria entre los tres equipos, formando así una red de área local (LAN) entre esos equipos. Una vez conectados al conmutador, hay que configurar todos los equipos con direcciones IP y máscaras que pertenezcan a la misma subred, a fin de que los diferentes equipos en esa subred puedan comunicarse entre sí. En este caso, se utiliza la subred 10.48.1.x. Antes de realizar los experimentos, se comprueba la conectividad de red en todos los equipos utilizando la herramienta *ping* para enviar mensajes ICMP entre equipos.

4.2.2.3 Configuración software

En esta sección se describen las herramientas software necesarias para ejecutar el experimento. En primer lugar, se instala el generador de paquetes en el equipo atacante. El equipo víctima (LPC1768) implementa un servidor TCP y es el destino de la inundación SYN. Por último, en el equipo cliente se instala la herramienta de red que proporciona el acceso al servicio TCP, y mide el rendimiento de red para esa conexión. Estas tres herramientas software forman parte de la configuración software del experimento, y a continuación se explican más en detalle el generador de paquetes y la herramienta de medición utilizada en el cliente. El servidor ya fue explicado en detalle en la sección de implementación del capítulo 3.

Inundar la víctima es relativamente sencillo. *hping* es una herramienta de línea de comandos que permite generar y analizar paquetes TCP/IP, y como tal tiene muchas utilidades, como crear paquetes SYN y determinar su tasa de envío. Para ello, es necesario introducir la dirección IP del equipo que se desea atacar, y después, especificar el número de puerto, la duración del ataque, la tasa de envío, y otros parámetros que permitan caracterizar el ataque. A continuación se muestra un ejemplo:

Listado 4.6 Ejemplo hping

```
$ sudo hping3 -i u5000 -S -p 45678 10.48.1.32
HPING 10.48.1.32 (eth0 10.48.1.32): S set, 40 headers + 0 data bytes

--- 10.48.1.32 hping statistic ---
149610 packets transmitted, 0 packets received, 100\% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

donde:

- -p 45678 es el puerto atacado.
- -S activa el flag SYN.
- -i u5000 indica que se esperan 5000 microsegundos entre el envío de paquetes (200 paquetes/s).
- 10.48.1.32 es la IP de destino, es decir la víctima.

Conviene destacar que puesto que no se está modificando la dirección IP de origen, los paquetes SYN-ACK de retorno llegarán al atacante, y como tampoco se está completando la fase de establecimiento TCP, el sistema operativo podría tratar de tomar el control y empezar a responder con un paquete RST, liberando de esta forma las entradas de la cola de conexiones del equipo víctima.

Esencialmente, el problema es que el programa de ataque se ejecuta en el espacio de usuario, y por lo tanto la pila TCP/IP recibirá el SYN-ACK primero. La pila enviará un RST ya que no tiene una conexión abierta para ese número de puerto. Para evitar este comportamiento se debe introducir una regla IP para bloquear cualquier paquete RST saliente:

Listado 4.7 Regla de tráfico

```
iptables -A OUTPUT -p tcp -s 10.48.1.46 --tcp-flags RST RST -j DROP
```

En este caso, la dirección IP 10.48.1.46 es la dirección de origen del ataque y la regla evita que se conteste con un RST a conexiones no establecidas. De esta forma, con *hping* y la regla anterior se consigue inundar el equipo víctima con paquetes SYN a una tasa constante.

En cuanto a la herramienta en el equipo cliente, se vuelve a utilizar Iperf como en los experimentos anteriores. Iperf es una herramienta construida con un modelo cliente-servidor, en la que el rendimiento de red se puede medir en las dos direcciones. En este caso, se ha replicado el funcionamiento de un servidor Iperf básico en el dispositivo C2. El servidor reconoce los datos del cliente lo más rápidamente posible, pero sin enviar datos de retorno. Por tanto, las medidas de rendimiento de red se hacen desde el equipo cliente.

4.2.3 Definición del experimento

A continuación, se describe la metodología para medir el impacto de un ataque de inundación SYN en el servicio del dispositivo C2, utilizando para ello dos estrategias de mitigación: reemplazar conexiones

semiabiertas y SYN cookies. Como en los experimentos anteriores, se utiliza el rendimiento de red –en Mbps– como métrica de evaluación. Sin ataque, el rendimiento medio de red es de 33,72 Mbps para una conexión al servidor C2 (visto en el capítulo 3). En caso de ataque, el servidor deberá dedicar recursos a procesar paquetes SYN falsos. El experimento pretende medir el deterioro que ese proceso tiene en el rendimiento de red de una conexión activa. La variable independiente será la configuración utilizada, y la variable dependiente será el rendimiento de red.

Como ya se ha comentado en el capítulo anterior, lwIP tiene dos estrategias de asignación de estado por el proceso TCP para las conexiones entrantes: backlog y cola de prioridad.

lwIP tiene una única cola de conexiones muy limitada, cuyo tamaño está definido por la variable MEMP_NUM_TCP_PCB. El valor definido en esa variable será el número de conexiones activas que podrá mantener el dispositivo al mismo tiempo, respetando siempre la memoria disponible. En caso de ataque, esta cola de conexiones mantendrá peticiones de conexión semiabiertas (estado SYN-RECEIVED) hasta 3 minutos, por defecto, si está activada la opción backlog. En este caso, cuando la cola llega a su máxima capacidad, lwIP descartará los paquetes SYN entrantes hasta que la cola tenga espacio suficiente para aceptar nuevas conexiones. Esta configuración no se va a probar ya que no hace falta ningún experimento para saber que si se manda el número de paquetes SYN equivalentes al número de posiciones de la cola cada 3 minutos, se mantendrá la cola llena y se dejará sin servicio a los clientes. Es decir, en caso de ataque, la configuración backlog no es apropiada para un dispositivo C2.

La cola de prioridad, en cambio, sustituye cada nueva entrada por la conexión más antigua presente en la cola, y por tanto no bloquea el funcionamiento del servidor en caso de ataque. Esta configuración de lwIP permite que los SYN entrantes puedan sustituir la entrada tcp_pcb semiabierta más antigua, y por tanto se comportaría como una de las defensas comunes descritas en el RFC 4987: “el reciclaje de los TCB semiabiertos más antiguos”. Cuando un cliente consigue establecer una conexión en presencia de un ataque SYN de baja tasa, esa conexión ocuparía un tcp_pcb, mientras que el resto de entradas de la cola serían ocupadas por peticiones SYN falsas (en estado SYN-RECEIVED). Estas entradas irían sustituyéndose por nuevos paquetes SYN del atacante.

En definitiva, esta configuración podría ser apropiada para un dispositivo C2 ya que mantendría el funcionamiento del servidor, aunque el ataque y el procesado derivado para reciclar una entrada en la cola de prioridad podrían tener un impacto significativo en el rendimiento de red.

A continuación, se muestran las variables cualitativas utilizadas para designar los diferentes grupos o configuraciones de la variable independiente:

- Cola(2): el tamaño de la cola de conexión (MEMP_NUM_TCP_PCB) es 2, y si se llena, se reemplaza la entrada semiabierta más antigua.
- Cola(4): el tamaño de la cola de conexión es 4, y si se llena, se reemplaza la entrada semiabierta más antigua.
- Cola(8): el tamaño de la cola de conexión es 8, y si se llena, se reemplaza la entrada semiabierta más antigua.

- Cola(16): el tamaño de la cola de conexión es 16, y si se llena, se reemplaza la entrada semiabierta más antigua.
- Cookies: en lugar de reemplazar las entradas más antiguas de la cola de conexiones, los paquetes SYN entrantes se procesan con cookies.

Como en los experimentos anteriores, se obtienen 30 muestras para cada configuración [Corder y Foreman, 2009]. Este tamaño de la muestra proporcionará suficientes observaciones para determinar la configuración que degrada menos el rendimiento de red del servicio proporcionado por el dispositivo C2 ante un ataque SYN.

En función del volumen de paquetes SYN, los ataques pueden clasificarse en dos tipos. En los ataques de alta tasa, el atacante envía multitud de paquetes SYN con el objetivo de saturar la cola de conexiones de la víctima lo más rápidamente posible. Por el contrario, los ataques de baja tasa tienen por objetivo degradar las conexiones TCP de manera más sutil, en ocasiones simulando tráfico legítimo. Kuzmanovic y Knightly [2006] fueron los primeros en identificar y caracterizar este tipo de ataque, en el que el atacante envía periódicamente flujos de paquetes con el objetivo de provocar la desconexión de flujos legítimos TCP y reducir su tasa de transmisión de una manera más desapercibida. Este tipo de ataque resulta muy efectivo en dispositivos C2, ya que el número de entradas en la cola de conexiones es muy limitado.

Muchos esquemas de detección están mejor diseñados para detectar fuentes de ataque de alta tasa [Sun et al., 2012]. Sin embargo, en los ataques DDoS actuales la inundación SYN suele distribuirse entre muchos equipos utilizando ataques de baja tasa para hacer la detección más difícil [Nashat et al., 2008]. Por lo tanto, un ataque SYN de baja tasa hace que el ataque sea más difícil de detectar, logrando el mismo resultado: el agotamiento de los recursos de la víctima. Por otra parte, no es factible que un servidor C2 sea capaz de resistir un ataque SYN de alta tasa generado por un equipo mucho más potente.

Por estos motivos se ha establecido la tasa de envío de paquetes SYN a 5000μs, es decir, un ataque que se podría catalogar de baja tasa en ordenadores de sobremesa, ya que equivale a enviar 200 paquetes por segundo, pero para un dispositivo C2 y su limitada cola de conexiones se trata de una cantidad significativa para averiguar qué configuración funciona mejor. Una vez decidida la tasa, solo hay que establecer la dirección IP de destino a la del equipo víctima y el puerto del servicio TCP con el comando *hping*. El ataque prevalecerá hasta que se pare el proceso.

Por último, una vez definidos el escenario del experimento y la tasa de ataque, solo queda configurar el equipo cliente para que realice conexiones TCP al servidor C2 y poder así obtener las medidas del rendimiento de red para las configuraciones definidas.

Al igual que en los experimentos anteriores, se ha desarrollado un script para automatizar la ejecución de múltiples instancias. El script asume que hay un solo servidor en funcionamiento y ejecuta múltiples clientes en serie. Entre cada ejecución, el script entra en un estado de inactividad por un período corto de tiempo.

En este caso el script toma cinco argumentos de entrada:

- Dirección IP del servidor (-c), en este caso 10.48.1.32.
- Número de puerto del servidor (-p), en este caso 45678.
- Tiempo de cada prueba (-t), en este caso 30.
- Número de pruebas a ejecutar, en este caso 30 muestras.
- Tiempo medio de inactividad entre las pruebas en segundos, en este caso 1.

Por último, se redirige la salida del script (el rendimiento medio de red para cada conexión) a un fichero. Una vez obtenidos todos los datos, se procesan con el mismo script en Python que en los anteriores experimentos para convertirlo en un fichero CSV. Este fichero CSV se utiliza para el posterior análisis con la herramienta estadística SPSS, cuyos resultados se presentan en el siguiente apartado.

4.2.4 Resultados y análisis

Para este experimento se ha realizado un análisis ANOVA de un factor. Este análisis sirve para comparar las medias de una variable dependiente en dos o más grupos.

En este caso se quiere contrastar a un nivel de confianza del 95 % si existen diferencias significativas respecto al rendimiento medio de red según el tipo de configuración, diferentes tamaños de cola para el reciclado de conexiones o SYN cookies, y si es así, determinar entre qué configuraciones se dan tales diferencias.

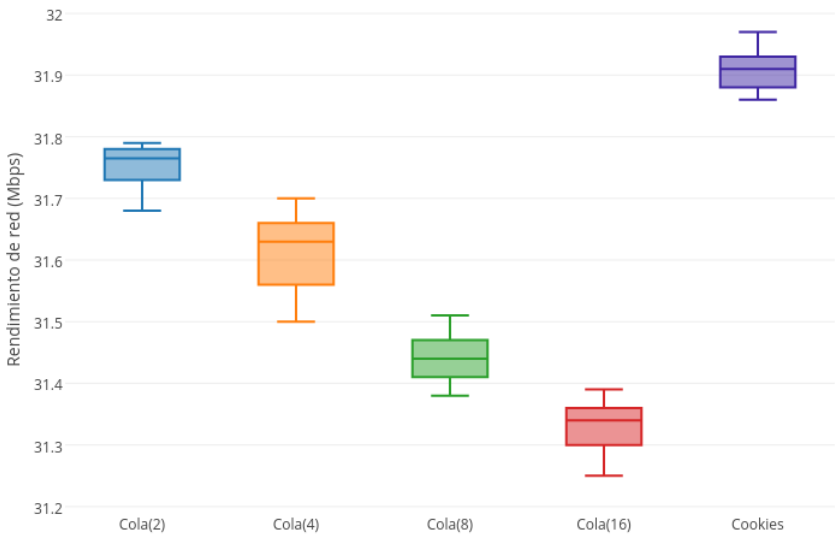


Figura 4.5 Diagrama de cajas

El gráfico 4.5 parece indicar que existen diferencias significativas en el rendimiento de red, en especial con las SYN cookies ya que sus intervalos no se solapan con ninguna otra configuración. En el resto de configuraciones sí hay solape. Por último, no se observa la presencia de valores atípicos. Una vez analizado el gráfico, se procede con el ANOVA de un factor.

Antes de contrastar la hipótesis de igualdad de medias, se va a comprobar si se cumple uno de los supuestos de aplicación del ANOVA, la homogeneidad, para determinar el procedimiento *post hoc* más apropiado.

La prueba de Levene establece como hipótesis nula la igualdad de varianzas de los grupos. En la tabla 4.12, junto con el valor estadístico de Levene, se muestran los grados de libertad de su distribución y el valor de significación. Puesto que el valor de significación es menor que 0,05, es muy poco probable que el azar fuese responsable de las diferencias obtenidas en las varianzas de las muestras. De esta forma se rechaza la hipótesis nula de que los grupos tienen varianzas iguales, y en definitiva, se puede determinar que en los grupos definidos por las cinco configuraciones las varianzas respecto a la variable dependiente no son iguales.

Los resultados de la prueba de Levene arrojan también una segunda lectura. Todavía no se ha analizado si las medias son distintas, pero ya se sabe que las varianzas lo son. Esto puede evidenciar que los grupos no han sido seleccionados a partir de poblaciones idénticas.

Tabla 4.12 Prueba de Levene sobre homogeneidad de varianzas

	Estadístico de Levene	**gl1**	**gl2**	**Sig.**
Rendimiento de red	11,348	4	145	0,000

Como ya se ha explicado en el apartado 4.1.4, de todos los supuestos, el de independencia es el más importante ya que el ANOVA no admitirá su violación. No obstante, tolera violaciones a los supuestos de normalidad y homogeneidad. Además, en este caso se trata de un análisis exploratorio, no confirmatorio, para deducir qué configuración tiene un mayor rendimiento medio de red.

Una vez analizados los supuestos, se procede a aplicar el ANOVA. La tabla 4.13 muestra el resultado.

Tabla 4.13 Resumen del procedimiento ANOVA de un factor

	Suma de cuadrados	**gl**	**Media cuadrática**	**F**	**Sig.**
Inter-grupos	6,573	4	1,643	938,966	0,000
Intra-grupos	0,254	145	0,002		
Total	6,827	149			

El ANOVA se basa en la descomposición de la variación total de los datos con respecto a la media global. Para ello se utiliza la variación dentro de las muestras (Intra-grupos), que cuantifica la dispersión de los valores de cada muestra con respecto a sus correspondientes medias, y la variación entre muestras (Inter-grupos), que cuantifica la dispersión de las medias respecto a la media total.

Para un nivel de significación del 5 %, el valor de significación resultante (0,000) lleva a rechazar la hipótesis nula de igualdad de medias. A la vista de los resultados, se concluye que la configuración utilizada repercute en el rendimiento de red.

Tabla 4.14 Estadísticos descriptivos del procedimiento ANOVA

	Cola de prioridad				
	Cola(2)	**Cola(4)**	**Cola(8)**	**Cola(16)**	**Cookies**
N	30	30	30	30	30
Media	31,7557	31,6113	31,4400	31,3297	31,9107
Desviación estándar	0,03126	0,06307	0,03778	0,03891	0,02924
Error típico	0,00571	0,01151	0,00690	0,00710	0,00534
Límite inferior	31,7440	31,5878	31,4259	31,3151	31,8997
Límite superior	31,7673	31,6349	31,4541	31,3442	31,9216
Mínimo	31,68	31,50	31,38	31,25	31,86
Máximo	31,79	31,70	31,51	31,39	31,97

La tabla 4.14 muestra para cada grupo el número de medidas, la media, la desviación estándar, el error típico de la media, los límites del intervalo de confianza para la media al 95 %, y los valores mínimo y máximo. El objetivo es investigar si procesar paquetes SYN falsos afecta de igual manera a todas las configuraciones.

Como se ha visto en la tabla 4.13, es posible determinar que existen diferencias entre las medias. Para saber entre qué configuraciones se encuentran las diferencias se realiza un contraste de comparaciones múltiples, denominado *post hoc*.

Como ya se ha explicado, se pueden utilizar diferentes procedimientos *post hoc* dependiendo de si se asumen las varianzas como iguales o no. Si se asumen como iguales, el procedimiento más común es el de Bonferroni. Si por el contrario no es posible suponer varianzas iguales, el procedimiento más utilizado es el de Games-Howell ya que controla mejor la tasa de error. Este método se basa en la distribución del rango estudentizado (la distribución que sigue la diferencia del máximo y del mínimo de las diferencias entre la media muestral y la media poblacional de un número de variables independientes e idénticamente distribuidas) y en un estadístico T en el que se corrigen los grados de libertad mediante la ecuación de Welch.

Las tablas 4.15 y 4.16 presentan los resultados de los procedimientos seleccionados. En la primera columna aparece el grupo de referencia (I) con el cual se compara. La segunda columna representa a cada uno de los grupos restantes (J). Por lo tanto en las tablas aparecen todas las posibles combinaciones, las diferencias entre los rendimientos medios de red de cada combinación, el error típico de esas diferencias y el nivel de significación asociado a cada diferencia.

La hipótesis nula que se pone a prueba con cada combinación es que las medias comparadas son iguales. Cuando las medias de los grupos comparados difieren al nivel establecido (0,05), se acepta que existe una diferencia estadística significativa con un nivel de confianza del 95 % entre las medias de esos dos grupos, y esa combinación aparece marcada en las tablas con un asterisco. A la vista de

Tabla 4.15 Comparaciones múltiples del procedimiento ANOVA de un factor (Bonferroni)

(I) Configuración	(J) Configuración	Diferencia medias (I-J)	Error típico	Sig.	Intervalo de confianza al 95 % Límite inferior	Límite superior
Cola(2)	Cola(4)	0,14433*	0,01080	0,000	0,1135	0,1751
	Cola(8)	0,31567*	0,01080	0,000	0,2849	0,3465
	Cola(16)	0,42600*	0,01080	0,000	0,3952	0,4568
	Cookies	-0,15500*	0,01080	0,000	-0,1858	-0,1242
Cola(4)	Cola(2)	-0,14433*	0,01080	0,000	-0,1751	-0,1135
	Cola(8)	0,17133*	0,01080	0,000	0,1405	0,2021
	Cola(16)	0,28167*	0,01080	0,000	0,2509	0,3125
	Cookies	-0,29933*	0,01080	0,000	-0,3301	-0,2685
Cola(8)	Cola(2)	-0,31567*	0,01080	0,000	-0,3465	-0,2849
	Cola(4)	-0,17133*	0,01080	0,000	-0,2021	-0,1405
	Cola(16)	0,11033*	0,01080	0,000	0,0795	0,1411
	Cookies	-0,47067*	0,01080	0,000	-0,5015	-0,4399
Cola(16)	Cola(2)	-0,42600*	0,01080	0,000	-0,4568	-0,3952
	Cola(4)	-0,28167*	0,01080	0,000	-0,3125	-0,2509
	Cola(8)	-0,11033*	0,01080	0,000	-0,1411	-0,0795
	Cookies	-0,58100*	0,01080	0,000	-0,6118	-0,5502
Cookies	Cola(2)	0,15500*	0,01080	0,000	0,1242	0,1858
	Cola(4)	0,29933*	0,01080	0,000	0,2685	0,3301
	Cola(8)	0,47067*	0,01080	0,000	0,4399	0,5015
	Cola(16)	0,58100*	0,01080	0,000	0,5502	0,6118

Tabla 4.16 Comparaciones múltiples del procedimiento ANOVA de un factor (Games-Howell)

(I) Configuración	(J) Configuración	Diferencia medias (I-J)	Error típico	Sig.	Intervalo de confianza al 95 % Límite inferior	Límite superior
Cola(2)	Cola(4)	0,14433*	0,01285	0,000	0,1077	0,1809
	Cola(8)	0,31567*	0,00895	0,000	0,2904	0,3409
	Cola(16)	0,42600*	0,00911	0,000	0,4003	0,4517
	Cookies	-0,15500*	0,00781	0,000	-0,1770	-0,1330
Cola(4)	Cola(2)	-0,14433*	0,01285	0,000	-0,1809	-0,1077
	Cola(8)	0,17133*	0,01342	0,000	0,1333	0,2094
	Cola(16)	0,28167*	0,01353	0,000	0,2433	0,3200
	Cookies	-0,29933*	0,01269	0,000	-0,3355	-0,2631
Cola(8)	Cola(2)	-0,31567*	0,00895	0,000	-0,3409	-0,2904
	Cola(4)	-0,17133*	0,01342	0,000	-0,2094	-0,1333
	Cola(16)	0,11033*	0,00990	0,000	0,0825	0,1382
	Cookies	-0,47067*	0,00872	0,000	-0,4953	-0,4461
Cola(16)	Cola(2)	-0,42600*	0,00911	0,000	-0,4517	-0,4003
	Cola(4)	-0,28167*	0,01353	0,000	-0,3200	-0,2433
	Cola(8)	-0,11033*	0,00990	0,000	-0,1382	-0,0825
	Cookies	-0,58100*	0,00889	0,000	-0,6061	-0,5559
Cookies	Cola(2)	0,15500*	0,00781	0,000	0,1330	0,1770
	Cola(4)	0,29933*	0,01269	0,000	0,2631	0,3355
	Cola(8)	0,47067*	0,00872	0,000	0,4461	0,4953
	Cola(16)	0,58100*	0,00889	0,000	0,5559	0,6061

los resultados, se concluye que todas las medias comparadas difieren significativamente, es decir, la configuración de cookies tiene un rendimiento medio mayor que el reciclado de conexiones con un tamaño de cola de 2, y ésta configuración a su vez tiene un rendimiento medio mayor que con un tamaño de 4, y así sucesivamente. En definitiva, se rechaza la hipótesis nula para todas las combinaciones. Por último, se observa que el número de diferencias significativas detectadas es el mismo con los dos métodos utilizados, aunque como no es posible asumir varianzas iguales (tabla 4.12), se debería prestar más atención a los resultados del método Games-Howell.

Tabla 4.17 Subgrupos homogéneos del procedimiento ANOVA de un factor

		Subgrupo para α=0,05				
Configuración	**N**	**1**	**2**	**3**	**4**	**5**
Cola de prioridad (16)	30	31,3297				
Cola de prioridad (8)	30		31,4400			
Cola de prioridad (4)	30			31,6113		
Cola de prioridad (2)	30				31,7557	
Cookies	30					31,9107
Sig.		1,000	1,000	1,000	1,000	1,000

La tabla 4.17 ofrece una clasificación de los grupos basada en el grado de parecido existente entre sus medias, usando como tamaño la media armónica de los tamaños de cada grupo, en este caso 30.

Los grupos se enumeran en orden ascendente de medias. Las medias que aparecen dentro del mismo subconjunto comprenden un conjunto de medias que no son significativamente diferentes. Se puede observar que en cada subgrupo está incluido un solo grupo o configuración, lo que significa que la media del rendimiento de red de cada configuración difiere significativamente de la media de las otras cuatro configuraciones restantes, y que obviamente no difiere de sí mismo (Sig. = 1,000). Esta clasificación por subgrupos no está disponible con todos los procedimientos *post hoc*. Aunque no es posible asumir que las varianzas sean iguales, este análisis se ha tenido que realizar con el procedimiento de Scheffé.

A la vista del análisis exploratorio, queda demostrado que las medias de los grupos son diferentes, ahora queda analizar el por qué. En primer lugar, se va a explicar en detalle la notación Big-O. Cuando se trabaja con algoritmos o funciones es interesante analizar su tiempo de ejecución. La notación Big-O es la representación de la complejidad de un algoritmo o función, y esa complejidad tiene en cuenta cómo se comporta el algoritmo cuando la entrada de datos aumenta. De esta forma, se puede categorizar el rendimiento de un algoritmo.

La tabla 4.18 muestra las complejidades más habituales.

Conviene señalar que el resultado de ejecución puede no coincidir exactamente con alguna de estas complejidades, pero se consideran como buenas aproximaciones.

En este experimento se comparan dos estrategias: la primera recicla las estructuras de datos de lwIP (tcp_pcb) semiabiertas más antiguas en caso de no tener más espacio disponible para una nueva conexión, y la segunda computa una cookie. Ambas estrategias se ejecutan cuando se recibe un paquete SYN.

Tabla 4.18 Complejidades habituales

Tipo	Notación	Significado
Constante	O(1)	La ejecución es independiente del número de elementos a procesar
Logarítmica	O(log(n))	La ejecución crece de forma logarítmica con respecto al número de elementos a procesar
Lineal	O(n)	La ejecución crece de forma lineal con respecto al número de elementos a procesar
N Log N	O(n*log(n))	La ejecución crece como un producto de complejidad lineal y logarítmica
Cuadrática	O(n^2)	La ejecución crece de forma cuadrática con respecto al número de elementos a procesar

A continuación, se muestra el orden de ejecución que sigue la función encargada de reservar una entrada en la cola de conexiones cuando se recibe un paquete SYN:

1. Intenta asignar una nueva estructura tcp_pcb (memp_alloc).
2. Si no es posible, intenta eliminar la conexión más antigua en estado TIME-WAIT (tcp_kill_timewait).
3. Intenta asignar otra vez un tcp_pcb.
4. Si no es posible, intenta eliminar la conexión activa más antigua que tiene menor prioridad que la especificada (tcp_kill_prio (u8_t prio)).
5. Libera esa conexión y su estructura de datos (memp_free).
6. Intenta asignar otra vez un tcp_pcb.
7. En este caso, la función memp_alloc es capaz de asignar un tcp_pcb.
8. Se procesa la solicitud TCP (paquete SYN) y se configura esa estructura tcp_pcb.

Como se puede observar, lwIP recicla las conexiones semiabiertas más antiguas después de intentar reemplazar las conexiones en estado TIME-WAIT. Este comportamiento es lógico teniendo en cuenta que ante un tráfico normal es más prioritario liberar conexiones en estado TIME-WAIT que en SYN-RECEIVED. Por otra parte, es preciso destacar que por defecto todas las conexiones entrantes tienen la misma prioridad, por lo que la cola de prioridad acaba comportándose como una FIFO. Habría que desarrollar funciones más avanzadas para dar una mayor prioridad a ciertas conexiones, pero este tipo de funciones aumentan la complejidad del sistema y no tienen por qué dar buenos resultados en caso de ataque.

Dado un array, una lista o cualquier otro tipo de estructura de datos, el tiempo de búsqueda será O(n) si no está ordenado, y O(log(n)) si está ordenado. En este caso, lwIP realiza dos búsquedas lineales de conexiones en una estructura de datos sin ordenar. A esto hay que añadirle el tiempo necesario para procesar el paquete SYN y guardar la información correspondiente en la estructura tcp_pcb (O(1)).

Por el contrario, se puede considerar que el cálculo de una cookie es O(1), ya que calcular la función hash tomará siempre el mismo tiempo suponiendo que los datos de entrada tengan la misma longitud. Una vez recibida y validada la cookie, habría que asignar un tcp_pcb, procesar el paquete SYN y configurar esa estructura de datos, pero esta vez para una conexión que ha demostrado ser legítima.

A la vista de los resultados, dado un tamaño de cola muy pequeño, el rendimiento de red se aproxima al de las cookies. Conforme el tamaño se incrementa, la diferencia en el rendimiento medio es más apreciable. Por lo tanto, se demuestra que la búsqueda lineal es la responsable del deterioro del rendimiento de red en el caso del reciclado de conexiones semiabiertas.

Lemon ya señaló en FreeBSD 4.4 que esta táctica tenía algunos aspectos negativos a medida que crecía el tamaño de la cola de conexiones [Lemon, 2002], derivados en su mayoría de la ineficiencia de las estructuras de datos y los algoritmos de búsqueda. Aunque este deterioro se apreciaba cuando la cola tenía una longitud por encima de varias centenas, la ineficiencia de los algoritmos de búsqueda tiene un impacto extrapolable a los dispositivos limitados.

A primera vista se puede pensar en varias alternativas. En primer lugar, se podría utilizar un hilo de ejecución que ordenase esas conexiones en la cola. Teniendo en cuenta que ordenar tiene un tiempo de ejecución de O(n*log(n)) y que en caso de ataque se estarían recibiendo multitud de conexiones en intervalos muy cortos de tiempo, el proceso estaría continuamente ordenando conexiones falsas y su rendimiento sería similar al caso actual (O(n*log(n))+O(log(n)) vs O(n)).

Otra opción sería utilizar una tabla hash en vez de la cola de conexiones. Como ya se comentó en el capítulo anterior, básicamente una tabla hash es un array en el que la posición de cada conexión se determina por una función hash (cualquier función que siempre asigne la misma entrada a la misma salida). Esta función se ejecuta en O(1), y por tanto se traduce en que la búsqueda o inserción sería también O(1) de media, pero habría que lidiar con colisiones al tener un tamaño de cola muy reducido, lo que podría ampliar la complejidad hasta su peor caso, O(n).

Teniendo en cuenta que el tiempo de ejecución del hash de la tabla y de la cookie serían iguales, con una complejidad O(1), utilizar cookies evitaría construir y gestionar estructuras de datos más complejas.

Una tercera opción pasaría por eliminar la primera búsqueda de las conexiones en estado TIME-WAIT. Aunque mejoraría la implementación original de la cola de prioridad de lwIP para el caso de un ataque SYN, seguiría siendo una búsqueda lineal, por lo que el tiempo de ejecución del reciclado de una conexión aumentaría de forma lineal con el tamaño de la cola.

Por último, cabe señalar una mejora adicional que se ha hecho al código de lwIP. La función tcp_kill_prio(u8_t prio) eliminaba cualquier conexión, y en este caso se ha priorizado que elimine una conexión en estado SYN-RECEIVED en vez de otra activa. Sin aplicar esta mejora adicional, un ataque SYN podría provocar que se eliminase una conexión activa que sea algo lenta, aún teniendo otras conexiones en estado SYN-RECEIVED (tiempo inactivo de la conexión en estado ESTABLISHED > tiempo inactivo de las conexiones falsas en estado SYN-RECEIVED).

4.2.5 Conclusión

Aunque el ataque de inundación SYN es conocido desde 1994, su mitigación aún está abierta a debate. De la revisión del estado del arte, se concluye que el trabajo realizado es el primero que aplica estrategias de mitigación que figuran en el RFC 4987 a dispositivos limitados (RFC 7228). En concreto, se quiere determinar cuál de los dos mecanismos utilizados cuando la cola de conexiones se llena funciona mejor

en un dispositivo C2. El experimento ha sido diseñado para medir la degradación del rendimiento de red respecto a cómo los paquetes SYN se procesan en el dispositivo C2. Se han utilizado cinco configuraciones: cuatro tamaños de cola diferentes para reciclar las conexiones semiabiertas más antiguas, y las SYN cookies.

A la vista de los resultados, existe una diferencia estadísticamente significativa entre esas configuraciones determinada por el ANOVA de un factor ($F(4,145) = 938{,}966$, $p = 0{,}000$). Aunque la diferencia entre las medias es pequeña, el análisis *post hoc* revela que esas diferencias en el rendimiento de red son estadísticamente significativas ($p = 0{,}000$). Además, el tamaño del efecto (d de Cohen) sugiere una relevancia práctica muy alta ($d > 0{,}8$) entre todas las configuraciones.

En definitiva, el rendimiento de red es mayor a medida que la cola de conexiones se hace más pequeña y, por otra parte, las SYN cookies son más eficaces que reciclar las conexiones semiabiertas más antiguas. Además, reciclar conexiones puede impedir establecimientos de conexión legítimos. Ciertas conexiones pueden necesitar más tiempo para establecerse que el necesario para llenar la cola de paquetes SYN falsos. A medida que el tamaño de la cola de conexiones disminuye, este resultado es más probable. De esta forma, las SYN cookies se convierten en el complemento perfecto para los servidores C2 en un esquema de balanceo DSR durante un ataque SYN de baja tasa.

No obstante, en la literatura son conocidos ciertos inconvenientes de las SYN cookies, como el hecho de que no toda la información de la conexión puede codificarse en el número de secuencia de 32 bits, y por tanto ciertas opciones necesarias para un alto rendimiento no pueden usarse. Además, se afirma que las SYN cookies alteran los procedimientos de sincronización TCP descritos en el RFC 793 al no reenviar paquetes SYN-ACK. Las implementaciones TCP/IP para dispositivos limitados, como lwIP, no implementan esas opciones para un alto rendimiento, ya que estas opciones requieren recursos que un dispositivo limitado no es capaz de proveer. En cuanto a no reenviar paquetes, un cliente legítimo realizará por lo general varios intentos de conexión al no recibir el SYN-ACK correspondiente, compensando de esta manera la incapacidad del servidor de reenviar paquetes SYN-ACK.

El problema surge cuando se pierde el paquete ACK que finaliza el establecimiento de la conexión y el protocolo de la capa de aplicación requiere que el servidor comience la comunicación (SMTP y SSH son dos ejemplos). En este caso, el cliente asume que la conexión se ha establecido con éxito y espera a que el servidor comience la comunicación o reenvíe el paquete SYN-ACK. Sin embargo, el servidor no guarda la conexión en memoria y por tanto no puede reenviar paquetes. TCP asume que la conexión está activa hasta que se demuestre lo contrario, por lo tanto el cliente solo podrá eliminar esta conexión una vez transcurrido el tiempo de espera de la capa de aplicación, y puede ser un tiempo relativamente largo. La solución sería utilizar las SYN cookies por puerto, es decir, únicamente para aquellas aplicaciones –identificadas por el número de puerto del servicio TCP asociado– en las que el cliente inicia el envío de datos una vez se establezca la conexión.

Por último, se considera que un dispositivo C2 debería utilizar dos colas distintas, como las versiones actuales de Linux: una cola SYN (o cola de conexiones incompletas) que guarde el estado mínimo de una conexión (la estructura tcp_pcb de lwIP necesita 168 bytes, demasiado para una conexión en estado SYN-RECEIVED), y una cola de conexiones completas. Las conexiones en estado SYN-RECEIVED se

añaden a la cola SYN, y posteriormente se mueven a la cola de conexiones completas cuando su estado cambia a ESTABLISHED, es decir, cuando se recibe el paquete ACK del cliente. Cuando la cola SYN está llena, las conexiones entrantes se procesan con SYN cookies. Si el ACK de retorno es válido, se crea una nueva conexión en la cola de conexiones completas. Finalmente, si se recibe un paquete ACK válido y la cola de conexiones completas está llena, se responde con un paquete RST.

CAPÍTULO

5

Conclusiones y líneas futuras

Este último capítulo resume los aspectos fundamentales del trabajo desarrollado. Se exponen las principales contribuciones realizadas, las conclusiones extraídas, y además se plantean posibles líneas de investigación que quedan abiertas.

5.1 Conclusiones y contribuciones

El objetivo de este trabajo es trasladar conceptos de la computación en la nube, como un alto rendimiento y escalabilidad, al borde más extremo de la red, en concreto a los dispositivos de recursos limitados. Para conseguirlo, el diseño de sistemas con dispositivos limitados –C2 según el RFC 7228– se debe basar en una infraestructura distribuida que sea transparente para el cliente, mediante una combinación de componentes hardware redundantes y un software para gestionar y distribuir el tráfico de red entrante. Este enfoque permitiría ejecutar servicios complejos que exigen más recursos que los ofrecidos por un único servidor C2, sin depender de dispositivos de recursos no limitados.

Para validar esta hipótesis se ha desplegado un clúster homogéneo de dispositivos C2 que ofrecen un servicio TCP y se ha analizado su rendimiento de red en presencia de conexiones concurrentes. En cuanto a los balanceadores seleccionados para la fase de experimentación, se han utilizado 3 dispositivos con características y recursos significativamente diferentes: un dispositivo C2, un ordenador de placa simple, y un ordenador de sobremesa de gama media. Como método de balanceo de carga se ha seleccionado DSR por su sencillez y eficiencia. Debido a la limitación de una única dirección IP impuesta por la versión utilizada de lwIP, se ha diseñado una estrategia alternativa para configurar DSR en el clúster de dispositivos C2.

El análisis estadístico (ANOVA) muestra que cuando el tráfico es alto, el balanceador de carga extiende el servicio a más clientes a los que permite acceder a una porción del ancho de banda. Además, cuando el número de clientes es el mismo, repartir ese tráfico entre más servidores mejora el rendimiento medio de red de esos clientes, lo que demuestra que crear clústeres y repartir el tráfico entre más servidores es una decisión acertada en dispositivos de recursos limitados. En términos generales, el balanceador C2 se comporta mejor que los otros dos balanceadores, aunque su rendimiento se iguala a medida que aumenta el número de clientes. Esto puede deberse a que se está alcanzando la capacidad

máxima del clúster de dispositivos C2, y por tanto el balanceador utilizado deja de ser un factor determinante.

A la vista de los resultados obtenidos en esta investigación, se confirma la hipótesis planteada: un clúster homogéneo de dispositivos de recursos limitados puede distribuir y procesar conexiones TCP concurrentes sin reducir su rendimiento de red respecto a balanceadores de carga de recursos no limitados.

En definitiva, se demuestra que la computación distribuida es factible en dispositivos C2. Un dispositivo C2 puede operar como hardware específico de red, en este caso como balanceador de carga, y dividir la carga de trabajo en tareas discretas (conexiones) que pueden ser resueltas por los procesadores de recursos limitados de los servidores C2. En principio, se trata de la primera implementación de un clúster de balanceo de carga formado exclusivamente por dispositivos de recursos limitados. Es por tanto un aporte original, con implementación y análisis real, además de un trabajo de investigación actual debido a la proliferación de este tipo de dispositivos en el paradigma IoT.

No obstante, el método de balanceo utilizado presenta un inconveniente que podría afectar al funcionamiento del clúster en caso de ataques de inundación. Debido a la naturaleza asimétrica de DSR, el balanceador no puede aplicar técnicas de defensa adecuadas para contrarrestar los efectos de un posible ataque. Por esta misma razón, es imprescindible que los servidores C2 implementen algún tipo de mecanismo de mitigación ante ataques de inundación SYN para mantener el correcto funcionamiento del clúster.

La literatura tasa estos ataques en más del 90 % de todos los ataques DoS observados en Internet [Nakashima y Sueyoshi, 2007]. Prevenir por completo los ataques de inundación SYN es imposible ya que un atacante puede ir variando diferentes parámetros en la solicitud SYN y el destinatario no sería capaz de diferenciarla de una solicitud legítima. Además, no se pueden tomar medidas que afecten a usuarios legítimos porque de esta forma solo se estaría ayudando a que el atacante consiga su objetivo.

En este trabajo de investigación se ha medido en un dispositivo C2 la efectividad de dos técnicas de mitigación definidas en el RFC 4987: SYN cookies y reemplazar conexiones en estado SYN-RECEIVED. Es preciso destacar que se trata del primer trabajo que implementa las SYN cookies en dispositivos limitados. Antes de su aplicación al dispositivo C2, se ha realizado un análisis de las implementaciones de SYN cookies que existen en los dos sistemas operativos que las utilizan actualmente. Se han detectado deficiencias en ambas implementaciones, en especial con el tiempo de validez de las cookies. Este tiempo de validez excesivo permite un nuevo ataque contra las propias SYN cookies.

Si la conexión se cierra en el servidor, la misma combinación de dirección de origen y puerto estará disponible nuevamente. La eliminación completa del estado cuando se utilizan SYN cookies hace imposible detectar repeticiones, y por lo tanto, un ACK con una cookie anterior se asumirá como un tercer paquete válido de una conexión TCP. En este escenario, las SYN cookies permiten a un cliente no esperar a un SYN-ACK la mayor parte del tiempo. Con la cola de conexiones tan limitada de los dispositivos C2, un cliente podría enviar intencionalmente el número requerido de paquetes para llenar la cola (por ejemplo, con una dirección IP falsa). A continuación, podría abrir una conexión, guardar la cookie recibida, y utilizarla para acelerar la apertura de sucesivas conexiones TCP durante 2 minutos

en el caso de Linux. Tras este análisis, se ha propuesto una implementación híbrida que solventa estas deficiencias.

El análisis estadístico (ANOVA) muestra que durante un ataque SYN de baja tasa, generar cookies –utilizando la implementación híbrida– tiene un menor efecto sobre el rendimiento de red de las conexiones activas que reemplazar conexiones en estado SYN-RECEIVED. Los resultados también confirman que a mayor tamaño de la cola de conexiones, mayor degradación del rendimiento. Buscar y descartar la entrada más antigua en estado SYN-RECEIVED, y después procesar y almacenar la nueva entrada afecta al rendimiento de red, y cuantas más posiciones tenga la cola de conexiones, mayor impacto. En definitiva, el trabajo de investigación demuestra que el diseño de cualquier estrategia de defensa para un dispositivo limitado debe ser cuidadosamente diseñado y verificado por el impacto no trivial en el rendimiento de red.

5.2 Líneas futuras

En esta sección se presentan algunas líneas de investigación futuras, tanto técnicas como aplicativas, que pueden ser objeto de interés atendiendo al trabajo expuesto en la presente tesis.

5.2.1 Técnicas

El principal inconveniente de los balanceadores de carga es que no tienen ninguna propiedad intrínseca de auto-configuración. Los administradores de sistemas deben configurar de manera manual el balanceador y los servidores para construir un clúster. Los dispositivos C2 de este trabajo de investigación también se han configurado de manera manual, p.ej. indicando las direcciones MAC de los servidores. En general, los usuarios tienen poco interés en realizar labores de configuración, simplemente quieren que las tecnologías o sistemas funcionen por sí mismos.

El IERC (European Research Cluster on the Internet of Things) define IoT como “una infraestructura de red global y dinámica con capacidad de auto-configuración basada en protocolos de comunicación estándar e interoperables donde los objetos físicos y virtuales tienen identidades, atributos físicos y personalidades virtuales, utilizan interfaces inteligentes, y están perfectamente integrados en la red de información”.

En base a esta definición, los dispositivos IoT –incluidos los de recursos limitados– deberían ser capaces de administrarse por sí solos, y por lo tanto deberían ser capaces de crear, configurar y organizar las funciones de cada dispositivo en un clúster (balanceador+servidores). Ante un aumento del tráfico, los dispositivos podrían identificar los dispositivos con el mismo servicio, intercambiar mensajes de administración para crear el clúster bajo demanda, y una vez que desaparezca ese tráfico volver a su estado inicial. Es decir, se podría aplicar el concepto de elasticidad que se utiliza en la computación en la nube [Truong y Dustdar, 2015], y cuyo objetivo no es otro que adaptar de manera autónoma los recursos disponibles para que se ajusten a la demanda actual.

Esta idea de auto-configuración se puede encontrar en las redes ad hoc. Las redes ad hoc son redes donde los nodos se agregan y comunican entre sí de forma inalámbrica. Por lo tanto, cualquier tipo de infraestructura central no es un requisito para la creación y el funcionamiento de estas redes. Los nodos gestionan las comunicaciones de forma autónoma, por lo que la única manera de desconectar una red ad hoc es apagar cada dispositivo de red. Esta es la característica principal que hace que las redes ad hoc sean más flexibles que las redes basadas en infraestructura. En sistemas con balanceadores se suele desplegar una arquitectura distribuida en la que existe más de una balanceador. Esta redundancia evita que una desconexión o fallo del balanceador aísle a los servidores. La característica de auto-configuración hace que los dispositivos de una red ad hoc puedan reconfigurarse automáticamente de acuerdo a la disponibilidad, a la proximidad o a parámetros como el ancho de banda. De esta forma, ante el fallo del balanceador, uno de los servidores del clúster podría tomar el rol de balanceador y permitir de esta manera que el clúster siga ofreciendo el servicio a los clientes. Otra opción pasaría por diseñar un balanceo sin un dispositivo dedicado, en el que los servidores configurasen reglas específicas de tráfico para repartirse la carga[1].

También se podría dotar de funciones de servidor al propio balanceador. En la actualidad, un balanceador es un dispositivo dedicado que solo se encarga de distribuir las conexiones. Sería interesante que el balanceador pudiera servir conexiones además de balancearlas, lo que permitiría tener un sistema más flexible. Habría que analizar si esta configuración afectaría al rendimiento de red de las conexiones balanceadas.

Por otra parte, se podrían repetir los experimentos realizados en un entorno inalámbrico y con componente de movilidad. En este tipo de red descentralizada cobrarían una mayor importancia si cabe las propiedades de seguridad y disponibilidad.

Por último, en lo referente al balanceo de carga, sería interesante realizar un ANOVA entre los dos experimentos explicados en el capítulo anterior, es decir, entre el experimento en el laboratorio 112 con 20 equipos clientes (fase II) y el experimento local con un único equipo cliente (fase I). En pruebas de rendimiento es una práctica común simular múltiples clientes TCP desde el mismo ordenador, es decir, todo el tráfico se genera desde la misma interfaz de red. En la fase I de la experimentación se concluye que el número de clientes es un factor muy significativo en los resultados de rendimiento del clúster de dispositivos C2, y por eso se decide realizar otro experimento en el que los clientes fueran reales y no simulados. El objetivo con este ANOVA sería determinar si la diferencia entre utilizar clientes reales o simulados es estadísticamente significativa, y en ese caso se demostraría que esa práctica común de simular los clientes no es apropiada en ciertas pruebas de rendimiento. Para poder realizar este análisis estadístico se tendrían que repetir los pasos de la fase I utilizando la configuración hardware de la fase II, es decir, como equipo cliente habría que usar uno de los equipos del laboratorio 112, y también el mismo balanceador de carga III. Por otra parte, a pesar de que los experimentos han reducido en la medida de lo posible las variables externas con el objetivo de generalizar los resultados estadísticos obtenidos, cada balanceador tiene un controlador de red diferente. Lo ideal sería que los controladores fueran los

[1] http://www.lartc.org/autoloadbalance.html

mismos, de esta forma se eliminaría esa variable y los resultados dependerían fundamentalmente de la velocidad y la capacidad de la CPU, y de la memoria disponible.

En cuanto a los ataques de inundación SYN en dispositivos C2, se podrían implementar más mecanismos de mitigación –presentados en el capítulo 2– y relacionar su coste en recursos y los beneficios obtenidos. De esta forma, se obtendría una tabla de referencia que ayudaría a determinar la mejor opción de seguridad ante ataques SYN dependiendo de los recursos específicos de cada dispositivo limitado.

5.2.2 Aplicativas

En cuanto a las líneas aplicativas, el ámbito de los laboratorios remotos resulta interesante. Estos laboratorios permiten la experimentación con dispositivos físicos de manera remota. La Universidad de Deusto es un referente internacional en el diseño, implementación y uso de estos laboratorios.

Después de analizar las arquitecturas adoptadas comúnmente en laboratorios remotos, se llega a la conclusión de que todos ellos se basan en un servidor potente encargado de llevar a cabo las tareas de interacción con el experimento y con el usuario. Este servidor se encarga de las tareas de administración y autenticación de usuarios, y además permite a los estudiantes interactuar con el experimento. Por lo tanto, el servidor está conectado a Internet para permitir el acceso a los estudiantes, y también está conectado con el experimento físico.

El sistema denominado WebLab-DEUSTO-PIC [García-Zubia et al., 2008] es capaz, por sí mismo, de realizar todas estas tareas, evitando la necesidad de configurar un servidor adicional entre el estudiante y el experimento. El estudiante se comunica directamente con el experimento y todas las tareas llevadas a cabo por el servidor se ejecutan en el microcontrolador incluido en la plataforma WebLab-DEUSTO-PIC.

En un artículo posterior [García-Zubia et al., 2010] se explican con más detalle los pros y contras de esta arquitectura. Entre los pros destacan el bajo coste del sistema y la facilidad de despliegue, y que al ser compatible con el protocolo DHCP la instalación del sistema en cualquier infraestructura es tan simple como conectar a través de un cable Ethernet el dispositivo. Por otro lado, las limitaciones impuestas por los microcontroladores determinan ciertos aspectos funcionales del laboratorio remoto. Entre estas limitaciones está el acceso concurrente, y que las tareas de administración y autenticación suponen una carga alta para el microcontrolador. Por este motivo, los investigadores de la Universidad de Deusto decidieron utilizar una arquitectura basada en servidores convencionales.

La idea sería desarrollar un laboratorio remoto de bajo coste diseñado para experimentos con microcontroladores, siguiendo el enfoque propuesto por WebLab-DEUSTO-PIC, pero reduciendo en lo posible las limitaciones antes expuestas. El balanceo de carga es una herramienta que resolvería la limitación de un solo usuario para esos laboratorios remotos. Desplegar clústeres de dispositivos C2 con los componentes necesarios para ejecutar el experimento permitiría dar servicio a usuarios concurrentes a un bajo coste (por el precio de un servidor convencional se pueden desplegar multitud de dispositivos C2). El balanceador se encargaría de las tareas de administración y de autenticación de los estudiantes (servicio web), además de distribuirlos entre los servidores disponibles del laboratorio (clúster).

Publicaciones

A continuación se presentan los artículos publicados derivados de este trabajo de investigación.

1. Echevarria, J. J., Ruiz-de Garibay, J., Vazquez, J. I., Alvarez, M., y Ayerbe, A. (2011). WebTag: smart tag with embedded web server. In *Proceedings of the 5th International Symposium of Ubiquitous Computing and Ambient Intelligence*, Riviera Maya, Mexico

2. Echevarria, J. J., Ruiz-de Garibay, J., Legarda, J., Alvarez, M., Ayerbe, A., y Vazquez, J. I. (2012). WebTag: Web Browsing into Sensor Tags over NFC. *Sensors (Q1)*, 12(7):8675–8690. DOI: 10.3390/s120708675

3. Echevarria, J. J., Legarda, J., Larrañaga, J., y Ruiz-de Garibay, J. (2016). lwAKE: A Lightweight Authenticated Key Exchange for Class 0 Devices. *International Journal of Distributed Sensor Networks (Q3)*. DOI: 10.1155/2016/6236494

4. Echevarria, J. J., Garaizar, P., y Legarda, J. (2017). An experimental study on the applicability of SYN cookies to networked constrained devices. *Software: Practice and Experience (Q3)*. DOI: 10.1002/spe.2510

Bibliografía

Al-Duwairi, B. y Manimaran, G. (2006). Intentional dropping: A novel scheme for syn flooding mitigation. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–5.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., y Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.

Arregoces, M. y Portolani, M. (2003). *Data Center Fundamentals*. Cisco Press.

Aumasson, J.-P. y Bernstein, D. (2012). Siphash: A fast short-input prf. In Galbraith, S. y Nandi, M., editors, *Progress in Cryptology - INDOCRYPT 2012*, volume 7668 of *Lecture Notes in Computer Science*, pages 489–508. Springer Berlin Heidelberg.

Bernstein, D. J. (1997). *SYN Cookies*. http://cr.yp.to/syncookies.html.

Botta, A., de Donato, W., Persico, V., y Pescapé, A. (2016). Integration of cloud computing and internet of things: A survey. *Future Generation Computer Systems*, 56:684 – 700.

Bourke, T. (2001). *Server Load Balancing*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Cardellini, V., Colajanni, M., y Yu, P. S. (1999). Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39.

Casado, M., Cao, P., Akella, A., y Provos, N. (2006). Flow-cookies: Using bandwidth amplification to defend against ddos flooding attacks. In *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pages 286–287.

Chang, C.-W., Lee, S., Lin, B., y Wang, J. (2010). The taming of the shrew: Mitigating low-rate tcp-targeted attack. *IEEE Trans. on Netw. and Serv. Manag.*, 7(1):1–13.

Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences (2nd Edition)*. Routledge, 2 edition.

Corder, G. W. y Foreman, D. I. (2009). *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. New Jersey: Wiley.

Crowcroft, J., Deegan, T., Kreibich, C., Mortier, R., y Weaver, N. (2007). Lazy susan: dumb waiting as proof of work.

Darst, C. y Ramanathan, S. (1999). Measurement and management of internet services. In *Integrated Network Management VI. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management. (Cat. No.99EX302)*, pages 125–140.

Dehmer, M., Emmert-Streib, F., Pickl, S., y Holzinger, A. (2016). *Big Data of Complex Networks*. Chapman & Hall/CRC.

Duquennoy, S., Grimaud, G., y Vandewalle, J. J. (2009). The web of things: Interconnecting devices with high usability and performance. In *2009 International Conference on Embedded Software and Systems*, pages 323–330.

Echevarria, J. J., Garaizar, P., y Legarda, J. (2017). An experimental study on the applicability of SYN cookies to networked constrained devices. *Software: Practice and Experience (Q3)*. DOI: 10.1002/spe.2510.

Echevarria, J. J., Legarda, J., Larrañaga, J., y Ruiz-de Garibay, J. (2016). lwAKE: A Lightweight Authenticated Key Exchange for Class 0 Devices. *International Journal of Distributed Sensor Networks (Q3)*. DOI: 10.1155/2016/6236494.

Echevarria, J. J., Ruiz-de Garibay, J., Legarda, J., Alvarez, M., Ayerbe, A., y Vazquez, J. I. (2012). WebTag: Web Browsing into Sensor Tags over NFC. *Sensors (Q1)*, 12(7):8675–8690. DOI: 10.3390/s120708675.

Echevarria, J. J., Ruiz-de Garibay, J., Vazquez, J. I., Alvarez, M., y Ayerbe, A. (2011). WebTag: smart tag with embedded web server. In *Proceedings of the 5th International Symposium of Ubiquitous Computing and Ambient Intelligence*, Riviera Maya, Mexico.

Faber, T., Touch, J., y Yue, W. (1999). The time-wait state in tcp and its effect on busy servers. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1573–1583 vol.3.

Feki, M. A., Kawsar, F., Boussard, M., y Trappeniers, L. (2013). The internet of things: The next technological revolution. *Computer*, 46(2):24–25.

Garber, L. (2000). Denial-of-service attacks rip the internet. *Computer*, 33(4):12–17.

García-Zubia, J., Angulo, I., Hernandez, U., Castro, M., Sancristobal, E., Orduña, P., Irurzun, J., y de Garibay, J. R. (2010). Easily integrable platform for the deployment of a remote laboratory for microcontrollers. In *IEEE EDUCON 2010 Conference*, pages 327–334.

García-Zubia, J., Angulo, I., Hernández, U., y Orduña, P. (2008). Plug&Play remote lab for microcontrollers: WebLab-DEUSTO-PIC. In *7th European Workshop on Microelectronics Education May*, pages 28–30, Budapest, Hungary.

Hassan, S., kamboh, A. A., y Azam, F. (2014). Analysis of cloud computing performance, scalability, availability, amp; security. In *2014 International Conference on Information Science Applications (ICISA)*, pages 1–5.

Herbst, N. R., Kounev, S., y Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA. USENIX.

Heuer, J., Hund, J., y Pfaff, O. (2015). Toward the web of things: Applying web technologies to the physical world. *Computer*, 48(5):34–42.

Hogg, R. y Tanis, E. (2006). *Probability and Statistical Inference*. Prentice Hall.

Hong, Y. S., No, J. H., y Kim, S. Y. (2006). Dns-based load balancing in distributed web-server systems. In *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pages 4 pp.–.

Jamali, S. y Shaker, V. (2014). Defense against {SYN} flooding attacks: A particle swarm optimization approach. *Computers & Electrical Engineering*.

Jiang, X., Yang, J., Jin, G., y Wei, W. (2013). Red-ft: A scalable random early detection scheme with flow trust against dos attacks. *Communications Letters, IEEE*, 17(5):1032–1035.

Kalwar, S. (2010). Introduction to reactive protocol. *IEEE Potentials*, 29(2):34–35.

Kavisankar, L. y Chellappan, C. (2011). A mitigation model for tcp syn flooding with ip spoofing. In *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 251–256.

Kim, T. H., Choi, Y. S., Kim, J., y Hong, S. J. (2008). Annulling syn flooding attacks with whitelist. In *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 371–376.

Koerner, M. y Kao, O. (2013). Optimizing openflow load-balancing with l2 direct server return. In *2013 Fourth International Conference on the Network of the Future (NoF)*, pages 1–5.

Koomey, J., Berard, S., Sanchez, M., y Wong, H. (2011). Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(3):46–54.

Korczynski, M., Janowski, L., y Duda, A. (2011). An accurate sampling scheme for detecting syn flooding attacks and portscans. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5.

Kum, S. W., hyun Kim, S., Jeon, Y. R., Moon, J., y Lim, T.-B. (2016). Design and implementation of iot planner: Device interaction service for iot. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 441–443.

Kuzmanovic, A. y Knightly, E. (2006). Low-rate tcp-targeted denial of service attacks and counter strategies. *Networking, IEEE/ACM Transactions on*, 14(4):683–696.

Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and anovas. *Frontiers in Psychology*, 4:863.

Lee, H. y Thing, V. (2004). Port hopping for resilient networks. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 5, pages 3291–3295 Vol. 5.

Lemon, J. (2002). Resisting syn flood dos attacks with a syn cache. In *Proceedings of the BSD Conference 2002 on BSD Conference*, BSDC'02, pages 10–10, Berkeley, CA, USA. USENIX Association.

Limoncelli, T. A. (2017). Are you load balancing wrong? *Commun. ACM*, 60(2):55–57.

Lix, L. M., Keselman, J. C., y Keselman, H. J. (1996). Consequences of assumption violations revisited: A quantitative review of alternatives to the one-way analysis of variance f test. *Review of Educational Research*, 66(4):579–619.

Luo, Y. B., Wang, B. S., Wang, X. F., Hu, X. F., Cai, G. L., y Sun, H. (2015). Rpah: Random port and address hopping for thwarting internal and external adversaries. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 263–270.

Mahimkar, A., Dange, J., Shmatikov, V., Vin, H., y Zhang, Y. (2007). Dfence: Transparent network-based denial of service mitigation. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, pages 24–24, Berkeley, CA, USA. USENIX Association.

Martinez, B., Montón, M., Vilajosana, I., y Prades, J. D. (2015). The power of models: Modeling power consumption for iot devices. *IEEE Sensors Journal*, 15(10):5777–5789.

Mathis, M., Semke, J., Mahdavi, J., y Ott, T. (1997). The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82.

McKusick, M. K., Neville-Neil, G., y Watson, R. N. (2014). *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley Professional, 2nd edition.

Milani, A. S. y Navimipour, N. J. (2016). Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends. *Journal of Network and Computer Applications*, 71:86 – 98.

Ming, Y. (2009). A probabilistic drop scheme for mitigating syn flooding attacks. In *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 1, pages 732–734.

Mirzaie, S., Elyato, A. K., y Sarram, M. A. (2010). Preventing of syn flood attack with iptables firewall. In *2010 Second International Conference on Communication Software and Networks*, pages 532–535.

Mohaghegh, N. y Aboelaze, M. (2012). Using low-power embedded microcontrollers as web servers. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 865–866.

Nakashima, T. y Sueyoshi, T. (2007). Performance estimation of tcp under syn flood attacks. In *Complex, Intelligent and Software Intensive Systems, 2007. CISIS 2007. First International Conference on*, pages 92–99.

Nashat, D., Jiang, X., y Horiguchi, S. (2008). Router based detection for low-rate agents of ddos attack. In *2008 International Conference on High Performance Switching and Routing*, pages 177–182.

Olanrewaju, R. F., Egal, A., y Khalifa, O. O. (2014). Bandwidth conservation framework for mobile cloud computing: Challenges and solutions. In *2014 International Conference on Computer and Communication Engineering*, pages 154–157.

Patel, P., Bansal, D., Yuan, L., Murthy, A., Greenberg, A., Maltz, D. A., Kern, R., Kumar, H., Zikos, M., Wu, H., Kim, C., y Karri, N. (2013). Ananta: Cloud scale load balancing. *SIGCOMM Comput. Commun. Rev.*, 43(4):207–218.

Preden, J.-S., Llinas, J., y Motus, L. (2016). *Middleware for Exchange and Validation of Context Data and Information*, pages 205–230. Springer International Publishing, Cham.

Preden, J. S., Tammemäe, K., Jantsch, A., Leier, M., Riid, A., y Calis, E. (2015). The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45.

Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., y Raghavan, B. (2011). Tcp fast open. In *Proceedings of the 7th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*.

Rainie, L. y Wellman, B. (2012). *The Internet Revolution*, pages 59–80. MIT Press.

Ravi, S., Raghunathan, A., Kocher, P., y Hattangady, S. (2004). Security in embedded systems: Design challenges. *ACM Trans. Embed. Comput. Syst.*, 3(3):461–491.

Safa, H., Chouman, M., Artail, H., y Karam, M. (2008). A collaborative defense mechanism against {SYN} flooding attacks in {IP} networks. *Journal of Network and Computer Applications*, 31(4):509 – 534.

Sample, A., Yeager, D., Powledge, P., Mamishev, A., y Smith, J. (2008). Design of an RFID-Based Battery-Free Programmable Sensing Platform. *Instrumentation and Measurement, IEEE Transactions on*, 57(11):2608–2615.

Sarkar, S., Chatterjee, S., y Misra, S. (2015). Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, PP(99):1–1.

Sessini, P. y Mahanti, A. (2006). Observations on Round-Trip Times of TCP Connections. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*.

Shi, W., Cao, J., Zhang, Q., Li, Y., y Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

Smith, C. y Matrawy, A. (2008). Comparison of operating system implementations of syn flood defenses (cookies). In *Communications, 2008 24th Biennial Symposium on*, pages 243–246.

Stevens, W. R. (1993). *TCP/IP Illustrated (Vol. 1): The Protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Stevens, W. R., Fenner, B., y Rudoff, A. M. (2003). *UNIX Network Programming, Vol. 1*. Pearson Education, 3 edition.

Stojmenovic, I. y Wen, S. (2014). The fog computing paradigm: Scenarios and security issues. In *2014 Federated Conference on Computer Science and Information Systems*, pages 1–8.

Sun, C., Hu, C., y Liu, B. (2012). Sack2: effective syn flood detection against skillful spoofs. *IET Information Security*, 6(3):149–156.

Thangavel, D., Ma, X., Valera, A., Tan, H. X., y Tan, C. K. Y. (2014). Performance evaluation of mqtt and coap via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6.

Thiruvathukal, G. K. (2005). Guest editorsíntroduction: Cluster computing. *Computing in Science Engineering*, 7(2):11–13.

Truong, H. L. y Dustdar, S. (2015). Programming elasticity in the cloud. *Computer*, 48(3):87–90.

Weiden, F. y Frost, P. (2010). Anycast as a load balancing feature. In *Proceedings of the 24th International Conference on Large Installation System Administration*, LISA'10, pages 1–6, Berkeley, CA, USA. USENIX Association.

Wu, G., Sun, J., y Chen, J. (2016). A survey on the security of cyber-physical systems. *Control Theory and Technology*, 14(1):2–10.

Zhang, B., Mor, N., Kolb, J., Chan, D. S., Lutz, K., Allman, E., Wawrzynek, J., Lee, E., y Kubiatowicz, J. (2015). The cloud is not enough: Saving iot from the cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA. USENIX Association.

Zuquete, A. (2002). Improving the functionality of syn cookies. In *Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security: Advanced Communications and Multimedia Security*, pages 57–77, Deventer, The Netherlands, The Netherlands. Kluwer, B.V.