ITBA
Instituto Tecnológico
de Buenos Aires

Proyecto Final
Ingeniería Informática

Detección de Patrones de Comportamiento en Tráfico de Red

Alumnos:

- Bond, Federico
- Itzcovich, Iván

Tutor:

- Vallés, Santiago

Diciembre 2016

Introducción

Este informe es el resultado de un proyecto de investigación y desarrollo que busca estudiar de qué manera se puede automatizar la interpretación y el procesamiento de la información disponible en el tráfico de red con el objetivo de obtener una descripción objetiva y lo más exacta posible del escenario sobre el cual la red se manifiesta.

El informe está organizado de la siguiente manera:

Sección I: **Problema**

Presenta el problema que se pretende resolver. Establece los objetivos, dificultades y desafíos del proyecto. Incluye un Marco Teórico con un breve resumen de aquellos temas cuyo conocimiento es necesario para la comprensión del informe.

Sección II: **Metodología**

Explica la metodología que se utilizó en este proyecto. Se mencionan y detallan en profundidad las etapas por las que pasó este trabajo: la etapa de investigación, la etapa de desarrollo y la etapa de prueba. No se habla del contenido de cada etapa, sino de su metodología y actividades.

Sección III: **Resultados de la etapa de investigación**

Presenta todos los resultados de la etapa de investigación. Incluye observaciones acerca de comportamientos estándar. Explica estrategias para detección de dispositivos y estrategias para detección de servicios. Define brevemente el alcance, las limitaciones y las estrategias para el sistema que se pretende desarrollar.

Sección IV: **Implementación**

Explica los resultados de la etapa de desarrollo. Introduce SonarWAN como sistema desarrollado. Menciona y justifica la tecnología utilizada, especifica el *input/output* del sistema. Desarrolla la arquitectura tanto física como de *software*. Explica las entidades y los componentes del diseño; como así también los algoritmos implementados. Resume el flujo de información del sistema y caracteriza la base de conocimiento en la que se basa el sistema.

Sección V: **Pruebas**

Presenta y analiza los resultados de la etapa de pruebas. Figuran tablas asociativas por cada prueba realizada. Explica las conclusiones obtenidas a partir de los resultados obtenidos.

Sección VI: **Conclusiones**

Presenta las conclusiones del proyecto. Trata las ventajas/desventajas del sistema desarrollado, como así también las expectativas. Reflexiona acerca del estado actual de NAT en relación a IPv6 y su impacto en este proyecto.

Anexo

Aparecen ejemplos de uso, capturas de pantalla del sistema desarrollado. Incluye el manual de uso de SonarWAN

Bibliografía

Abstract del proyecto

El análisis forense de tráfico de red, tanto del tipo cifrado como el plano pueden revelar patrones de comportamiento en el uso de servicios de Internet. El lugar de captura de flujo de información es clave para la resolución del problema, pero no siempre es posible obtener el tráfico desde una red interna, sino desde un punto externo al que se desea analizar.

Los objetivos del proyecto son: estudiar los tipos de conexiones que generan una comunidad de personas a través de un único enlace de conexión a Internet; proponer métodos de parametrización de patrones de comportamiento; implementar métodos de detección de tráfico y de individualización de terminales y personas físicas que utilizan dichos servicios.

El proyecto debe estar orientado a resolución de problemas con dimensiones de casos reales, ya que la cantidad de información a ser analizada es un factor clave del proyecto.

Índice

Sección I: Problema 8

Marco Teórico 9
- Red de computadoras 9
- Protocolos de comunicación 10
- Internet 14
  - Protocolo IP 15
- Redes Privadas y NAT 16
- Redes LAN y WAN 19
- DNS y consumo de servicios por nombres 20
- Palabras finales 21

Qué se pretende resolver 22
- Detección de patrones de comportamiento 22
- Obtención de tráfico 22
- Objetivo del trabajo 23
  - Filosofía de “Código Abierto” 25

Aplicaciones en la realidad 26
  - ¿Por qué el lugar de captura? 26

Caso real 27

Sección II: Metodología 28

Introducción 29

Investigación del estado del arte 31
- Área de la Investigación 31
- Área del Desarrollo 31

Recolección y análisis de capturas 35
- Proceso de recolección de capturas 35
- Proceso de análisis de capturas generadas 37
  - Ejemplos de observaciones en capturas 37
    - Conexión a la red de un dispositivo Android 38
    - Consumo de servicios HTTP por parte de una aplicación en un dispositivo iOS 38

Desarrollo del sistema 40
  - Core 40
    1) Elección y familiarización con las tecnologías a utilizar 40
    2) Definición de arquitectura 40
    3) Programación de funcionalidad incremental 40

GUI 40
Etapa experimental en escenarios reales 42
1) Pruebas con capturas genéricas propias 42
2) Pruebas con capturas desconocidas facilitadas por el tutor 42
3) Pruebas de stress con capturas extensas 42

Sección III: Resultados etapa de investigación **43**

Introducción **44**

Comportamiento de NAT **45**
[RFC 2663 - Sección 4.1] Cambios en los paquetes 45
[RFC 2663 - Sección 7.0] Payload de los paquetes 45
[RFC 2663 - Sección 4.1] Tipos de NAT Tradicional 45
[RFC 3022 - Sección 4.1] Modificaciones 46
[RFC 5382 - Sección 4.1] Reutilización de conexiones TCP para un mismo host 46
[RFC 5382 - Sección 7.1] Asignación y reutilización de puertos para sesiones TCP 46
[RFC 4787 - Sección 4.2.1] Asignación de puertos en colisión 47
Acerca de otras características 47

Estrategias para detectar dispositivos y aplicaciones **48**
Campo ID del datagrama IP 48
User Agent Fingerprinting por HTTP plano 51
Acerca de los User Agents 51
Breve referencia a HTTP 51
Realizar “device fingerprinting” con el header “User-Agent” 52
Utilidad del User-Agent para el sistema 53
Distinción de dispositivos por Handshake TLS/SSL 55
Acerca del TLS/SSL y el handshake 55
Acerca del ClientHello 56
Utilidad del ClientHello para la distinción de dispositivos y/o aplicaciones 56
Otras estrategias HTTP 58
Header “Referer” 58
Cookies 58

Estrategias para detectar servicios consumidos **59**
Detección por IP 59
Detección por dominio 60
DNS Reverso 60
Caché de búsquedas DNS 61
Header Host de los request HTTP 61
¿Dominio implica nombre de servicio? 62
Detección por propietario 62

Breve referencia al alcance, estrategias y limitaciones del Sistema **64**

Sección IV: Implementación **66**

Introducción **67**
SonarWAN 67
Repositorios 67

Tecnología Utilizada **68**
Python 68
PyShark 69

Input / Output **71**
Input 71
Acerca de PCAP, “.pcap” y “.pcapng” 72
Acerca de los parámetros que recibe 72
Output 72

Arquitectura Física **74**

Arquitectura del Software **75**
Las entidades 75
Dispositivos 75
Información Intrínseca 75
Referencias a otras entidades 76
Diagrama 76
Aplicaciones 76
Información Intrínseca 76
Referencias a otras entidades 76
Diagrama 77
Servicios 77
Información Intrínseca 77
Referencias a otras entidades 77
Diagrama 78
Los componentes 78
Environment 78
Handlers 79
Tools 79
Comunicación entre componentes 80
El stream como unidad de análisis 81

Flujo de información y algoritmos implementados **85**
Paquetes DNS 85
Paquetes TCP y UDP 85
Paquetes TLS 86
Paquetes HTTP 86
Algoritmo de similitud 87

Algoritmos de detección de un servicio 89

Base de conocimiento 90

Breve mención de la performance 92

Especificación del output 92

Breve mención a la GUI 93

Nivel de satisfacción del sistema desarrollado 93

Sección V: Resultados 94

Introducción 95
Ajustes durante la etapa de prueba con capturas propias 95
Sistema de puntuación 96

Resumen de los resultados de las pruebas 97
Pruebas con capturas propias 97
Pruebas con capturas desconocidas facilitadas por el tutor 98
Pruebas de stress con capturas extensas 98

Conclusiones de las pruebas 100
Acerca del tráfico automático 101

Sección VI: Conclusiones 103

Palabras iniciales 104

IPv6 y NAT 104

Conclusiones acerca de SonarWAN 105
Ventajas y desventajas del sistema desarrollado 105
Futuras extensiones del sistema 105
Expectativas cumplidas 106
Expectativas futuras 106

Anexo 107

Manual de SonarWAN Core 108
Requisitos 108
Instalación 108
Uso 108
Ejemplos de corridas 109
Ejemplos de outputs en formato texto 110

Manual de SonarWAN GUI 112
Requisitos 112
Instalación 112
Uso 112

Resultados Pruebas de capturas conocidas - 1 dispositivo **116**

Resultados Pruebas de capturas conocidas - 2 dispositivos **117**

Resultados Pruebas de capturas conocidas - 3 o más dispositivos **118**

Bibliografía **119**

Papers consultados: 120

Recursos Web para la Sección I: 120

Recursos Web para la Sección III: 120

Recursos Web para la Sección IV: 120

Sección I: **Problema**

Marco Teórico

A continuación se desarrolla una breve explicación teórica de aquellos temas cuyo conocimiento es necesario para la comprensión del informe.

Los temas que se explican a continuación son:

- Red de computadoras y el proceso de comunicación.
- Protocolos de comunicación
- Internet y el protocolo IP en detalle
- Redes privadas y NAT
- Redes LAN y WAN
- DNS

Red de computadoras

Una red de información es, en esencia, un grafo; es decir, un conjunto de nodos y un conjuntos de conexiones entre nodos.

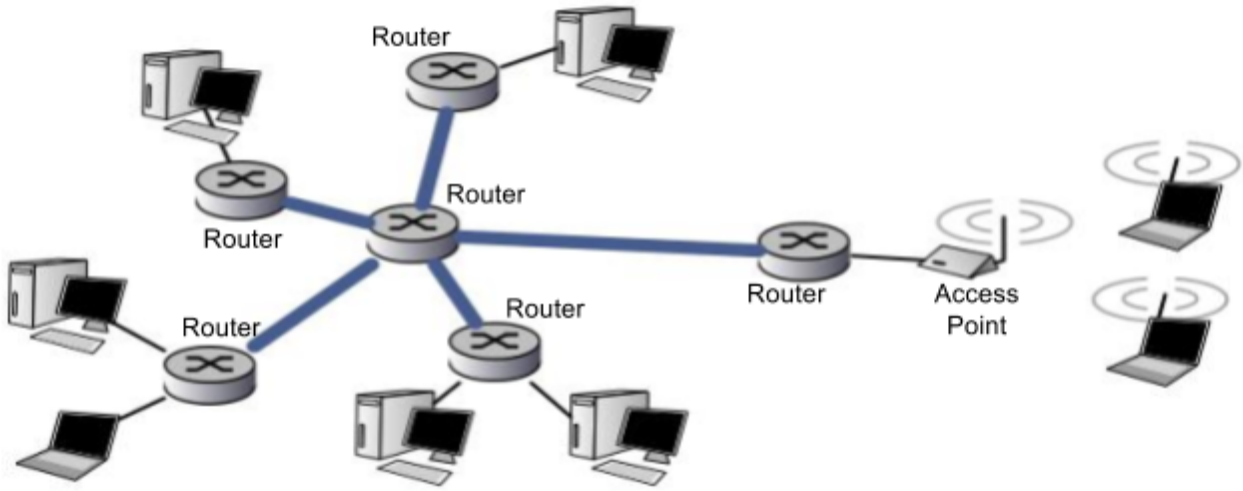


Fig 1: Estructura de una red de computadoras básica.

En la Fig 1 se puede observar un esquema de red. Los nodos, en este caso, son las computadoras de escritorio, *laptops*, *routers y access points.* Las conexiones son los cables que aparecen como líneas en el esquema, como así también existen conexiones inalámbricas entre algunas *laptops* y el *access point.*

El objetivo de una red de computadoras, también llamada red de información o de comunicaciones, es la **transmisión de datos** entre unidades de procesamiento para compartir información, recursos y ofrecer servicios.

Al igual que en cualquier proceso de comunicacion, la comunicacion en una red de computadoras posee los siguiente componentes básicos:

- **Emisor/Receptor**. Son unidades de procesamiento, también llamadas terminales o *hosts*. Por ejemplo, computadoras personales, servidores o *routers*. El emisor es el que origina la comunicación con el objetivo de enviar un mensaje. El receptor recibe el mensaje y lo interpreta.

- **Medio**. El medio puede ser cualquier entorno que soporte ondas electromagnéticas o impulsos eléctricos para la transmisión del mensaje. Por ejemplo, cables coaxil, fibras ópticas o el aire (redes WiFi).

- **Mensaje**. En el caso de la red de computadoras, el mensaje es un conjuntos de *bits*, es decir, 0's y 1's. Dependiendo del mensaje en cuestión, este conjunto de *bits* va a tener una estructura y un significado determinado que va a ser interpretado por el receptor.

Protocolos de comunicación

Los protocolos de comunicación son necesarios para estandarizar la comunicación entre los equipos. Un protocolo de comunicaciones es un sistema de reglas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellas para transmitir información. Se trata de reglas que definen la **sintaxis**, **semántica** y **sincronización** de la comunicación, así como también los posibles métodos de **recuperación de errores**.

Los protocolos se dividen o categorizan en capas. Estas capas conforman una estructura jerárquica, en la cual cada capa influye en el proceso de comunicación. Los protocolos de las distintas capas trabajan en conjunto para transmitir la información entre los nodos del sistema. Cada capa tiene una funcionalidad determinada que difiere de otras capas.

Existen diversos modelos que especifican cada una de las capas de protocolos. Los modelos más conocidos son **TCP/IP** y el modelo **OSI**.

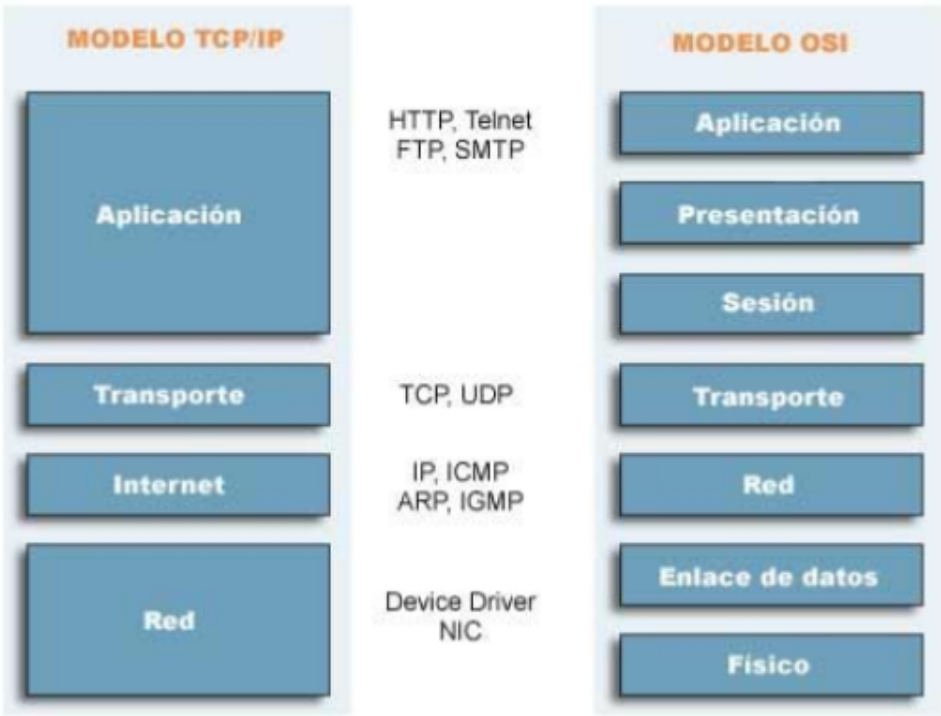


Fig 2: Modelos TCP/IP y OSI.

En la Fig 2 se pueden apreciar distintos modelos estandarizados de capas de protocolos. Para este trabajo, se va a tomar un modelo que toma ciertos componentes del **TCP/IP** y del **OSI.** A continuación se presenta el modelo de capas que se va a usar como referencia para este trabajo.

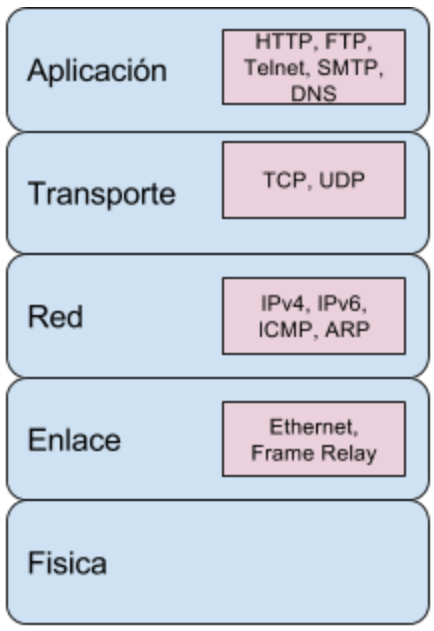


Fig 3: Modelo de capas con ejemplos de protocolos.

El objetivo de esta categorización es la independencia entre cada capa en la elección del protocolo de la capa en cuestión.

La división de protocolos en capas es simplemente una cuestión de organización conceptual y existen ciertos problemas con este carácter conceptual, en el sentido de que existen ciertos protocolos que no son fácilmente identificables con una capa en particular. Por ejemplo, **TLS** (o **SSL**) es una capa superior a la de transporte, pues utiliza TCP como protocolo de transporte pero no llega a estar a nivel de aplicación, pues por encima de TLS puede viajar, por ejemplo HTTP (*suit*e conocida como HTTPS).

Cada capa del modelo se “responsabiliza” de un componente distinto en la comunicación. A continuación se presenta un esquema de esta idea.

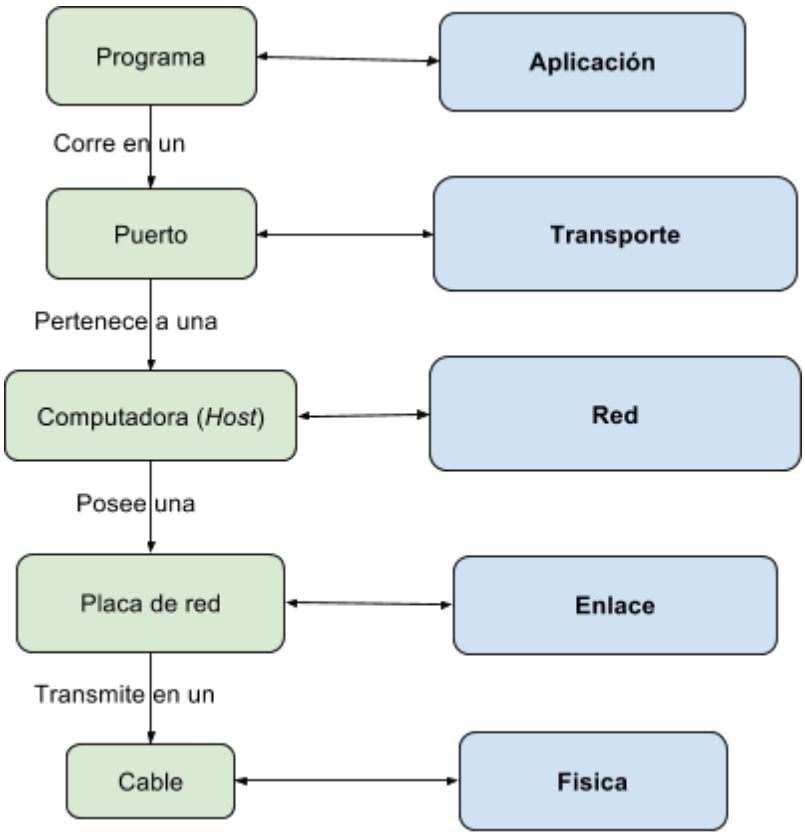


Fig 4: Responsabilidad de cada capa

Siguiendo el concepto de la Fig 4, se puede concluir lo siguiente:

- **La capa de aplicación** se encarga de la comunicación entre programas.
- **La capa de transporte** se encarga de la comunicación entre puertos
- **La capa de red** se encarga de la comunicación entre *hosts*
- **La capa de enlace** se encarga de la comunicación entre placas de red.
- **La capa física** se encarga de la conexión física entre placas (cables, aire, etc.)

En una comunicación entre dos programas (o mejor dicho, procesos) que correr en computadoras distintas de una red, se van a recorrer todas las capas del modelo, es decir,

van a entrar en juego protocolos de todas las capas. En el emisor, cada protocolo va a ir agregando determinada información al mensaje, información que va a ser consumida por el mismo protocolo en el receptor. El flujo de datos en la comunicación tiene forma de U, y puede representarse con el siguiente diagrama para un mensaje "**M"**.

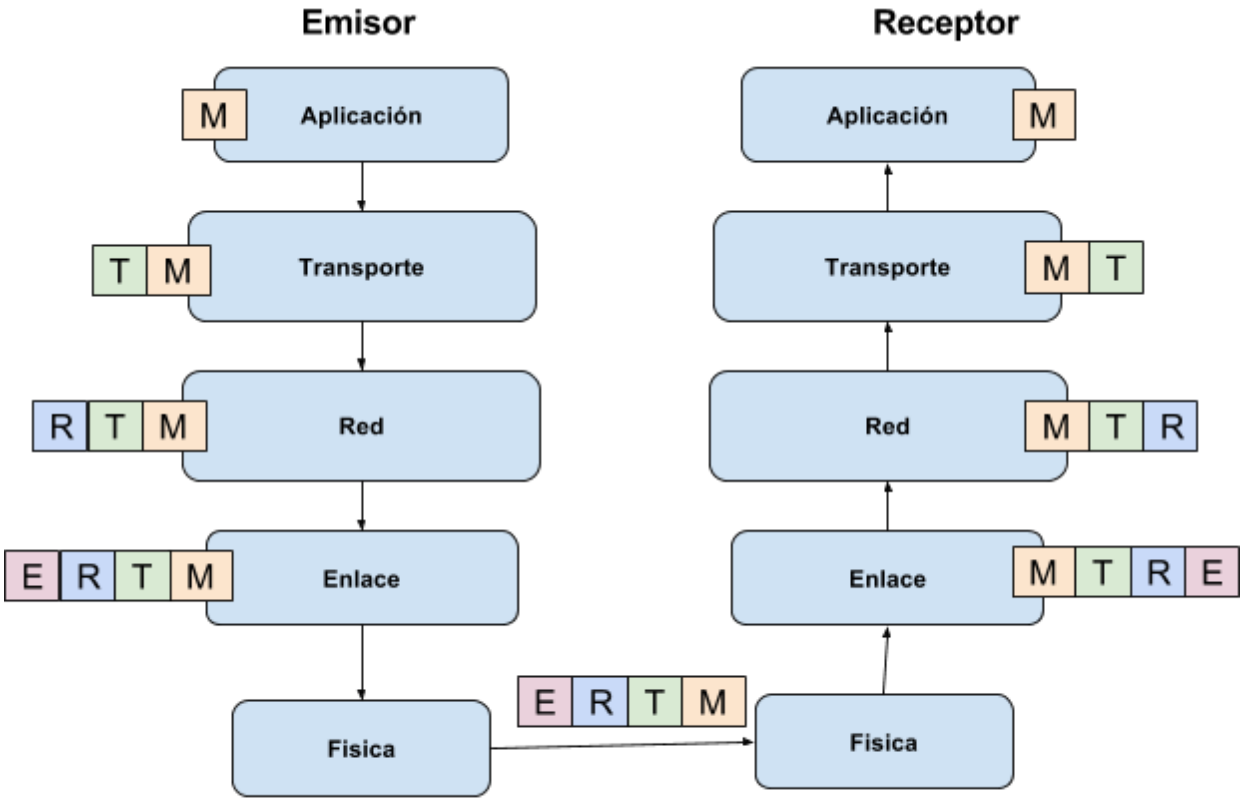


*Fig 5: Flujo de datos en una comunicacion. **M** es el mensaje original, **T, R, E** es la información se agrega cada capa, Transporte, Red, y Enlace respectivamente*

En el emisor, **cada protocolo toma la información que viene del protocolo de la capa superior, le agrega información particular encapsulando la información anterior y se lo pasa al protocolo de la capa inferior.**

En el receptor, **cada protocolo toma la información que viene del protocolo de la capa inferior, consume e interpreta la información que le corresponde, desencapsula la información contenida y se la pasa al protocolo de la capa superior.**

Es importante mencionar que no todos los protocolos tienen la funcionalidad de encapsular información que contiene un mensaje de un protocolo de Aplicación. Existen protocolos que son auto-contenidos y que no son de Aplicación. Estos protocolos representan una comunicación que no es entre dos programas necesariamente. Por ejemplo, ICMP es un protocolo de Red que no encapsula información de Transporte. Este protocolo comunica a dos computadoras (sin llegar a puertos ni programas, ver Fig 4). Es decir, la comunicación puede darse, además de Programa-Programa, entre Puerto-Puerto o *Host-Host* (ICMP, ARP, RIP, OSPF, etc).

Como se dijo antes, cada capa es independiente de la otra. Es decir, cada capa solo puede interpretar la información que le corresponde a esa capa y la otra información le es indiferente. Por ejemplo, la información que maneja TCP (cuya unidad es denominada **paquete**) como protocolo de Transporte puede contener información a nivel Aplicación del tipo HTTP, SMTP, FTP, etc; y su comportamiento es el mismo.

Este principio es el que hace tan potente a dicha arquitectura de capas, y es la que potencia a Internet.

***Nota**: A la unidad de información se lo va a denominar Paquete (en el caso de protocolos de Transporte), Datagrama (en el caso de protocolos de Red) o Frame (en el caso de protocolos de Enlace).*

Internet

Internet, también denominada “Red de redes”, es al fin y al cabo una red de computadoras. Se caracteriza por su alta cantidad de *hosts* que la componen, como así también la cantidad de servicios, recursos y contenido que provee. Estar “conectado” a Internet significa tener acceso a los *hosts* que la componen para consumir su información.

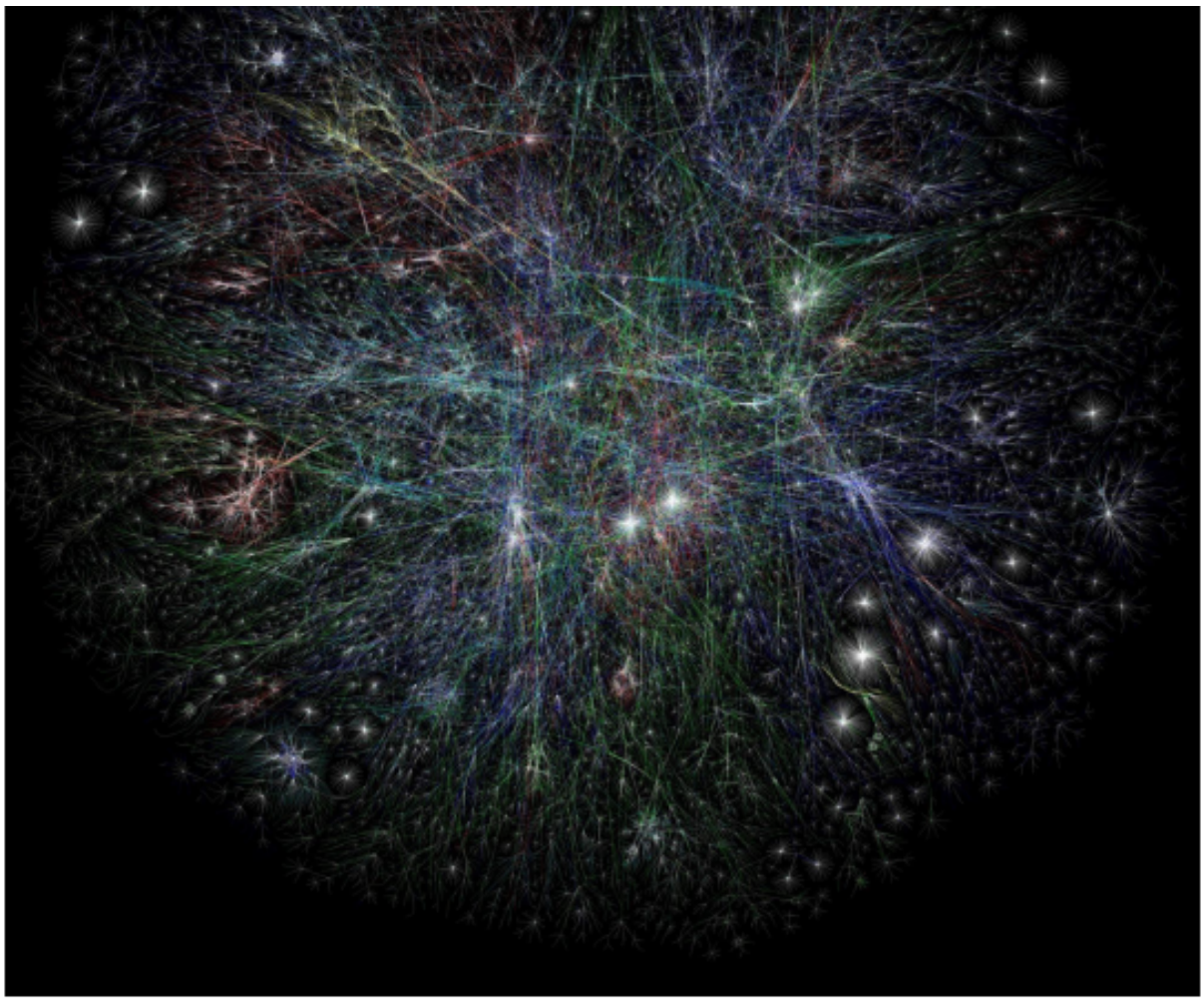


Fig 6. Representacion artistica de las conexiones en Internet, realizado por Opte Project.

Los protocolos que se manejan en Internet son muy amplios en lo que respecta a la capa de Transporte, Aplicación, Enlace y Físico. Sin embargo, todos los *hosts* que componen a esta red se comunican con el protocolo IP a nivel de capa de Red, que se detallara a continuación.

Protocolo IP

El protocolo IP es la base de Internet (cuyas iniciales por cierto significan **I**nternet **P**rotocol) y como se explicó anteriormente, al ser un protocolo de capa de Red, es responsable de la comunicación entre *hosts*. Este protocolo identifica con un número (denominado número o dirección IP) único a cada *host* que pertenece a la red. Es necesario que las direcciones sean únicas para un correcto funcionamiento del protocolo. Adicionalmente, un *host* puede pertenecer a distintas redes, adquiriendo de esta manera distintos identificadores únicos, uno por cada red de la que forma parte.

La versión actual del protocolo, IP versión 4 (IPv4), establece que la dirección IP es una secuencia de 32 bits, o lo que es equivalente, 4 bytes. La notación de una dirección IPv4 es la secuencia de bytes en valor decimal (de 0 a 255) separados por un punto (‘.’). Por ejemplo, **10.6.9.8**, **8.8.8.8**, **255.255.255.255**, **127.0.0.1**, etc.

Como se dijo antes, los *hosts* de la red deben tener una dirección IP única e irrepetible, es decir, no puede ocurrir que el *host* A y el *host* B contengan la misma dirección IP dentro de la misma red.

Dicho esto, se puede deducir fácilmente que la cantidad de direcciones IP disponibles en una red es de:

256^4**= 4,294,967,296**

A esta cantidad de direcciones disponibles, hay que restarle ciertos conjuntos debido a distintas razones:

1. Direcciones reservadas para la interfaz *loopback* (interfaz virtual del *hosts* únicamente visible por el). Son todas cuyo primer byte tiene el valor de 127 (o también, segun su notacion tecnica, **127.0.0.0/8**). Es decir, abarca 256^3 direcciones IP.
2. Direcciones reservadas para redes privadas (va a ser explicado luego)
3. Direcciones de identificación (todas cuyo último byte es 0)
4. Direcciones de broadcast
5. Rangos de gran cantidad de direcciones (de hasta 256^3 direcciones) asignadas a entidades privadas o universidades cuando Internet era un proyecto de laboratorio. Estas direcciones no están utilizadas y representan un desperdicio de posibles *hosts*.

Teniendo en cuenta todos estos factores y considerando el carácter masivo y global que tomo Internet en los últimos 15 años, es evidente concluir que la cantidad de

direcciones IP disponibles para Internet (o lo que equivale, la cantidad de *hosts* que pueden formar parte de Internet) son muy bajas.

Frente a este problema, se plantean dos soluciones:

- **Migración a IPv6, la nueva versión del protocolo IP**. En lugar de 32 bits, cada *host* se identifica con 128 bits. Es decir, hay aproximadamente 3.4E38 *hosts* disponibles. Esta solución se está empezando a implementar pero va a pasar mucho tiempo hasta que la migración de IPv4 a IPv6 finalice completamente. La corriente de tecnología IoT (*Internet of Things)*, por ejemplo, necesita de esta tecnología para su correcto funcionamiento.

- **Uso de redes privadas.** Es la solución que se utiliza actualmente. Conceptualmente consiste en armar redes que contengan varios *hosts* en una red de computadoras y que accedan a Internet a través de un conjunto de *hosts* distintos cuyo cardinal sea menor que el de los *hosts* que se encuentran en la red. Esta propuesta se va a estudiar más en detalle a continuación.

Redes Privadas y NAT

Como se mencionó anteriormente, la idea de las redes privadas, en conjunto con la técnica de NAT, surge principalmente como solución a la baja disponibilidad de direcciones IP que existe en Internet.

Una red privada es una red de computadoras que tienen asignado un rango de direcciones IP estandarizado por el RFC (Request For Comments) 1918. Estas direcciones son:

- 10.0.0.0 - 10.255.255.255 (10/8)
- 172.16.0.0 - 172.31.255.255 (172.16/12)
- 192.168.0.0 - 192.168.255.255 (192.168/16)

Estas direcciones no pertenecen a ningún *host* (nodo) de Internet. Estas direcciones IP suelen denominarse direcciones IP **privadas**, mientras que las restantes se denominan direcciones IP **públicas** al poder ser adquiridas por cualquier *host* de Internet.

Ahora bien, las redes privadas necesitan, en muchos casos, acceso a Internet. Como se explica anteriormente, el acceso se da a través de un número menor de *hosts* que pertenecen tanto a Internet (poseen una dirección IP pública) como a la red privada (poseen también dirección IP privada). En casi todos los casos, este *host* que pertenece a ambas redes es uno solo. Es decir, todo el tráfico originado desde la red privada que se dirige a Internet pasa por este *host*, que se denomina *gateway*.

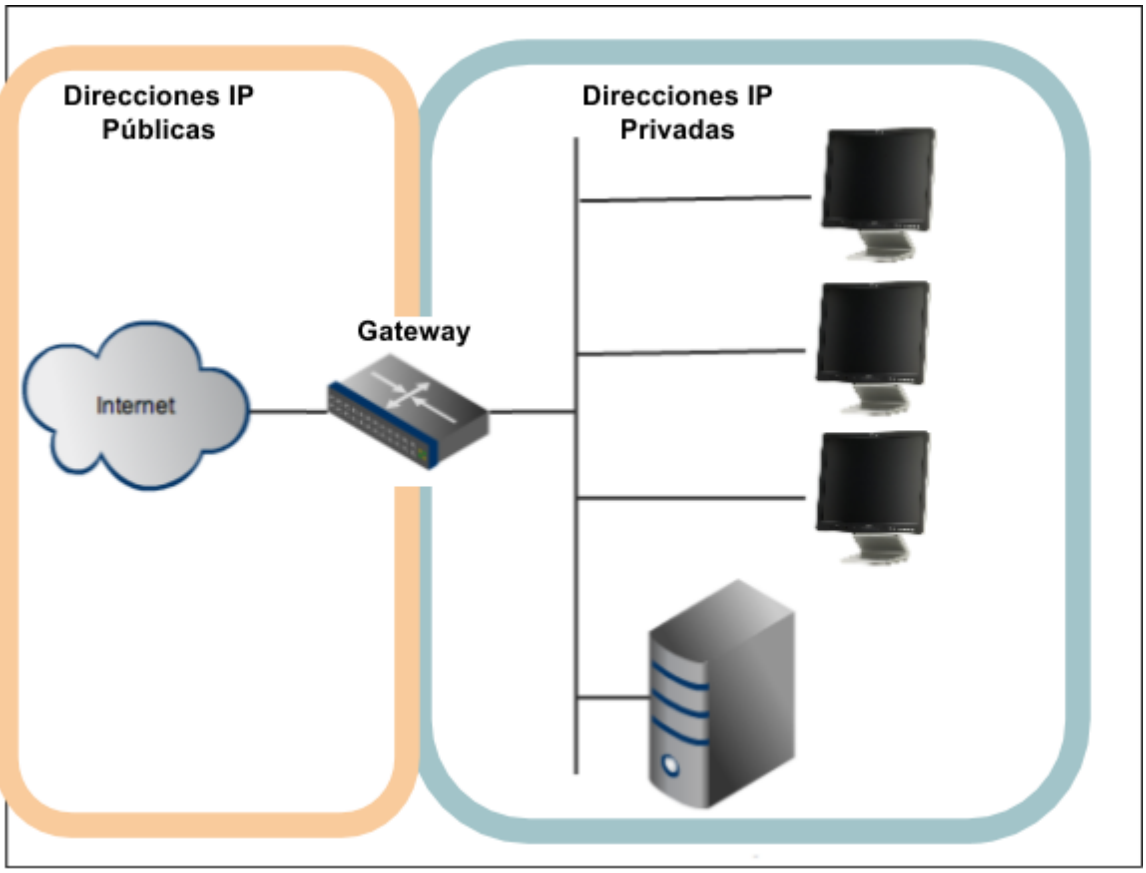


Fig 7: Estructura básica de una red privada.

Existe una incompatibilidad en la comunicación entre la red privada e Internet a nivel del protocolo IP, pues los *hosts* de Internet no pueden identificar a los *hosts* de la red privada, al no formar parte (estos últimos) de la red de Internet. Es decir, las direcciones IP de los *hosts* privados no corresponden a ninguna dirección de un *host* perteneciente a Internet.

Frente a este problema, surge la **traducción de direcciones de red**, también conocido como **NAT** (**N**etwork **A**ddress **T**ranslation). Esta técnica es un mecanismo utilizado por intercambiar información entre dos redes que tienen direcciones IP incompatibles. Se basa en la traducción en tiempo real de las direcciones utilizadas en los datagramas transportados. Las técnicas de NAT están estandarizadas en el RFC 2663. Funciona de la siguiente manera:

- Cuando un paquete sale de la red privada hacia Internet, el *gateway* reemplaza en el datagrama la dirección IP origen por la dirección IP pública que posee. De esta manera, para los *hosts* de Internet, el emisor del datagrama fue el *gateway*, que al tener IP publica, pertenece a la red de Internet. Este proceso se conoce como Source NAT, o **SNAT**.
- Al realizar **SNAT**, el *gateway* guarda en una tabla a que *host* de la red privada le realizó la traducción.
- Cuando vuelve una respuesta de un *host* de Internet, va a recibirlo el *gateway*, pues fue el “emisor” original del mensaje para los *hosts* de Internet.

El *gateway* va a consultar la tabla de traducciones y va a cambiar la dirección IP destino del datagrama por la dirección de IP privada del *host* original de la red privada, para que este sea verdaderamente el receptor del mensaje. Este proceso se conoce como Destination NAT, o **DNAT.**

El proceso de **NAT** es transparente para los *hosts* de la red privada, en el sentido que se pueden comunicar libremente con *hosts* de Internet como si fueran parte de esa red. Paralelamente, para los *hosts* de Internet, la comunicación se da únicamente con el *gateway.* Ningún *host* de Internet puede saber si está intercambiando información con un *host* unitario, o si está comunicándose con un *gateway* que posee miles de dispositivos detrás en redes privadas interconectadas. La composición de Internet con redes privadas y su posible comunicación a pesar de incompatibilidades en las direcciones IP hacen que sea denominada la "Red de redes".

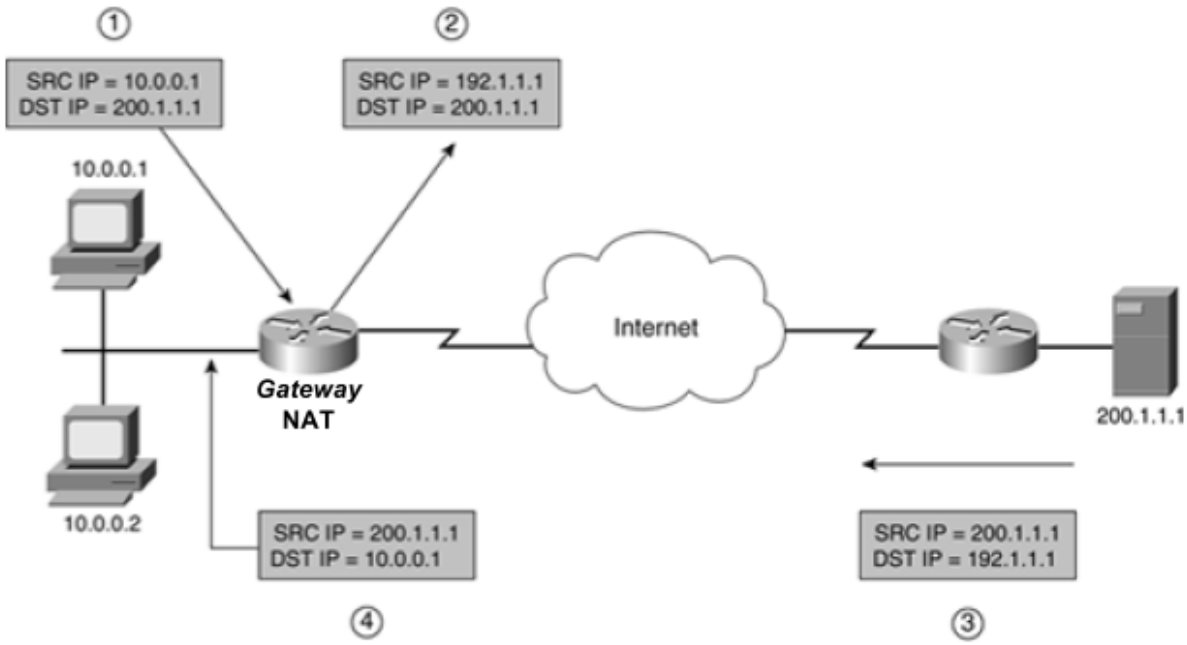


Fig 8: Flujo ejemplo de NAT.

Siguiendo este esquema de NAT, pueden existir infinitas redes privadas que no tengan conexión entre sí y que usen las mismas direcciones IP, todas con conexión a Internet. Esta ventaja soluciona la disponibilidad de direcciones IPv4 en la actualidad.

Otra ventaja notable es el tema de la seguridad. Los *hosts* de Internet desconocen por completo a los *hosts* que se encuentran en redes privadas, y lo que es aún más, al tener un *gateway* que es el único punto de entrada a Internet, se pueden definir políticas de seguridad sobre la información que puede salir/entrar a la red privada. Estas políticas de seguridad entran en lo que se denomina **Firewall.**

Todas las redes hogareñas que tienen acceso a Internet funcionan de esta manera. La red que se genera es una red privada, en la cual cada *host* (celulares, computadoras, *SmartTV*) posee una dirección IP privada. El *gateway* suele ser el *router*, que además de poseer dirección IP privada, posee una dirección IP pública (asignada por el proveedor del servicio, también conocido como **ISP**). El *router* es el encargado de realizar **NAT**.

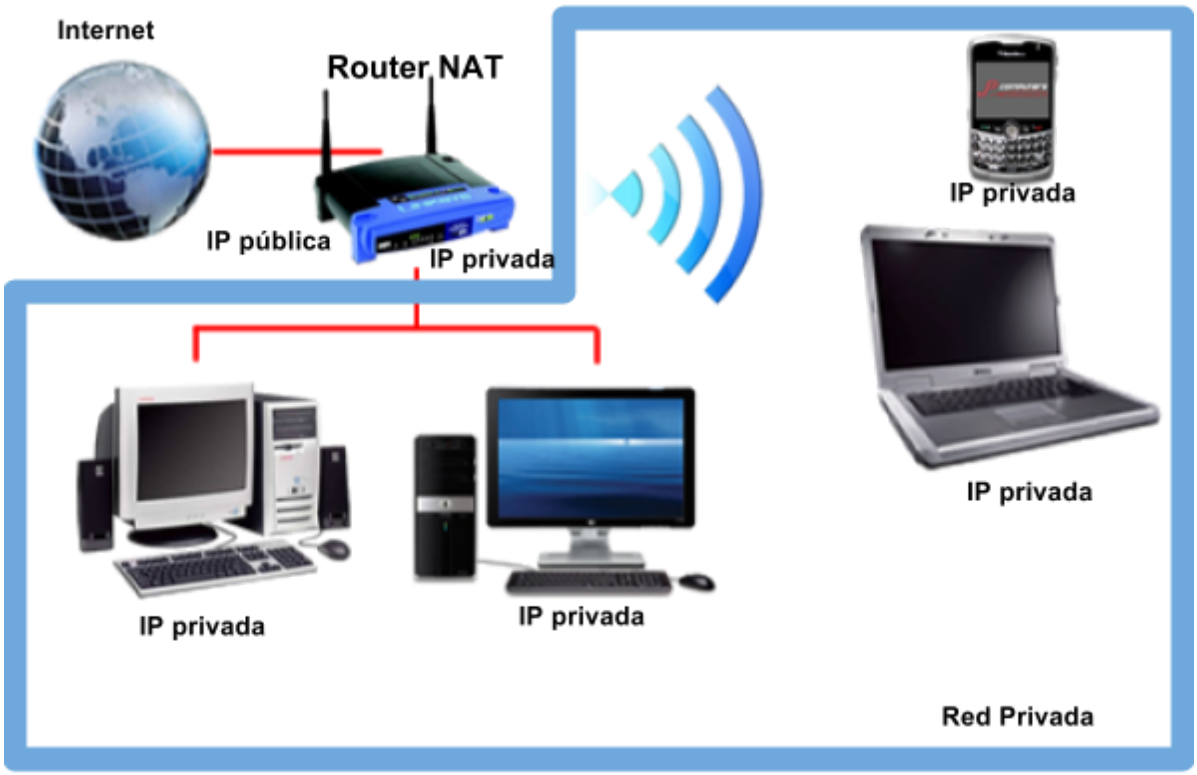


Fig 9. Red doméstica modelo.

Redes LAN y WAN

Una red **LAN** (**L**ocal **A**rea **N**etwork) define a una red que está conformada por una cantidad de nodos relativamente pequeña. Suele estar referida a redes que abarcan un área física reducida, como por ejemplo redes en casas, escuelas, hoteles, universidades, etc. Una red **WAN** (**W**ide **A**rea **N**etwork) se denomina a aquellas redes que se extienden por zonas en las cuales existe una distancia considerable entre sus nodos, y que suelen tener también una cantidad de nodos más alta en relación a las **LAN**. Ejemplos de redes **WAN** son las redes de las ISP (Internet Service Provider), IXP (Internet Exchange Point), redes empresarias con varias sedes, etc.

La diferencia entre el comportamiento de redes LAN y WAN se encuentra en los protocolos que se manejan en la capa física y de enlace únicamente, pues las necesidades a este nivel son distintas. En la capa física, en WAN se utilizan conexiones de alta velocidad (fibras ópticas generalmente) por la amplia distancia entre nodos, mientras que en redes LAN la velocidad de transmisión física no es un factor clave en la calidad de la

comunicación (considerar que el costo de una conexión de alta velocidad es ampliamente superior a una conexión que no es de alta velocidad). En la capa de enlace, en LAN los *frames* se transmiten "inundando" la red, es decir, por *broadcast* (cada *frame* inunda la red y llega a todas las placas de red de los *hosts,* y cada una procesa únicamente los *frames* que se dirigen hacia ella; en el caso de haber interferencias, se transmiten los *frames* de nuevo). En WAN, la transmisión no puede ser por *broadcast* pues al haber una alta cantidad de nodos, la interferencia seria continua y todos los *frames* se estarían continuamente re-transmitiendo. En estas redes, se utilizan otros protocolos como por ejemplo Frame Relay o PPP.

DNS y consumo de servicios por nombres

Como se dijo antes, cada *host* de Internet está identificado por su dirección IP. Estos *hosts* suelen proveer servicios que son consumidos por otros *hosts*, como por ejemplo, nodos de una red privada. Navegación por la web, *Streaming* de música, Sistema de e-mail son algunos ejemplos de los servicios que proveen distintos nodos de Internet, o como también se dice, servicios "*hosteados"* en Internet.

Ahora bien, un *host* que busca acceder a un recurso (consumir un servicio) necesita poder ubicar a dicho servicio en Internet. Si conoce la dirección IP del *host* que provee este servicio. Otra manera (de hecho, la más usual) es ubicar el servicio a través de un **nombre**. Dicho **nombre** necesita ser traducido a una dirección IP. De esta tarea, entre otras, se encarga el protocolo DNS (**D**omain **N**ame **S**ystem). De esta manera, el consumidor realiza una consulta DNS a un **servidor DNS**, el cual realiza una traducción entre el nombre provisto y una dirección IP. Teniendo la dirección IP, el consumidor puede comunicarse con el *host*.

El protocolo DNS se lo ubica en la capa de Aplicación, y se complementa, generalmente, con el protocolo UDP a nivel Transporte.

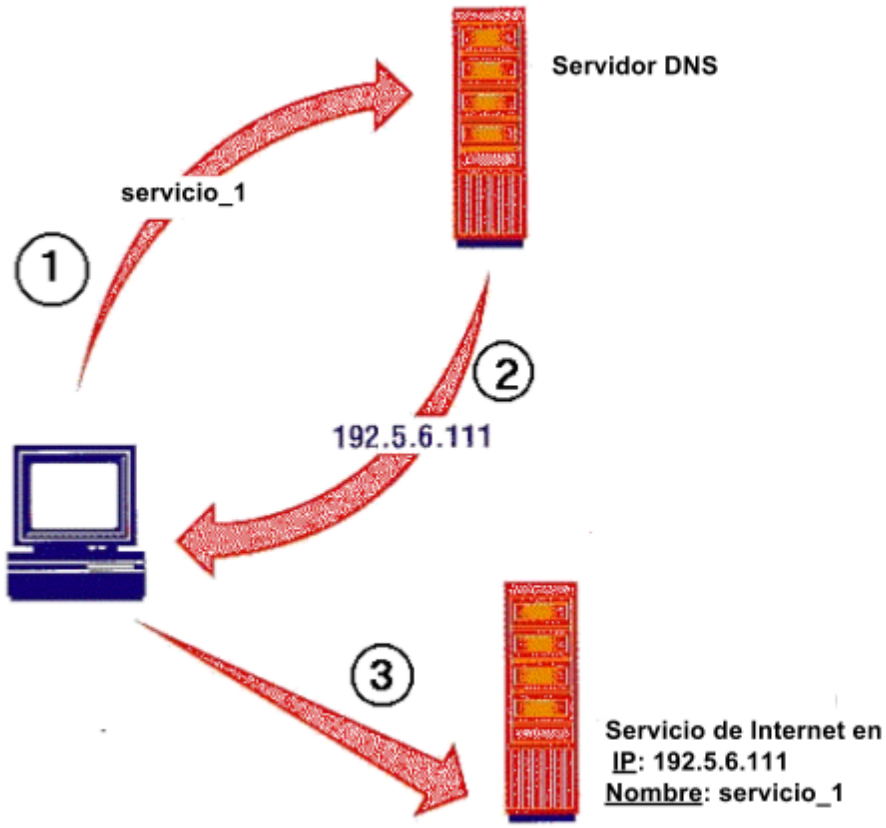


Fig 10. Flujo básico de un consumo de servicio por nombre a través de una consulta DNS

Palabras finales

El marco teórico no fue más que **una leve descripción de algunos conceptos teóricos que se desarrollan en el siguiente trabajo**. Todos los temas que se trataron en esta sección tienen un alto nivel de profundidad y especificación. Lo que es aún más, todos los conceptos que se explicaron se conectan con muchos otros temas técnicos, como por ejemplo concepto de puertos, *routeo* de paquetes, arquitecturas cliente/servidor, procesos, interfaces o *sockets*, por nombrar algunos. Estos temas fueron estudiados para llevar a cabo este trabajo, pero no es el objetivo del trabajo explicarlos.

Esta sección tiene el objetivo de bajar la vara de comprensión del trabajo para que lectores no técnicos puedan entender y apreciar tanto el proceso de investigación y desarrollo como los documentos y programas finales.

Qué se pretende resolver

Detección de patrones de comportamiento

Desde un punto de vista objetivo, el tráfico de red es, por definición, el conjunto de paquetes que salieron de la red y el conjunto de paquetes que entraron a la misma. Sin embargo, efectuando un análisis de dicho tráfico es posible detectar patrones de comportamiento de la red que no son observables a simple vista.

Un patrón de comportamiento va a significar, para este trabajo, a **toda característica o actividad de la red a analizar que emerge a partir del proceso de inspección, relacion, procesamiento y vinculación de los paquetes que conforman su tráfico, tanto encriptado como plano.** A continuación se listan a modo de ejemplo algunos patrones de comportamiento de una red:

- Detección de dispositivos presentes en la red
    - Dispositivos móviles/portables
    - Computadoras
    - Dispositivos IoT
    - Electrodomésticos de audio y video
- Actividad en red para cada dispositivo
    - Aparición en la red / Conexión
    - Picos de tráfico
    - Momento de ausencia / Desconexión
- Servicios consumidos por cada dispositivo
    - Servicios de mensajeria
    - Redes sociales
    - Contenido multimedia

Obtención de tráfico

El análisis para descubrir los patrones de comportamiento se realiza sobre el conjunto de datos. Por lo tanto, la obtención del tráfico (proceso conocido como *sniffing*) es una tarea indispensable para cumplir este objetivo.

No solo la obtención es clave para el análisis de del tráfico, sino también el **lugar de obtención** es importante para la caracterización del comportamiento de la red. Esto se debe principalmente a la presencia de NAT.

El dispositivo que realiza NAT pertenece a dos redes incompatibles entre sí a nivel IP. Pertenecer a dos redes equivale a decir que posee dos interfaces, pues una interfaz se define como **el punto de interconexión entre una computadora y una red.**

Sean A y B estas dos redes, siendo X el *gateway* que realiza el proceso de NAT. Si el *sniffing* se realiza en la interfaz B de X, entonces los paquetes que salgan de A van a tener la dirección IP origen equivalente a la dirección IP de X en la interfaz B (proceso SNAT). Además, los paquetes que retornen de B a A a través de X, van a tener dirección IP destino la equivalente a la de X en B(proceso DNAT). Por lo tanto, si se desea realizar un análisis de la red A con ese tráfico *sniffeado* en B, el gran obstáculo que se presenta es que **todo el tráfico de la red A va a manifestarse desde un solo *host* (el *gateway*).** Resulta imposible rastrear cada dispositivo de la red siguiendo el principio básico de que cada uno tiene una dirección IP distinta dentro de la red A.

En el contexto del trabajo que se desarrolla, se consideran las redes A y B como redes **LAN** y **WAN**, respectivamente. **Se va a considerar como premisa que el lugar de captura del tráfico va a ser la interfaz WAN del *gateway* (o router) encargado de realizar el proceso de NAT entre la red LAN y la red WAN.**

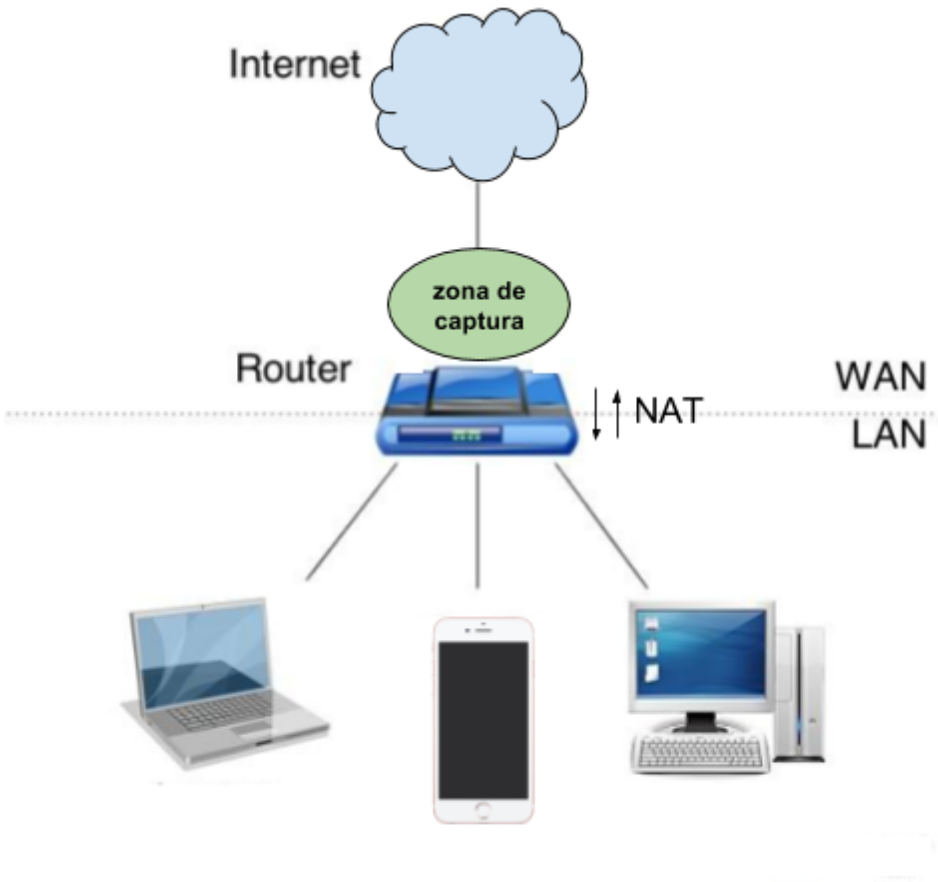


Fig 11. Representación de un escenario de redes en el cual se basa el trabajo.

Objetivo del trabajo

Habiendo detallado el concepto de patrón de comportamiento y habiendo establecido el lugar (o interfaz) de captura, se puede definir a alto nivel el objetivo del trabajo:

El objetivo de este trabajo es construir un sistema que pueda caracterizar al escenario que se manifiesta en una red LAN a través de la detección de sus patrones de comportamiento y a partir del análisis de tráfico de dicha red capturado desde la interfaz WAN del *gateway* que efectúa el proceso de NAT.

La dificultad de lograr este objetivo radica en varias restricciones:

- Para la deteccion y caracterizacion de dispositivos (individualización de terminales) que están involucrados en la red LAN, se tiene el gran obstáculo de que el tráfico se encuentra “NATeado” (traducido por NAT) por el *gateway*. Es decir, todos los datagramas del tráfico saliente van a tener la dirección IP origen del *gateway* en la interfaz WAN, y el tráfico entrante lo mismo pero para la dirección IP destino.

- Presencia de dispositivos similares en una red son casi imposibles de detectar. Esto quiere decir que si en una red existen dispositivos con características similares (mismos modelos, mismas versiones de sistemas operativos, etc) el tráfico que generan es indistinguible entre sí. Por ejemplo, un celular A navega por dos pestañas en el *browser*. El tráfico que genera A en este escenario es indistinguible del tráfico que genera un escenario en el cual A navega en una sola pestaña y B, un celular exactamente similar a A, navegando en otra pestaña del mismo *browser*.

- Para la detección y caracterización de servicios, la restricción radica en que en la actualidad la mayoría de los servicios que involucran datos sensibles de los clientes que lo consumen tienen tráfico encriptado. Esto significa que la inspección de esos paquetes encriptados no revelan absolutamente nada de información aparte de las direcciones IP y puertos involucrados.

- No existe un mapeo claro entre servicios y el nombre con el que están registrados en la infraestructura de DNS, es decir, entre servicio y URL. Por ejemplo, el nombre (o url) **fbcnd-sphotos-g-a.akamaihd.net** representa un servicio que provee fotos de Facebook.

- Tampoco existe un mapeo claro entre servicios y direcciones IP. Esto se debe a que las asignaciones de direcciones IP son de carácter dinámico (una de la razón por la cual aparece la infraestructura DNS). Por lo tanto, un servicio puede estar en un instante dado en un *host* registrado bajo una determinada dirección IP y en un instante después, figurar en otra dirección IP.

Por último, se presenta la dificultad más importante de este trabajo y se basa en lo siguiente:

Gran parte del potencial y de la vigencia de las redes privadas LAN radica en la seguridad que proveen. Como se explica en la sección anterior, una de las características de las redes privadas (por algo su nombre de “privadas”) son los mecanismos de protección frente a los *hosts* públicos que forman parte de Internet. Estos mecanismos de protección no solo aparecen en las políticas de Firewall o en el proceso de NAT, sino también en el tratamiento de cierta información de los paquetes que salen de cada *host* de la red privada (como por ejemplo la generación de los valores del campo ID de los datagramas IP). Este tratamiento evita, en parte, ciertas estrategias que podrían ponerse en práctica para lograr el objetivo del trabajo.

Al fin y al cabo, **parte del objetivo de este trabajo es poder deducir ciertas características de una red, características que la red busca proteger.**

Filosofía de “Código Abierto”

Si bien el motor de este trabajo es el desarrollo de un proyecto final de la carrera universitaria, se va a seguir una filosofía de código abierto. Esto quiere decir que el sistema que se desarrolle para lograr el objetivo del trabajo va a contribuir con la comunidad “Open Source”, lo que significa que tanto la documentación como el código fuente involucrado va a ser público. Lo que es aún más, todas las herramientas y librerías que se se utilicen a lo largo de la investigación y el desarrollo van a ser también de código abierto. El principio que se persigue es básico: **cuando la comunidad de programadores / analistas / científicos pueden acceder, interpretar, modificar, redistribuir el código fuente de un sistema, éste evoluciona, se desarrolla y mejora.**

Aplicaciones en la realidad

El análisis de tráfico que se efectúa en el sistema que se plantea como objetivo del trabajo tiene diversas aplicaciones reales, dependiendo de las características del tráfico con el cual se esté trabajando. Existen básicamente dos tipos de análisis de tráfico y cada uno es útil para distintas aplicaciones.

Por un lado, **se puede realizar un análisis de tráfico en un sistema "real time"**. Estos significa que se procesan lo paquetes en el instante en el cual se está realizando la captura, es decir, la captura es "en vivo". En este escenario, existen múltiples aplicaciones para un sistema que realice lo que se describe como objetivo y que trabaje con este tipo de análisis, principalmente **sistemas de monitoreo**. Por ejemplo, sistemas que estén monitoreando los servicios que se consumen en una red, sistemas que brinden QoS (es decir, control sobre el ancho de banda de una red) o incluso sistemas que rastreen dispositivos y/o gente por motivos de seguridad. lo cual significa que el tráfico procesado se está capturando "en vivo".

Por otro lado, **se puede analizar tráfico luego de que este haya sido capturado**. Esto quiere decir que los paquetes que se están procesando pertenecen a una captura realizada en el pasado. Bajo este escenario, un sistema como el de este proyecto puede ser útil principalmente para el área de la informática forense. La informática forense es una disciplina cuyo objetivo principal es el análisis de sistemas informáticos en busca de evidencia que colabore a llevar adelante una causa judicial o a emitir un fallo judicial. Para esta disciplina, el análisis de tráfico no sirve únicamente en casos de delitos o fraudes informáticos, sino que puede ser útil para cualquier tipo de causa. El sistema que plantea el proyecto puede ser útil para realizar una investigación sobre las actividades de una persona, sobre conversaciones entre sospechosos, etc.

¿Por qué el lugar de captura?

Ahora bien, con respecto a que el sistema puede ser utilizado en el ámbito de la informática forense, es importante justificar el lugar de captura que fue impuesto anteriormente.

Como se mencionó anteriormente, una de las premisas del trabajo es que el lugar de captura del tráfico va a ser la interfaz WAN del *gateway* (o router) de una red LAN. La clave radica en que la interfaz LAN del *gateway* (lugar donde sería ideal realizar una captura, pues no interviene NAT) se la puede considerar propiedad privada, y por lo tanto, inviolable en términos de acceso y colocación de un *sniffer* (instrumento para llevar a cabo el *sniffing*). Sin embargo, la interfaz WAN del *gateway* es propiedad del proveedor de Internet (ISP) contratado por el dueño de la red LAN. Con la orden legal apropiada, es posible proceder a realizar un *sniffing* en esta interfaz.

Caso real

Este proyecto puede servir para investigar a una persona o grupo de personas de manera no intrusiva ya que solo se toma el tráfico por fuera del domicilio. Analizando días o semanas completas de tráfico se pueden detectar patrones como ser, cantidad de personas en el domicilio, llegadas y partidas de personas en el domicilio, comunicaciones con externos. Y resulta muy interesante para aquellos protocolos que generan conexiones P2P ya que se pueden obtener las direcciones destino de terceros.

Esto es análogo a los informes de comunicaciones que entregan las empresas de telefonía, pero con el agregado que **en este caso se obtiene también el contenido de las comunicaciones.**

Sección II: **Metodología**

Introducción

Esta sección del informe se centra en la **metodología**. La metodología hace referencia al conjunto de procedimientos (o actividades) utilizados para alcanzar un objetivo. **El objetivo de esta sección es la de realizar una descripción detallada de cada una de las actividades que llevaron a cabo cada procedimiento.**

Este trabajo se puede dividir en tres etapas principalmente: **la etapa de investigación, la etapa de desarrollo** y la **etapa de prueba**.

La etapa de investigación consistió en todas las actividades previas al desarrollo que involucran:

- Conocer **el estado del arte**, es decir, el estado último del conocimiento en términos de Investigación y Desarrollo de aquellas áreas que involucra este proyecto. El estado del arte involucra la base teórica que existe en investigación sobre los temas que trata este trabajo.

- **Observar y analizar detalladamente** varias capturas de tráfico conociendo previamente su comportamiento para poder detectar aquellos “lugares” (tipo de paquetes, campos de paquetes, etc) donde se ubicaba la información útil para la detección de patrones de comportamiento.

- Poder **definir el alcance de un posible desarrollo de un sistema** que solucione el problema que se plantea como objetivo del trabajo. El alcance iba a estar restringido por las limitaciones técnicas, limitaciones en materia de conocimiento y limitaciones en términos de tiempo disponible para el desarrollo.

La etapa de desarrollo fue posterior a la etapa de investigación, cuando estaba ya definido el alcance que iba a tener el sistema que se deseaba construir. La etapa de desarrollo involucró a todas las actividades cuya finalidad fuera la construcción del sistema de software que resuelve el problema planteado en este trabajo. Algunas de las actividades que esta etapa involucró son:

- Elección de la tecnología a utilizar
- Diseño de la arquitectura del *software*
- Escritura del código fuente

La etapa de prueba consistió en las actividades que se llevaron a cabo para poder verificar que el sistema desarrollado pueda cumplir con las expectativas planteadas, es decir, que pueda cumplir el objetivo que se propone en el trabajo. El objetivo, expresado nuevamente, es **crear un sistema que pueda caracterizar al escenario que se manifiesta en una red LAN a través de la detección de sus patrones de**

comportamiento y a partir de trafico capturado en la interfaz WAN del *gateway* (tráfico *NATeado*).

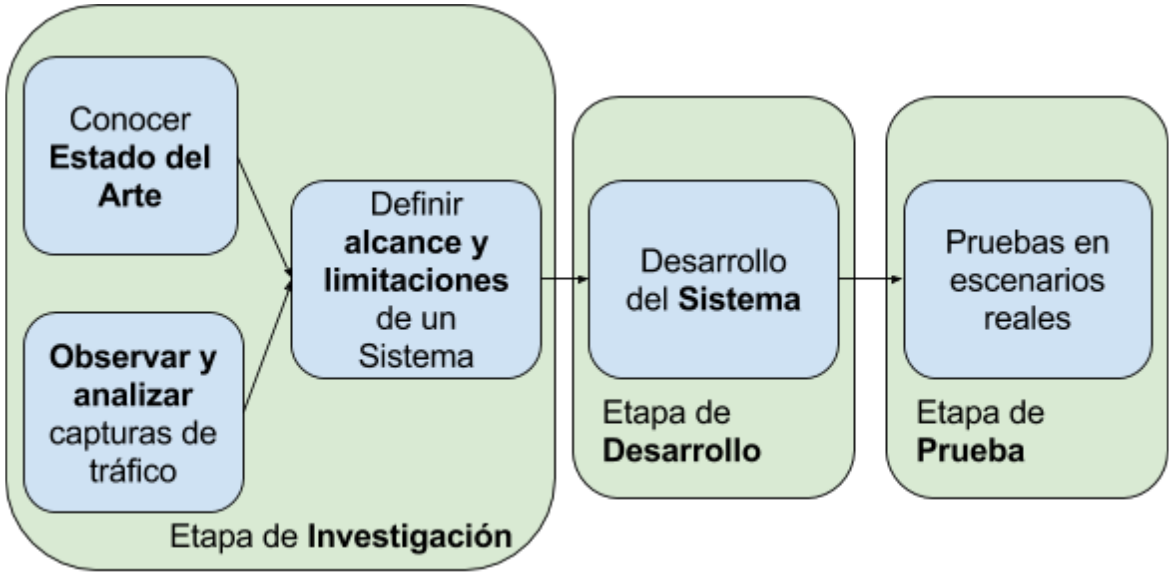


Fig 12. Metodología que se aplicó para este trabajo.

La figura anterior (Fig 12) representa un resumen de la metodología que se utilizó en este trabajo. Es importante remarcar nuevamente que el desarrollo del sistema tuvo lugar recién cuando se pudo finalizar la definición del alcance y las limitaciones del sistema. Al mismo tiempo, esta definición del alcance y limitaciones pudo realizarse una vez que se conocía el estado del arte y una vez que se hayan estudiado las capturas de tráfico con las que se iba a ejecutar el sistema.

En esta sección se detallan cada una de las actividades que se efectuaron para la realización de este trabajo. Se agrupan bajo las siguientes categorías:

- Investigación del estado del arte
- Recolección y análisis de capturas

- Desarrollo del Sistema

- Etapa experimental en escenarios reales

Investigación del estado del arte

Como se mencionó antes, el **estado del arte** se entiende como el estado actual de conocimiento en las áreas que involucra este trabajo. Este conjunto de conocimiento es influenciado por dos áreas principalmente, la **investigación** y el **desarrollo**.

El área de la **investigación**, en este contexto, va a involucrar todos aquellos estudios que tengan como objetivo la generación de nuevo conocimiento; mientras que el área de **desarrollo**, en este contexto, se entiende como los sistemas existentes que realizan una tarea similar a la que se pretende resolver en este proyecto.

Área de la Investigación

La búsqueda de contenido de investigación, para este trabajo, se redujo al **ámbito académico**. El principal problema que se presentó fue la poca cantidad de material relacionado a la investigación disponible.

Para obtener material de investigación que sea útil para el proyecto, se consideraron dos fuentes principalmente:

- Estudio de *papers* académicos, o lo que es similar, artículos científicos.
- Reunión con profesores.

Ningún *paper* consultado estudiaba el mismo objetivo que se plantea como objetivo del trabajo. Se pudieron encontrar algunos artículos que partían de otros supuestos o que utilizaban estrategias obsoletas. La información que se extrajo de estos *papers* va a estar detallada en la sección siguiente.

Con respecto a la otra fuente de material, se tuvo una reunión con el Dr. Ing. José Ignacio Alvarez-Hamelin, docente del ITBA y especialista en redes de computadoras, en la que transmitió ideas clave para la implementación del sistema y para poder tener más en claro el alcance y las limitaciones del sistema a desarrollar.

Área del Desarrollo

En el ámbito del desarrollo que conforma el **estado del arte**, lo más importante es remarcar que **no existe hoy en día un sistema que se encargue de la detección de patrones de comportamiento de una red detrás de un *gateway*.**

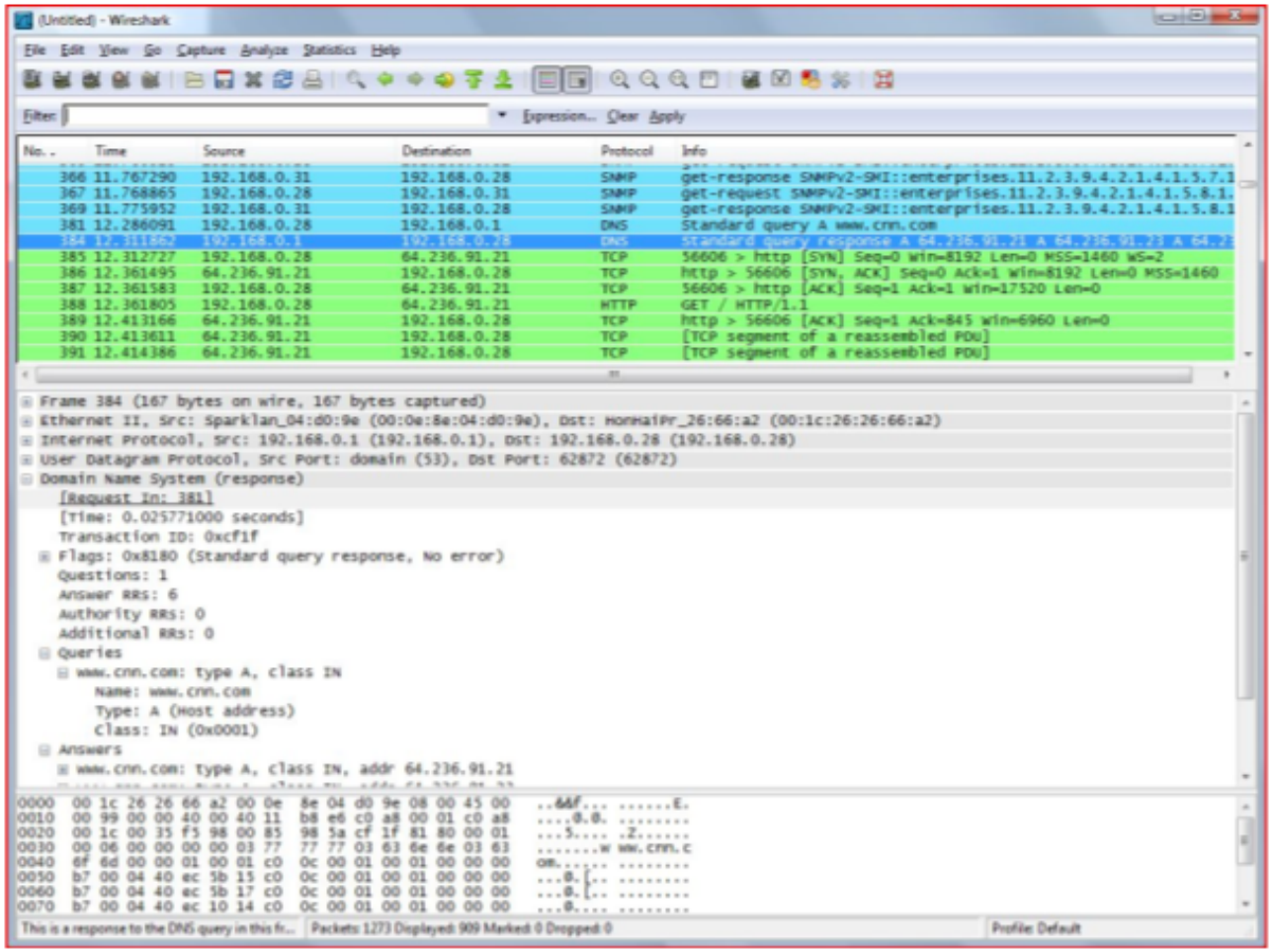


Fig 13. Interfaz del programa Wireshark. Solo permite la inspección de paquetes.

Por un lado, existen los analizadores de tráfico, que permiten la inspección de los paquetes que conforman una captura de tráfico de red. La herramienta más conocida de esta categoría es *Wireshark*, herramienta que además posee la opción de efectuar la captura (*sniffing*). Estos analizadores de tráfico no tienen ningún tipo de procesamiento sobre los paquetes, sino que ofrecen una interfaz gráfica para poder inspeccionarlos.

Por otro lado, si se quiere obtener algún tipo de procesamiento sobre los paquetes y no una simple inspección, lo que sí existe son herramientas que permiten visualizar distintos aspectos del tráfico de red capturado, aspectos que sería muy difícil poder identificar o calcular trabajando con un analizador tradicional de tráfico. Estas herramientas tienen un alto desarrollo en materia de visualización para la fácil identificación de ciertos patrones de la red. Un ejemplo es *CapAnalysis*, una herramienta web de visualización de grandes cantidades de tráfico. Permite, entre otras cosas, visualizar estadísticas sobre las direcciones IP más consumidas, visualizar la localización de las direcciones IP involucradas, visualizar la actividad de la red según la hora o visualizar la cantidad de tráfico por protocolo.

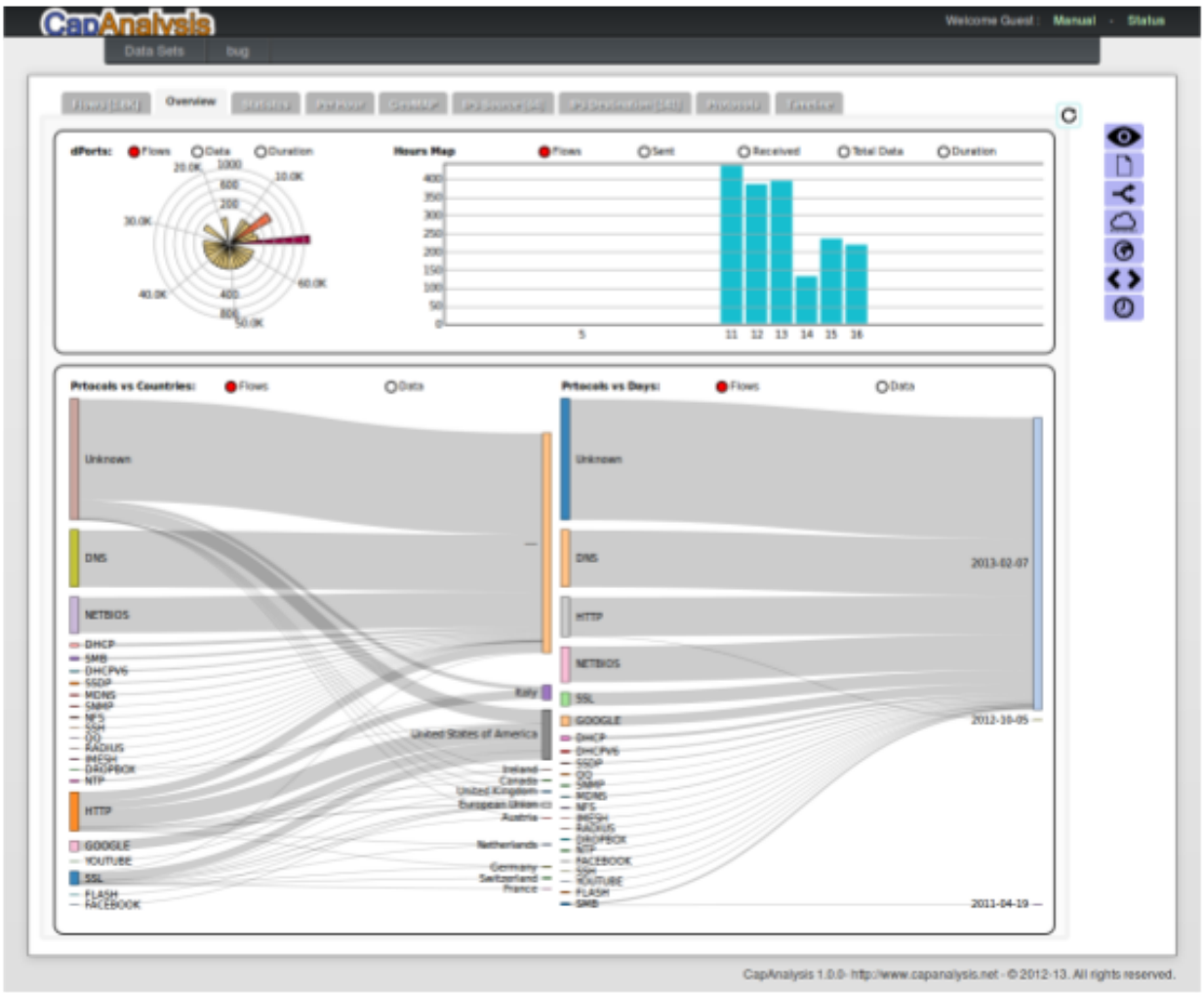


Fig 14. Interfaz del CapAnalysis. Permite la visualización de distintos aspectos del tráfico. capturado.

Ninguna de estas herramientas realizan una análisis exhaustivo del tráfico con el objetivo de obtener información extra que no surge con la simple inspección individual de cada paquete. Para poder realizar la detección de patrones de comportamiento de una red LAN (sea tráfico *NATeado* o no), como por ejemplo la individualización de dispositivos o la identificación de servicios consumidos, es crucial la interpretación del tráfico.

A continuación se presenta un ejemplo simple de análisis de tráfico que puede identificar el consumo de un servicio:

"En la captura de tráfico de una red X, se puede detectar una consulta DNS para obtener la dirección IP de la URL *photos.facebook-cdn.com*. La respuesta DNS ante tal consulta informa que esa URL se encuentra bajo la IP *X.Y.W.Z*. Más adelante, se puede observar una comunicación TLS (encriptado) con la dirección *X.Y.W.Z.* Esto significa que existe un usuario que esta consumiendo los servicios de la URL *photos.facebook.cdn.com* (a menos que haya habida otra consulta DNS por otra URL que haya mapeado a la misma

direccion IP). Si además, se sabe que esta URL se encarga de enviar las fotos de la plataforma *Facebook* a un usuario *logueado,* entonces se puede concluir que existe un usuario en la red X que está consumiendo la plataforma social Facebook en ese instante."

Este tipo de análisis no es posible realizarlo hoy en día de una manera automática, pues ninguna herramienta provee este análisis del tráfico. La única manera de realizarlo es de manera "*manual"*, observando en un analizador de tráfico los distintos eventos que llevan a deducir el consumo de un servicio. Cuando el tráfico involucra varios GB de captura, el proceso "*manual*" queda descartado.

Para el desarrollo de un sistema que cumpla con las expectativas de este proyecto, va a tener que trabajarse en la automatización de este tipo de análisis, pues es la única manera de extraer del tráfico de red información adicional que no surge de la inspección independiente de paquetes. En este trabajo, este tipo de análisis va a ser denominado **análisis de tráfico relacional e interpretativo**.

Recolección y análisis de capturas

Siguiendo en la etapa de investigación del proyecto y en paralelo a la investigación acerca estado del arte, tuvo lugar un proceso de recoleccion y análisis detallado de capturas. Al poner en evidencia la falta de *artículos* académicos que plantearan estrategias acerca de cómo cumplir el objetivo perseguido o la falta de herramientas que automatizaran lo que antes se denominó **análisis relacional e interpretativo**, no quedó otra opción que **recurrir a las capturas propiamente dichas y observar paquete por paquete para identificar aquellos recursos clave donde sea posible extraer información necesaria para realizar la detección de patrones de comportamiento.**

Esta etapa consistió en **realizar capturas sobre escenario conocidos para poder ver de qué manera se manifestaba en el tráfico la información que se pretendía detectar**. Con “escenarios conocidos” se refiere a que las capturas que se realizaron fueron sobre una red completamente controlada en términos que se sabía exactamente qué dispositivos formaban parte de la misma, el horario de conexión/desconexión, los servicios que consumía cada dispositivo, etc.

El objetivo era ver en el tráfico, por ejemplo, cómo se manifestaba un mensaje de *WhatsApp* saliente de un Nexus 5 con Android 6.0, o como se manifestaba la conexión a la red de un iPad con iOS 10, o como navegaba en Internet a través de un *web browser* una *notebook*.

Proceso de recolección de capturas

Si bien el escenario real del proyecto establece como tráfico a analizar aquel tráfico capturado desde la interfaz WAN de un router que realiza el proceso de NAT, para esta etapa de recolección y análisis se optó por que la captura se realice dentro de la red LAN. Es decir, que se pueda distinguir cada dirección IP que posee cada uno de los dispositivos que pertenecen a la red (no se observa tráfico *NATeado*). Esto se debe a que con esta estrategia, **fue posible aislar cada dispositivo y ver únicamente su tráfico** (filtrando el tráfico por dirección IP origen de cada dispositivo).

Ahora bien, la captura de una red LAN se puede realizar de varias maneras:

1. Con un dispositivo *sniffer* (también denominado *network tap*) conectado a la interfaz LAN del *gateway*.

2. Con un *sniffer* en modo “promiscuo” (*Wireshark* provee esta funcionalidad). Se basa en interpretar todos los frames que llegan a la placa de red de la computadora, los cuales son procesados aunque la MacAddress destino no coincida con la de la placa. Solo funciona en redes no seguras.

3. Generar una red en la cual el *gateway* sea la computadora misma. Esta funcionalidad viene en la opción de *"compartir Internet"* y se encuentra presente en la mayoría de los sistemas operativos modernos. La idea es que si la computadora posee una conexión por cable a Internet, puede utilizar la misma placa Wi-Fi para emitir una señal similar a la de un *Access Point* permitiendo que otros hosts se conecten a través de su Wi-Fi a la conexión que posee la computadora. La computadora genera una red privada y hace tanto de *gateway* como de servidor DHCP (asigna las direcciones IPs a los *hosts* que se pretenden conectar a la red). El hecho que la computadora misma se encuentre en una LAN no introduce ningún problema, pues la red que se genera es independiente a la que pertenece dicha computadora.

Para este trabajo, se optó por la tercera opción debido a su practicidad. Realizando el *sniffing* desde el programa *Wireshark* sobre la interfaz del Wi-Fi, resultaba muy simple efectuar las capturas de tráfico de toda la red generada por la computadora misma al "compartir Internet". A continuación se presenta un esquema del escenario descrito anteriormente.

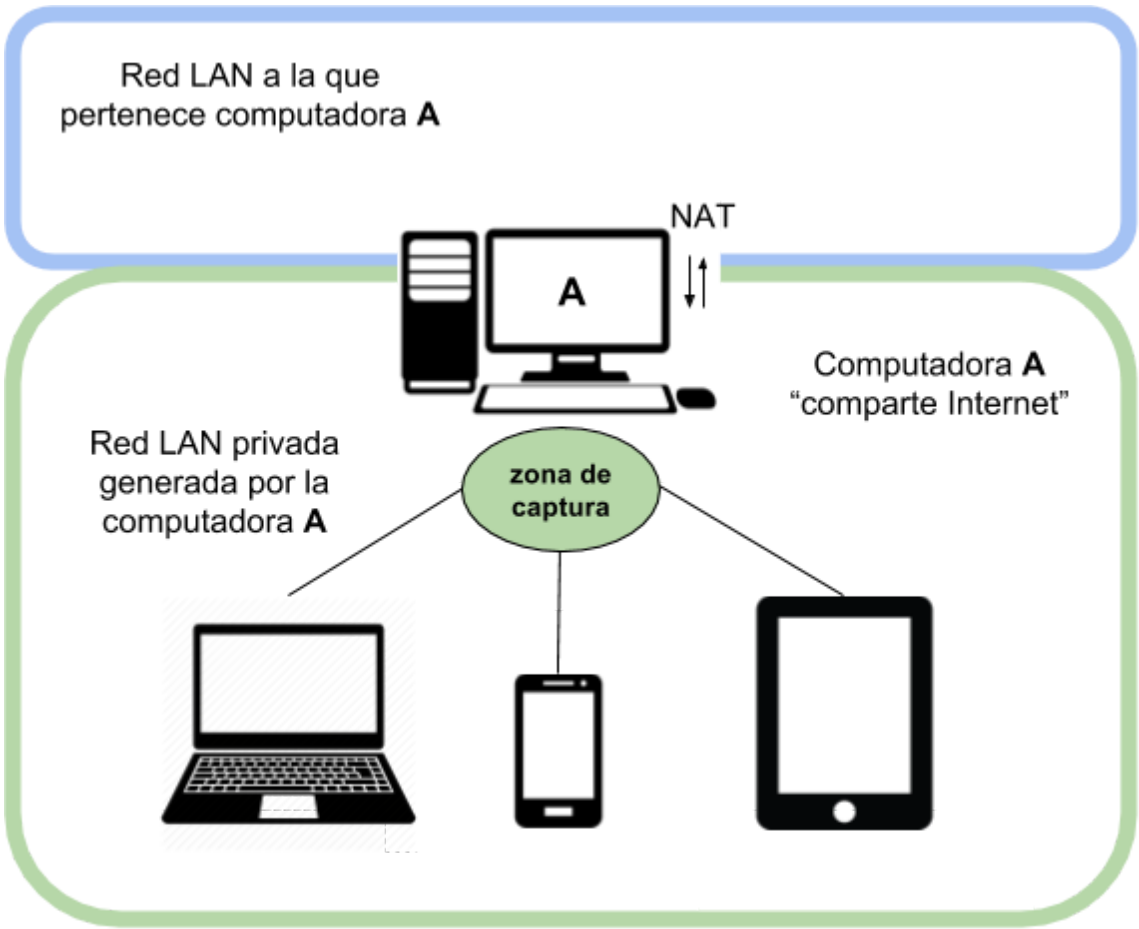


Fig 15. Representación de como se realizaron las capturas en la etapa de recolección

Siguiendo esta estrategia, se generaron tantas capturas como información que se quería detectar. **Para cada evento que se quería identificar, se armaba una red y se simulaba únicamente dicho evento para poder capturarlo y estudiarlo luego.** Por ejemplo:

- Captura de un Android conectándose a la red LAN.
- Captura de una laptop viendo videos en *youtube.*
- Captura de un iPad viendo una película en *netflix.*
- Captura de un Nexus mandando un mensaje en *telegram.*
- Captura de un iPhone jugando al *Clash Of Clans.*

Proceso de análisis de capturas generadas

Luego de realizar cada captura, el proceso consistía en **una lectura detallada de todos paquetes del tráfico que manifestaba el dispositivo a estudiar**. Como se dijo antes, al conocer la dirección IP del dispositivo que se estaba estudiando (pues las capturas se estaban realizando directamente en la red LAN), resultaba muy fácil aplicar filtros por dirección IP para poder aislar únicamente el tráfico del dispositivo en cuestión. La herramienta que se utilizó para este proceso fue *Wireshark.*

El objetivo principal de este proceso de análisis de capturas era poder descubrir dónde y cómo se manifestaban los eventos que se simulaban en cada captura.

Si bien cada captura era representativa de un evento particular a estudiar, lo que se intentó buscar fueron **generalizaciones**, es decir, comportamiento que no solo aplique al caso particular de estudio, sino a muchos casos similares.

Por ejemplo, se pudo generalizar que los *requests* HTTP que tenían como objetivo consumir una API para una determinada aplicación en los sistemas operativos iOS tenían una sintaxis en el campo *User Agent* que dejaba información acerca de la aplicación y de la versión de iOS en cuestión. Esta generalización fue posible gracias al análisis y la inspección detallada de cada paquete HTTP de aquellas capturas donde se registraba el tráfico de un usuario consumiendo una aplicación de este tipo (por ejemplo, aplicación de noticias, juegos, etc.) desde un dispositivo iOS (iPad, iPhone, etc).

En muchos casos también fue posible corroborar y justificar algunas de las generalizaciones descubiertas, recurriendo a los códigos fuente de los sistemas operativos o a la documentación de los servicios de los sistemas operativos, por ejemplo.

Ejemplos de observaciones en capturas

A continuación se presentan algunos casos puntales de observaciones de capturas que fueron útiles para detectar cómo se manifestaban los eventos que se simulaban en cada captura.

Conexión a la red de un dispositivo Android

Fig 16. Request (rojo) y Response (azul) HTTP de un dispositivo Android al conectarse a la red.

En la imagen de la figura 16, se puede observar en un intercambio HTTP que ocurre cuando un dispositivo Nexus 5 con Android 6.0.1 se conecta a la red.

Este comportamiento pudo generalizarse y se llegó a que **todo dispositivo con Android 6 (6.1, 6.0.1, etc) al conectarse a una red envía un "*GET /generate_204*" por HTTP plano y en el *User Agent* deja ver información acerca del dispositivo.**

Esta generalización fue corroborada y su explicación proviene de lo que se conoce como *portal cautivo* (el portal fuerza a los usuarios a pasar por una página especial de autenticación o aceptación de condiciones si quieren tener acceso a Internet, muy usado en hoteles, shoppings, etc.). En el caso que este *request* no tenga *response*, entonces el dispositivo se encuentra en un *portal cautivo*.

Consumo de servicios HTTP por parte de una aplicación en un dispositivo iOS

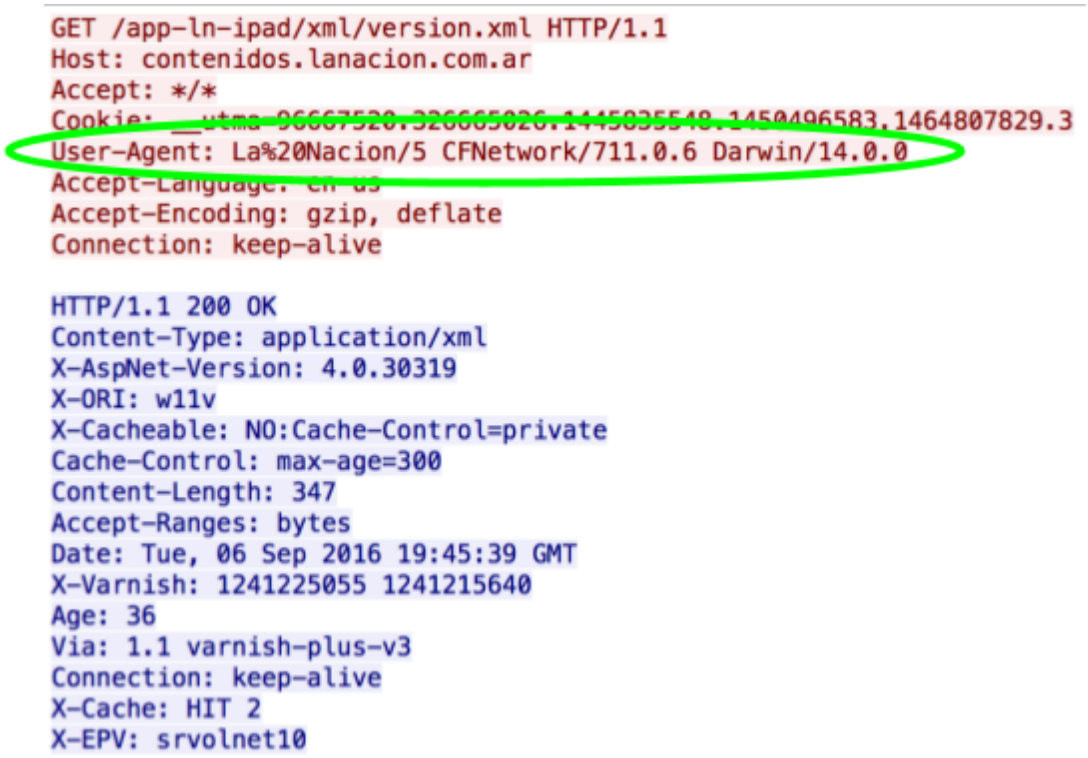


Fig 17. Request (rojo) y Response (azul) HTTP de un dispositivo iOS al usar una aplicación que consume servicios por HTTP.

En la imagen de la figura 17, se puede observar lo que explico en el ejemplo anteriormente sobre los dispositivos iOS que usan una aplicación que consume servicios de Internet por HTTP. En este caso, se trata de la aplicación de noticias del diario La Nación.

Como se puede observar, en el campo *User Agent* aparece el nombre de la aplicación, como también versiones de "*CFNetwork*" y "*Darwin*". "*CFNetwork*" es el nombre del servicio que provee a los desarrolladores librerías para el manejo de los protocolos de comunicación. Se pudo descubrir también que sabiendo las versiones de "*CFNetwork*" y "*Darwin*", se puede deducir la versión del sistema operativo iOS.

Es importante poner en evidencia la importancia de haber descubierto estas generalizaciones. Solo con estos dos ejemplos ya se puede construir un sistema que pueda detectar dos patrones de comportamiento, por un lado a un dispositivo Android conectándose a la red, y por el otro el consumo de algunas aplicaciones de algún dispositivo iOS que pertenezca a la red. La única manera de descubrir estas generalización fue la captura y el análisis exhaustivo de capturas de tráfico de distintos eventos.

Desarrollo del sistema

Habiendo terminado la etapa de investigación y habiendo definido tanto el alcance como las limitaciones del sistema a desarrollar, comenzó la etapa de desarrollo. Como se explica anteriormente, esta etapa consistió en todas las actividades relacionadas a la construcción del sistema de software cuya funcionalidad solucione lo propuesto como objetivo del trabajo.

El desarrollo del sistema abarcó los últimos 3 meses del proyecto (a partir de Septiembre de 2016), habiendo necesitado los primeros 6 meses del proyecto (desde Abril de 2016) para la etapa de investigación.

Core

Lo que se comenzó a desarrollar primero fue el *backend* del sistema, o *core*, como se denominara en este trabajo. Este es el componente más importante del sistema, pues es la implementación de toda lógica que recibe como entrada los archivos de captura y que tiene, como *output*, la descripción de los patrones de comportamiento de la red en cuestión. Las etapas que estuvieron involucradas en la construcción del *core* se pueden enumerar en:

1) Elección y familiarización con las tecnologías a utilizar

No solo el paradigma y el lenguaje de programación, sino también las librerías que permiten realizar la inspección de los paquetes a partir de un archivo de captura de tráfico.

2) Definición de arquitectura

Diseñar la arquitectura del sistema, es decir, los componentes, la comunicación entre componentes, los modelos y sus características, etc.

3) Programación de funcionalidad incremental

Para el proceso de programación del *core*, se implementó un modelo **incremental.** En este modelo, la idea era ir agregando funcionalidad manteniendo siempre una base del sistema funcionando con toda la funcionalidad previa implementada. Primero se busco programar la detección de los dispositivos, para luego proceder con los servicios que consume cada dispositivo; y finalmente agregar la capacidad de detectar servicios “anónimos”, es decir, servicios que por sus características no son atribuibles a ningún dispositivo.

GUI

Si bien el *core* tiene la opción de ser ejecutado directamente desde una terminal bajo un comando específico o también funcionar como una API para que cualquier desarrollo de *front-end* lo consuma, se decidió incluir en el desarrollo del sistema una interfaz gráfica básica que consuma el *core.* Esta componente se denomina, en este informe, la *GUI* (por sus siglas en inglés, *graphical user interface*).

El desarrollo de la *GUI*, al ser un componente relativamente simple en complejidad en relación al *core* (es solo el consumo de los servicios del *core* y una manera “amigable” de mostrar los resultados del *core* al usuario), involucró únicamente las etapas de:

1. Elección de tecnología
2. Programación.

El desarrollo de la *GUI* tomó lugar hacia el final de la etapa de desarrollo, cuando ya se contaba con un *core* avanzado y con un *output* estandarizado para ser consumido como API.

Tanto del *core* como de la *gui* se va a hablar en la sección de Implementación. En esta sección solo se mencionan cuáles fueron las etapas en el desarrollo de cada uno de estos componentes.

Etapa experimental en escenarios reales

La etapa experimental en escenario reales forma parte de la **etapa de prueba** del sistema. Esta etapa es la última de las etapas de la metodología implementada para la resolución de este proyecto (recordando, etapa de investigación + etapa de desarrollo + etapa de prueba) y consiste en **poner a prueba el sistema en escenario reales para ver como responde de acuerdo al alcance y limitaciones definidas anteriormente, y, de ser necesario, realizar los ajustes correspondientes para terminar de consolidar el sistema desarrollado.**

Para esta etapa, se consideraron tres instancias de prueba:

1) Pruebas con capturas genéricas propias

Son capturas que se obtuvieron siguiendo la misma estrategia que en la recolección de capturas en la etapa de investigación, es decir, a partir de una computadora que genera una red a partir de la opción de “compartir Internet”.

A diferencia de las capturas anteriores de la etapa de investigación, cada captura no representaba un solo evento en particular, sino que cada captura poseía varios eventos (conexiones/desconexiones de dispositivos, consumo de distintos servicios, etc.). Muchas de estas capturas involucraron a compañeros que colaboraron con las pruebas y conectaban sus dispositivos a la red generada para poder registrar su tráfico.

2) Pruebas con capturas desconocidas facilitadas por el tutor

El tutor del proyecto brindo en esta etapa una serie de capturas de tráfico. Lo interesante de este escenario de prueba era que él era el unico que conocía que eventos habían tenido lugar en la red (que dispositivos formaron parte, que servicios consumieron, etc.). El objetivo era correr el sistema desarrollado, detectar los patrones de comportamiento y corroborar si los resultados eran correctos.

3) Pruebas de stress con capturas extensas

Ahora bien, los dos tipos de capturas anteriores eran algo “ficticias”, pues no era capturas “reales”, sino que era tráfico generado especialmente para la captura. Para testear realmente el sistema en un ámbito real, fue necesario hacer capturas reales (de varias horas de *sniffing*) donde se pueda trabajar con tráfico verdadero de una red LAN, y no un escenario simulado para una captura representativa.

Esta prueba de stress (pues los archivos de captura en un proceso de *sniffing* extenso ocupan varios GB) va a constituir la evaluación final de la eficacia que tiene el sistema desarrollado.

Sección III: **Resultados etapa de investigación**

Introducción

La etapa de investigación permitió establecer la base de conocimiento a partir de la cual se iba a desarrollar el sistema propuesto. **Esta sección comprende una descripción de las observaciones que se obtuvieron en este proceso de investigación.**

En esta sección, primero se van a mencionar algunos conceptos clave en el comportamiento de los dispositivos NAT. Dados que las capturas a analizar son de tráfico NATeado (capturados desde la interfaz WAN de un *gateway* NAT), es esencial conocer en profundidad los estándar del mecanismo de NAT. Luego, van a pasar a detallarse las estrategias que se pueden implementar en este escenario tanto para la detección de dispositivos como para la detección de servicios. Un conjuntos reducido de estas estrategias van a ser las implementadas en el sistema desarrollado. Finalmente, se describen el alcance, las estrategias elegidas y las limitaciones que van a ser tenidas en cuenta para la implementación del sistema (cuyo detalle se encuentra en la sección siguiente).

Nuevamente, es necesario mencionar que **toda la información que sigue en esta sección (comportamiento de NAT, estrategias para detección de dispositivos y servicios, y alcance/limitaciones del sistema) son los resultados de la etapa de investigación del proyecto**, etapa cuya metodología y actividades fueron descritas en la sección anterior.

Comportamiento de NAT

Las capturas que se van a analizar en este trabajo corresponden a tráfico de una red privada que se encuentra NATeado debido a que el lugar de captura es la interfaz WAN del *gateway* que realiza el proceso de NAT. **Es crucial para el trabajo conocer en profundidad el comportamiento que tiene NAT en lo que respecta a aquellos aspectos que influyen en el tráfico de paquetes de la red privada**. Este comportamiento es el que va a permitir la implementación de ciertas estrategias en el sistema.

El mecanismo de NAT se encuentra estandarizado en varios documentos RFC. Todos los *routers* que funcionen como *gateway* de una red privada que realicen NAT para el acceso a Internet **deberían** implementar correctamente lo establecido en estos documentos. Para este trabajo, los documentos RFC sobre NAT que fueron estudiados en esta etapa de investigación fueron:

- RFC 2663: Terminología y comportamiento de NAT.
- RFC 3022: NAT Tradicional (el que se manifiesta en las redes que se estudian en este trabajo).
- RFC 5382: Requisitos de NAT para TCP.
- RFC 4787: Requisitos de NAT para UDP

A continuación se listan aquellas características de NAT que son importantes mencionar y explicar para este trabajo, es decir, que tienen algún tipo de influencia en la manera en que se refleja el tráfico de la red privada en la captura. Cada característica está referenciada con la sección y el documento RFC de donde fue extraída.

[RFC 2663 - Sección 4.1] Cambios en los paquetes

El mecanismo de NAT solo puede realizar modificaciones en los campos origen/destino y *checksum* en los *headers* de paquetes IP, TCP, UDP, ICMP.

[RFC 2663 - Sección 7.0] *Payload* de los paquetes

Está garantizado que los dispositivos que realizan NAT no van a examinar ni alterar el *payload* (es decir, el contenido de datos) de ningún paquete. Por esta razón, algunos protocolos que poseen información de las direcciones IP o puertos dentro de su *payload* (por ejemplo, IPSec) no son compatibles con NAT.

[RFC 2663 - Sección 4.1] Tipos de NAT Tradicional

Existe el ***Basic NAT* y el *NAPT.***

El primero (*Basic NAT*) es una simple traducción de direcciones IP. Una vez que un la dirección IP privada de un *host* es traducida a una dirección IP pública, entonces en todos los próximos paquetes de ese mismo *host*, la dirección IP será traducida bajo la misma

dirección IP pública. Esta opción sirve cuando hay igual o mayor cantidad de dirección IP públicas que direcciones IP privadas dentro de la red LAN. Los puertos no se modifican.

El segundo tipo (*NAPT*, por sus iniciales *Network Address and Port Translation*), tiene lugar cuando las direcciones IP públicas disponibles son menores que las direcciones IP de la red privada. En este caso, las modificaciones llegan hasta los puertos. Entonces, las traducciones se realizan no entre direcciones IP sino entre tuplas (dirección IP, puerto). Esta técnica va a ser la utilizada en las redes estudiadas en este trabajo, pues en este escenario solo existe una dirección IP pública (la de la interfaz WAN del *gateway/router*).

[RFC 3022 - Sección 4.1] Modificaciones

Las modificaciones tienen lugar en:

- Dirección IP origen: para paquetes salientes de la red privada.
- Dirección IP destino: para paquetes entrantes a la red privada.
- Puerto origen: para paquetes salientes de la red privada cuando hay NAPT.
- Puerto destino: para paquetes entrantes a la red privada cuando hay NAPT.
- *Checksum* de los paquetes IP, UDP, TCP: el *checksum* es el resultado de un cálculo matemático de todos los *headers* del paquete. Al modificar valores de algunos *headers*, es necesario recalcular el *checksum*.

[RFC 5382 - Sección 4.1] Reutilización de conexiones TCP para un mismo *host*

Un dispositivo NAT debe respetar lo que se conoce como "Endpoint-Independent Mapping". Este comportamiento expresa lo siguiente:

“Un *host* de la red privada origina una conexión desde la tupla [X:x] (que representa dirección IP:puerto) hacia [Y1:y1] (siendo Y1 una dirección IP pública). El dispositivo NAT lo mapea a la tupla [X1:x1], siendo X1 la dirección IP pública del dispositivo NAT. Luego, ese mismo *host* inicia desde mismo [X:x] una conexión hacia [Y2:y2], recibiendo el mapeo [X2:x2] en el dispositivo NAT. Entonces, **para que el dispositivo NAT tenga comportamiento ‘Endpoint- Independent Mapping’, [X1:x1] debe ser igual a [X2:x2] para cualquier valor de [Y2:y2]**”

[RFC 5382 - Sección 7.1] Asignación y reutilización de puertos para sesiones TCP

Se supone la siguiente situación:

Un *host* de la red privada origina una conexión desde la tupla [X:x] (que representa [dirección IP:puerto]) hacia [Y:y] (siendo Y una dirección IP pública). El dispositivo NAT lo mapea a la tupla [Z:z]. Si ahora otro *host* desde [X2:x1] quiere comunicarse con [Y:y], entonces **el dispositivo NAT no puede reutilizar el mismo mapeo [Z:z]**, sino que tiene que generar una conexión nueva a partir de un mapeo nuevo.

Bajo este comportamiento de evitar reutilizar los mapeos cuando dos *hosts* se comunican con la misma tupla [dirección IP:puerto], se dice que **el dispositivo NAT no tiene un comportamiento de "Port Overloading" en lo que respecta a la asignación de puertos.** La explicación de este comportamiento es bastante lógica, pues si dos *hosts* comparten el mismo mapeo para comunicarse con la misma tupla [dirección IP:puerto] pública, entonces cuando llega un paquete entrante de esta tupla, resultaría confuso para el dispositivo NAT saber a cuál de los dos *hosts* estaba destinado.

[RFC 4787 - Sección 4.2.1] Asignación de puertos en colisión

El comportamiento de "Port Overloading" surge a partir de que los dispositivos NAT tienden a mantener en el puerto mapeado el mismo puerto del *host* de la red privada. Por eso, si dos *hosts* se quieren comunicar con [Y:y] desde el mismo puerto, por ejemplo, 1000, ocurre la colisión pues NAT no puede reutilizar el mismo mapeo para *hosts* distintos. En este caso, el dispositivo NAT debe tener elegir otro puerto para alguno de estos *hosts*. Este nuevo puerto debe estar entre (0 y 1023) si el puerto del *host* esta en este rango, y sino entre (10-24 y 65535).

Acerca de otras características

Existen muchas otras características interesantes que no fueron incluidas en este listado (como por ejemplo políticas de cierre de sesiones TCP, manejo de comunicaciones a través del dispositivo NAT de dos *hosts* de la red privada o manejo de paquetes ICMP) porque no tenían ningún tipo de relación con el escenario de estudio de este trabajo o no tenían influencia sobre los paquetes que se iban a analizar en las capturas.

Estrategias para detectar dispositivos y aplicaciones

Campo ID del datagrama IP

La siguiente observación fue extraída del *paper* “Source attribution for network address translated forensic captures” de Cohen, publicado en 2009. Este artículo académico es el único que plantea exactamente el mismo escenario que se plantea en este trabajo, es decir, el análisis de tráfico *NATeado* para sacar información acerca de la red privada que se encuentra detrás del *gateway*. En este caso en particular, su objetivo es poder identificar la cantidad de dispositivos que se encuentran en una red.

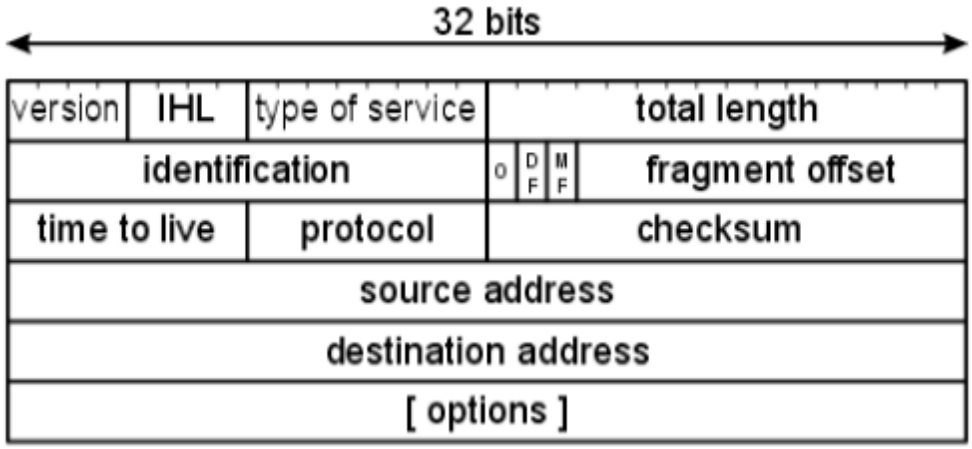


Fig 18. “Header” de un datagrama IP.

Cada datagrama IP cuenta con un *header*, o encabezado, que posee información propia del paquete, como por ejemplo, la dirección IP origen y la dirección IP del destino. Como se puede observar en la figura anterior (Fig 18), existe un campo “*identification*”, o ID, de 16 bits. Este campo es un número único por cada datagrama que se envía.

La funcionalidad de este campo es sencilla: el datagrama al viajar por Internet va atravesando distintos *routers*. Pueden existir *routers* en su camino hacia el destino que fragmenten ese datagrama en varios datagramas menores, enviando cada (sub)datagrama por una ruta distinta. Todos los fragmentos (o sub-datagramas) poseen el mismo valor en el campo ID, para que el *host* destino pueda ensamblar los fragmentos en los datagramas originales.

Ahora bien, el *paper* parte del supuesto que los sistemas operativos utilizan números consecutivos para colocar en el campo ID en cada datagrama que se envía. Entonces, por ejemplo, un dispositivo con un sistema operativo puede enviar datagramas con valores ID de 200, 201, 202, etc. mientras que otro dispositivo puede enviar datagramas con valores 1400, 1401, 1402, etc. Es decir, van a ser todos valores incrementales para un mismo dispositivo.

Teniendo en cuenta esta información, este artículo académico propone una estrategia bastante clara: si se analizan todos los campos ID de todos los datagramas presentes en la captura de tráfico, entonces es posible encontrar secuencias incrementales de valores que representan a un dispositivo. **Encontrando todas las secuencias incrementales, entonces se puede conocer el número mínimo de dispositivos activos que pertenecen a la red** (es un número mínimo pues puede ocurrir que varios dispositivos tengan los mismos números en la secuencia).

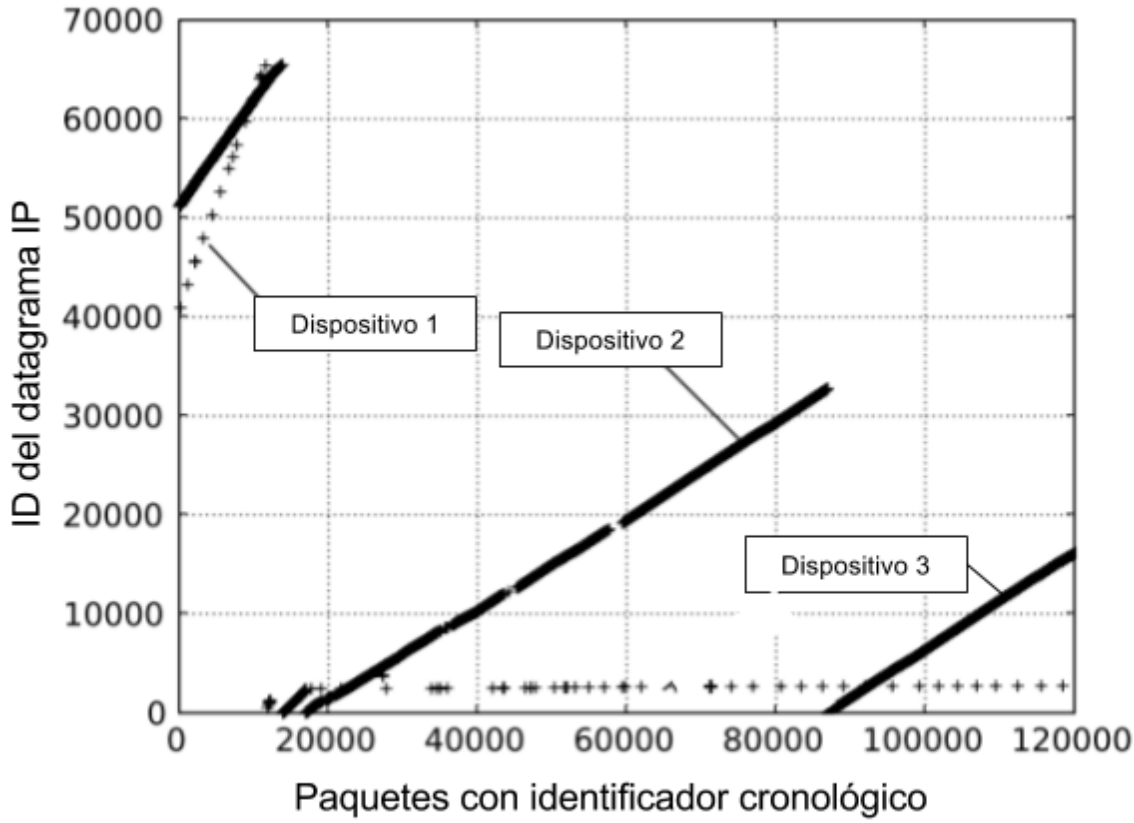


Fig 19. Imagen extraída del paper. Se puede observar que hay 3 secuencias incrementales de números ID.

En la figura anterior (Fig 19), se grafican a los ID de los datagramas IP para todos los paquetes presentes en la captura. En el eje X, se representan a los paquetes asociados a un número que se relaciona con el orden cronológico de aparición en la captura (el paquete 0 fue el primer paquete en la captura, mientras que el 120000 fue el último). En el eje Y, aparecen los ID de los datagramas IP de cada paquete. Resulta muy clara la formación de rectas en el gráfico, que representan las secuencias incrementales de números en el campo ID. Cada una de estas rectas representa al menos un dispositivo activo en la red.

Sabiendo esto, entonces sería posible crear un sistema que considere este comportamiento y se encargue de ir encontrando estas secuencias incrementales para tener un número mínimo de dispositivo que pertenecen a la red. Sin embargo, **esta**

estrategia no fue tenida en cuenta para el sistema que se desarrolló pues el supuesto de secuencias incrementales para la generacion de numeros a colocar en el campo ID de los datagramas IP no se respeta en la totalidad de los sistemas operativos. Es más, muchos de los sistemas operativos modernos evitan este comportamiento como una medida de seguridad en contra de ataques que explotan esta característica, como por ejemplo, un "*Idle scan*" que sirve para hacer un *scanneo* de puertos disponibles en una computadora.

Tanto las últimas versiones del kernel de Linux (en el cual está basado Android) como las de Solaris y OpenBSD ya poseen algoritmos de generación de números pseudoaleatorios para el campo ID de los datagramas IP. **Esta estrategia solo funcionaria con pocos sistemas operativos y tendría "fecha de vencimiento", pues tarde o temprano todos los sistemas operativos dejaran las secuencias incrementales.** Es por esta razón que no se utilizó esta estrategia para el desarrollo del sistema.

User Agent Fingerprinting por HTTP plano

Luego de toda la etapa de investigación, había un concepto que predominaba: **los paquetes HTTP son los que proveen mayor información tanto del dispositivo que los genera como del servicio que están consumiendo.** El hecho que a partir de los mensajes HTTP (en especial los *requests*) se pueda revelar información clave para el sistema a desarrollar como la marca y modelo de los dispositivos, su sistema operativo y su versión, o la aplicación (por ejemplo, *browsers*) que está consumiendo; posee una explicación y tiene que ver con el concepto de "*device fingerprinting"*.

El "*device fingerprinting"* es una disciplina que tiene como objetivo recolectar información acerca un dispositivo para su identificación (el concepto viene de *fingerprint*, o huella digital). **Esta identificación de dispositivos se realiza principalmente a partir del campo "*user agent"* de los mensajes *request* del protocolo HTTP.**

Acerca de los *User Agents*

El *user agent* es, por definición, un programa que funciona como cliente en un protocolo de red. El ejemplo más claro de *user agents* son los navegadores web, o *web browsers* (Chrome, Safari, Firefox, etc.), pero también pueden ser *user agents* aplicaciones móviles que consumen servicios en la red o lo que se conoce como *web crawlers* (programas que inspeccionan las páginas Web con el motivo de, por ejemplo, mantener la base de datos de un motor de búsquedas).

Breve referencia a HTTP

Como ya se explica en el apartado "Marco Teórico" de la primera sección del informe, HTTP (*HyperText Transfer Protocol*) es un protocolo de la capa de aplicación. Este protocolo involucra a dos partes, un cliente y un servidor. El cliente es el que solicita un recurso a través de un mensaje *request* y el servidor es el que brinda el recurso a través de un mensaje *response.*

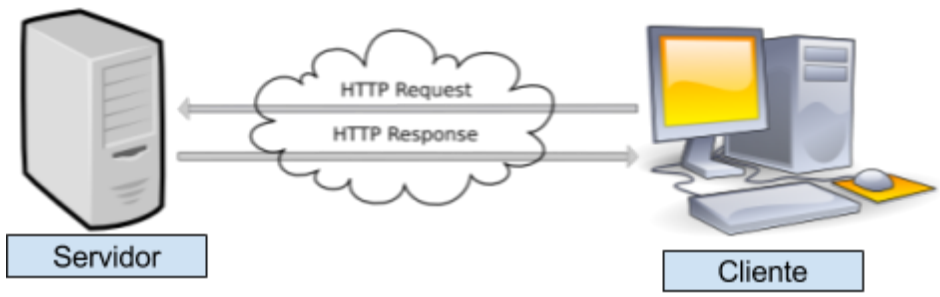


Fig 20. Flujo HTTP.

El documento RFC 2616, que es donde está estandarizado el protocolo HTTP, establece la sintaxis para los *request* y *response*. Este estándar establece que el *request*

está compuesto, en parte, por los *headers*, que son campos del tipo *"key: value"* con distinta información. Uno de esos *headers* es el *user agent*, con la sintaxis "User-Agent: …" (por ejemplo, *"User-Agent: CERN-LineMode/2.15 libwww/2.17b3"*). Según el RFC, **este campo es obligatorio y sirve para realizar estadísticas, detectar violaciones de protocolos y automatizar el reconocimiento de los *user agents*.**

Realizar "*device fingerprinting*" con el header "*User-Agent*"

Hoy en día, el header "*User-Agent*" se utiliza con el objetivo de realizar *device fingerprinting.* Los servidores (aquellos que reciben los *requests* y envían los *response*) buscan reconocer características del dispositivo para entregar los recursos adecuados.

El ejemplo más claro de esto es el que se describe a continuación. Cuando un servidor Web (o *Web server*) recibe un *request* solicitando el recurso "lanacion.com/ultimas-noticias.html", observando el header *User Agent* del *request* puede deducir si se trata de un navegador de una computadora o de un *smart-phone*. En el caso de una computadora, va a entregar el documento *html* solicitado con una versión que sea visible correctamente en una navegador de una computadora. En el caso de que sea un *smart-phone*, va a entregar una versión del documento *html* que sea adecuado visualmente para una pantalla reducida.

Fig 21. Versión HTML para un navegador de escritorio y versión HTML para un navegador de smartphone.

Utilidad del *User-Agent* para el sistema

El comportamiento de revelar información acerca del dispositivo a través del header *User-Agent* va a ser el motor del sistema a desarrollar en lo que respecta a:

- **Identificación de dispositivos**: poder caracterizar y distinguir dispositivos dentro de la red, con información acerca de la marca, modelo, sistemas operativos y versión, numero de *build,* etc.
- **Identificación de aplicaciones de dispositivos**: distinguir y caracterizar las aplicaciones que están vinculadas a cada dispositivo y que son las que realmente consumen los recursos HTTP. Por ejemplo, para cada dispositivo los navegadores web que posee y su versión, las aplicaciones instaladas y sus versiones, etc.

Es importante remarcar que **no se ha encontrado en toda la etapa de investigación ningún otro "lugar" que no sea el header *User-Agent* de los *request* HTTP a partir del cual se pueda extraer esta información.** Resulta importante también remarcar que esta información está disponible en la captura siempre y cuando sea tráfico HTTP plano, es decir, sin encriptar. La versión segura de HTTP, conocida como HTTPS, no es inspeccionable en las capturas de tráfico con las cuales se trabaja, pues su información se encuentra encriptada bajo el protocolo *TLS*. En estas capturas, solo se pueden observar paquetes TLS, cuyo contenido (que puede ser por ejemplo un request HTTP) no es posible interpretar.

A continuación se presentan algunos ejemplos de *requests* que permiten apreciar la importancia de la información que proveen en la búsqueda de los dispositivos que se encuentran detrás del *gateway* que realiza NAT.

```
GET / HTTP/1.1
Host: triamax.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MOB30M) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/51.0.2704.81 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: __utma=226398047.666483624.1406516500.1410136613.1412864694.10; _ga=GA1.2.666483624.1406516500
```

Fig 22. El User Agent provee información acerca de un Nexus 5 con Android 6.0.1 que posee un navegador Chrome version 51.0.2704.81

```
GET /generate_204 HTTP/1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 6.0.1; MotoG3 Build/MOB30Z)
Host: connectivitycheck.gstatic.com
Connection: Keep-Alive
Accept-Encoding: gzip
```

Fig 23. El User Agent provee información acerca de un Motorola G3 con Android 6.0.1

```
GET /d8ffd/de023cdb169171ffeeb10f6bb7768640196d8ffd.webp HTTP/1.1
Host: art-2.nflximg.net
Accept: */*
Accept-Language: en;q=1
Connection: keep-alive
Accept-Encoding: gzip, deflate
User-Agent: Argo/8.8.0 (iPad; iOS 8.0; Scale/2.00)
```

Fig 24. El User Agent provee información acerca de un iPad iOS 8.0

```
GET / HTTP/1.1
Host: lanacion.com.ar
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Fig 25. El User Agent provee información acerca de un dispositivo con Windows 10 con un browser Firefox versión 48.0

Distinción de dispositivos por *Handshake* TLS/SSL

La siguiente observación tiene que ver con información disponible en el proceso de *handshake* del protocolo TLS (o SSL) que podría servir para implementar una estrategia para conocer la cantidad mínima de dispositivos en la red. Es decir, es una estrategia que tendría el mismo objetivo que la estrategia descrita anteriormente del campo ID en los datagramas IP.

Acerca del TLS/SSL y el *handshake*

TLS (Transport Layer Security) y su antecesor SSL (Secure Socket Layer) son protocolos criptográficos que tienen la funcionalidad de proveer una comunicación segura a través de una red no segura entre un cliente y un servidor. Se habla de TLS/SSL como el mismo protocolo por un tema de versiones, pues a partir de SSL 3.0 se empezó a denominar TLS.

Tanto el protocolo mismo como los métodos de encriptación que utiliza tienen una complejidad y una profundidad que excede a este apartado. Lo que es importante saber de este protocolo es que cuenta con un procedimiento previo al intercambio de información donde se establecen cuales van a ser las reglas para la comunicación (las reglas son los algoritmos de encriptación, de *hashing*, etc.). Este procedimiento se conoce como *handshake* y consiste en una serie de mensajes entre cliente y servidor. En la figura siguiente (Fig 26) aparece un diagrama de los pasos que componen el *handshake*.

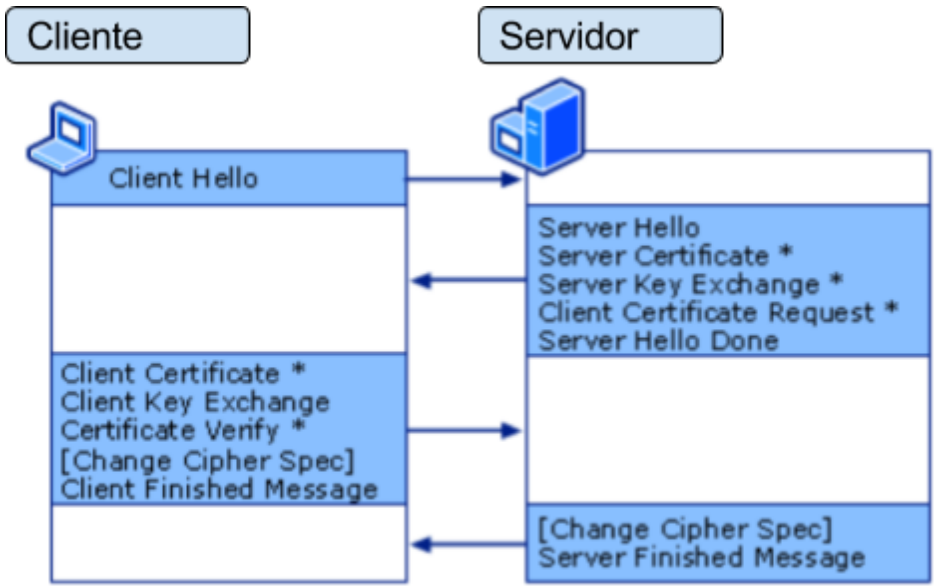


Fig 26. Flujo del handsharke TLS.

Acerca del *ClientHello*

Como se puede observar en la figura anterior, el primer mensaje del *handshake* que envía el cliente es el *ClientHello*. Este mensaje contiene principalmente la siguiente información acerca del cliente:

- La versión más alta de protocolo TLS que soporta
- Un número *random*
- ***Cipher Suite:*** **una lista de métodos de cifrado soportados**
- ***Compression Methods:*** **una lista con los métodos de compresión soportados**

El objetivo de enviar el *Cipher Suite* y los *Compression Methods* sirven para negociar con el servidor los algoritmos de cifrado y compresión a usar en la comunicación. El cliente, con esta información, está comunicando sobre aquellos algoritmos que soporta. Con estas listas, el servidor no va a sugerir (en mensajes siguientes del *handshake*) aquellos algoritmos que el cliente no soporta, pues debe elegirlos de la lista.

Utilidad del *ClientHello* para la distinción de dispositivos y/o aplicaciones

Como se explica anteriormente, el cliente en el *ClientHello* expone los métodos de cifrado y compresión que soporta. Ahora bien, el soporte para métodos de cifrado y compresión lo ofrece:

- El sistema operativo. Cada sistema operativo va a poseer soporte para determinados algoritmos de cifrado/compresión. Al mismo tiempo, las aplicaciones que usan los *drivers* de comunicación del sistema operativo, van a contar con los algoritmos con los que cuenta el sistema operativo.

- Una aplicación en particular. Una aplicación puede no consumir los algoritmos disponibles en el sistema operativo e implementar otros algoritmos. Por ejemplo, *web browsers* o aplicaciones de “gigantes” como Facebook, Twitter, etc. muy probablemente posean algoritmos de cifrado/compresión distintos a los que provee el sistema operativo.

Este comportamiento puede utilizarse para distinguir dentro de una red dispositivos y/o aplicaciones. Un dispositivo cuyo sistema operativo está estableciendo comunicaciones con servidores remotos (por ejemplo, para consultar actualizaciones disponibles) va a presentar en todos los *ClientHello* los mismos algoritmos soportados (es decir, aquellos que soporta ese sistema operativo). Lo que es aún más, si hay aplicaciones que utilizan los *drivers* de comunicación del sistema operativo de este dispositivo en cuestión para establecer conexiones TLS, también van a presentar los mismos algoritmos en el *ClientHello*. En el momento que aparezca un *ClientHello* con otro *Cipher Suite* u otra lista de *Compression Methods*, entonces se puede deducir que ocurre alguna situación de las siguientes:

- Existe un dispositivo distinto que posee soporte para otros algoritmos de cifrado/compresión
- El mismo dispositivo empezó a utilizar una aplicación que posee sus propios algoritmos de cifrado/compresión.

Esta estrategia sirve únicamente para distinguir y saber que hay dispositivos/aplicaciones distintos en una red cuando aparecen distintas combinaciones de *Cipher Suite* y *Compression Methods*. Sin embargo, esta estrategia no ayuda en el caso de que hayan sistemas operativos distintos que soportan los mismos algoritmos, o que hayan varias aplicaciones que soportan los mismos algoritmos, pues no van a ser distinguibles.

Considerando que hoy en día están relativamente estandarizados los algoritmos que deberían soportar los sistemas operativos como las aplicaciones, entonces la mayoría de las veces ocurre la situación en la que esta estrategia no funciona (todos presentan los mismos algoritmos que se encuentran "estandarizados").

Por ejemplo, sea el caso en el que en la captura se detectaron dos combinaciones de algoritmos de cifrado/compresión distintos. La combinación A soporta C y D como algoritmos de cifrado y compresión respectivamente, mientras que la combinación B soporta C' y D' como algoritmos de cifrado y compresión, con C distinto a C' y D distintos a D'. Esta estrategia informa que en la red podría haber al menos 2 dispositivos distintos, o 1 dispositivo con 1 aplicación con algoritmos propios, o 2 aplicaciones distintas con algoritmos propios en el mismo dispositivo. Sin embargo, el escenario real podrían ser 10 dispositivos, donde 6 poseen sistemas operativos con soporte para los mismos algoritmos y 4 de ellos para otra combinación de algoritmos.

Como se puede observar, esta estrategia no tiene tanta utilidad y es por esa razón que no se ha tenido en cuenta para el sistema que se desarrolló.

Otras estrategias HTTP

Existen dos estrategias más, que aparecen en el mismo artículo académico donde se explica la estrategia del campo ID del datagrama IP, que explotan ciertas caracteristicas del protocolo HTTP. Estas estrategias tienen un campo de aplicación menor y resultan menos eficientes en la detección de dispositivos. A continuación, se menciona brevemente de qué se tratan.

Header "Referer"

El *header* conocido como "*referer*" o "*referrer*" se utiliza cuando una página Web solicita un recurso específico (como por ejemplo una imagen para mostrar o un código fuente *javascript* para efectuar animaciones). Entonces, esta página (a través de su URI) aparece como *"referer"* en el *request* de este recurso solicitado.

Fig 27. Ejemplo de request con header referer. La URI que inicia en "dvdjcdjcd.eu" solicito un recurso en "can.berryelectricalusa.org"

Teniendo esto en cuenta, se puede establecer que cuando se detecta que de un dispositivo se inició una comunicación HTTP donde aparece este *header*, entonces, es casi seguro que ese mismo dispositivo fue el que intercambió mensajes HTTP con el *host* de la URL original.

Cookies

Una cookie es una secuencia de bits que aparecen para poder mantener sesiones de usuario ya que el protocolo HTTP es *stateless* (no mantiene estado entre *requests*). La *cookie* es dada al cliente cuando inicia la sesión en alguna plataforma y la debe presentar en cualquier *request* futuro hacia esa plataforma para que sea identificado. Las *cookies* son lo suficientemente *random* como para no poder ser adivinadas o repetidas. Teniendo en cuenta esto, se puede deducir que varios *requests* HTTP que poseen la misma *cookie* provienen del mismo dispositivo.

Esta estrategia se encuentra ciertamente obsoleta pues las plataformas donde existen sesiones de usuario están migrando su tecnología hacia HTTPS.

Estrategias para detectar servicios consumidos

Parte de la detección de comportamiento de patrones en una red consiste en **el descubrimiento de los servicios que se están consumiendo.** En este contexto, **un servicio es un recursos disponible para que un usuario consuma**. Algunos ejemplos de servicios pueden ser:

- Un diario digital.
- Un portal de streaming de películas, como *Netflix*.
- Un programa de streaming de música, como por ejemplo Spotify.
- Una red social, como Facebook o Twitter por ejemplo

Hasta aquí se habían detallado estrategias para la detección de dispositivos y de aplicaciones, pero no se había contemplado el tema de los servicios.

Una parte importante para poder realizar un análisis completo de los patrones de comportamiento de una red tiene que ver con "**que" se consumió en la red**. No tienen la misma calidad un análisis de patrones de comportamiento que reporta "*en la red se encontró actividad el dia Lunes de 18:30 hs a 21:00 hs de un dispositivo iPhone 5 con iOS 10.1*" y otro que afirma "*en la red se encontró actividad de un dispositivo iPhone 5 con iOS 10.1, el cual de 18:30 a 19:30 utilizó la plataforma Netflix y luego de 19:30 a 21:00 estuvo consumiendo con un navegador los diarios digitales de La Nación y Página 12; al mismo tiempo que intercambiaba mensajes utilizando WhatsApp y reproduce música con la plataforma Spotify"*. Con este ejemplo se quiere demostrar que **la detección de servicios consumidos es un complemento esencial para poder realizar un análisis completo e intensivo de los patrones de comportamiento de una red.**

A continuación se detallan diversas estrategias que se pueden implementar para poder detectar los servicios que se consumen. De acuerdo a las investigaciones que se realizaron acerca de este tema, algunas resultan más efectivas que otras. Algunas de estas estrategias fueron utilizadas para el sistema que se desarrolló, cuyos detalles de implementación se encuentran en la sección siguiente.

Detección por IP

Los servicios que se consumen pueden ser identificados por la dirección IP de la cual provienen. Como se explica en el apartado de "Marco Teórico", los servicios se ofrecen (o *hostean*) en un *host* público de Internet que está identificado con una dirección IP. Si se conoce la dirección IP del *host* que provee un determinado servicio; entonces al detectar en la captura comunicación con esa dirección IP se va a poder deducir que un cliente de la red está consumiendo ese servicio en cuestión.

Sin embargo, la efectividad de poder mapear el servicio con direccion IP hoy en dia es muy baja. Esto se debe a dos razones principalmente:

- **Las direcciones IP cambian**. Esto quiere decir que una servicio bajo la dirección IP “X” puede estar en otro momento bajo la dirección IP “Y”. Esto puede ser por una política de asignación de direcciones IP dinámicas por parte de los proveedores de direcciones IP públicas o por políticas propias de los administradores de servicios de cambiar el conjunto de direcciones IP que destinan a un servicio. **Esta razón es la principal justificación de la aparición de la arquitectura DNS para la ubicación de *hosts* por nombre**.

- **Bajo una dirección IP pueden existir múltiples servicios**. Un *host* público es, al fin y al cabo, una computadora en *Internet*. Esta computadora puede no estar atada a un servicio único. En este escenario, con solo detectar una comunicación con este *host*, no se puede deducir el servicio que se consume, pues puede estar consumiendo cualquiera de los *N* servicios que se ofrecen en ese *host*.

Los únicos casos en que esta técnica resulta efectiva son cuando se puede asegurar que **el conjunto de direcciones IP es fijo para ese servicio y que los *hosts* que se encuentran en esas direcciones IP son dedicados a ese servicio.**

Detección por dominio

Como también se explica en el “Marco Teórico”, la ubicación de un *host* se puede conseguir a través de un *nombre*. Este *nombre* es denominado dominio, o en inglés, *domain name*, y consiste en una secuencia de caracteres “*case-insensitive”* (no diferencia mayusculas de minusculas) que representan a una (o varias) direcciones IP. La funcionalidad principal de la infraestructura de DNS, conocida también como “resolución de nombres”, es la de mapear una dirección IP a partir de un dominio. Los *domain names* se encuentran estandarizados en el RFC 1034.

Ahora bien, como se puede observar en la figura 18 que representa la estructura de un datagrama IP, el destino se identifica con la dirección IP del *host* y no por su dominio. Existen diversas estrategias que, a partir de una captura y sabiendo que los datagramas solo manejan direcciones IP, pueden conocer el dominio del *host* con el cual se está comunicando un cliente. A continuación se mencionan dichas estrategias.

DNS Reverso

Aparte de las búsquedas DNS tradicionales, que obtienen la dirección IP a partir del dominio, existe la **búsqueda de DNS reverso**, la cual, como resulta evidente, puede **obtener el dominio a partir de la dirección IP.** De esta manera, sabiendo las dirección IP (presente en los datagramas IP), se podría realizar una búsqueda de este tipo para obtener el nombre del dominio.

Sin embargo, la infraestructura de DNS reverso no es tan exacta como debería. En la mayoría de los casos, el mapeo no es el adecuado. Esto quiere decir que si a través de una búsqueda DNS tradicional el dominio A mapea a la dirección IP ‘X’, la búsqueda por DNS reverso de la dirección ‘X’ muy probablemente mapee a un nombre B distinto de A. Las razones por la cuales ocurre esto son:

- **Mal configuración de los servidores DNS.** Los dueños de los dominios deben configurar servidores DNS, también conocidos como servidores de nombres, que se encargan de realizar los mapeos de los dominios en cuestión. Generalmente, al no ser esencial la búsqueda de DNS reverso (la mayor utilidad es para chequeos *anti-spam* de *mails*), la configuración para esta funcionalidad se encuentra ausente o mal realizada.

- **Una IP bajo muchos dominios**. Muchos dominio puede mapear a una misma dirección IP. Esto ocurre por ejemplo cuando alguien monta varios servicios en un mismo *host* que los atiende. En este caso, la búsqueda de DNS reverso puede devolver cualquiera de los N dominios registrados en esa dirección IP, el cual puede no coincidir con el buscado originalmente.

Caché de búsquedas DNS

Como se explica anteriormente, la información que necesita un *host* para comunicarse con otro es la dirección IP, y no el dominio. Entonces, si un *host* cliente conoce el dominio del *host* destino con el cual se quiere comunicar, para poder establecer una comunicación debe hacer una búsqueda DNS primero para averiguar la dirección IP. Estas consultas se pueden detectar en las capturas de tráfico como un intercambio DNS de dos mensajes, una mensaje de búsqueda y un mensaje de respuesta. Este intercambio se da con un servidor de nombres, el cual el cliente tiene ya predeterminado (es decir, conoce su dirección IP de antemano).

Ahora bien, resulta muy útil tener una base de datos con las búsquedas DNS que se realizaron (lo que se denomina en este trabajo como *caché)* para que cuando se detecte una comunicación con una determinada dirección IP, entonces se consulte en el *caché* si esa dirección IP fue la respuesta de una consulta DNS. De ser así (y ser la única que dio esa dirección IP como respuesta), entonces se puede concluir que esa comunicación se establece indudablemente con el dominio que fue buscado por DNS anteriormente.

Header Host de los *request* HTTP

Antes se habló de la utilidad del *header User Agent* en los *request* HTTP. Existe un campo que es útil para poder deducir el dominio con el cual se está comunicando un cliente que realiza un *request* HTTP. Este *header* es el *Host*, el cual contiene literalmente el nombre del dominio del *host* que recibe el *request.*

Este *header* sirve para que el *server* que recibe el *request* sepa a qué dominio se refiere el recurso solicitado (a través del protocolo HTTP un cliente accede a un recurso, como por ejemplo “ultimas-noticias.html” o “video_2016.mp4”). De esta manera, si un *host* funciona como *web server* para distintos dominios, no se confunde en el recurso solicitado, pues viene especificado en el *header Host.*

La unica contra que tiene esta estrategia es que se puede utilizar únicamente con los paquetes que son *requests* HTTP planos (sin encriptar).

¿Dominio implica nombre de servicio?

Lo que se detallaron hasta este puntos fueron tres estrategias (DNS reverso, *cache* DNS y *header Host*) para poder obtener el dominio teniendo en cuenta que los datagramas IP que viajan entre *hosts* sólo poseen información de las direcciones IP. Pero, este apartado buscaba **encontrar estrategias para poder obtener información acerca de los servicios que se consumían en una red**. **Lo que se obtiene con estas estrategias son dominios, no servicios.**

Muchas veces, si es posible identificar a partir de un dominio el nombre de servicio que se está consumiendo. Por ejemplo, “www.lanacion.com”, “facebook.com” o “instagram.com” corresponden a La Nacion, Facebook o Instagram.

Sin embargo, muchas otras veces, el nombre del dominio no tiene absolutamente ninguna correlación con el nombre de servicio. Por ejemplo: “scdn.co” es el dominio asociado al servicio de streaming de musica de Spotify; o “me.com” es el dominio asociado al servicio de iCloud de Apple.

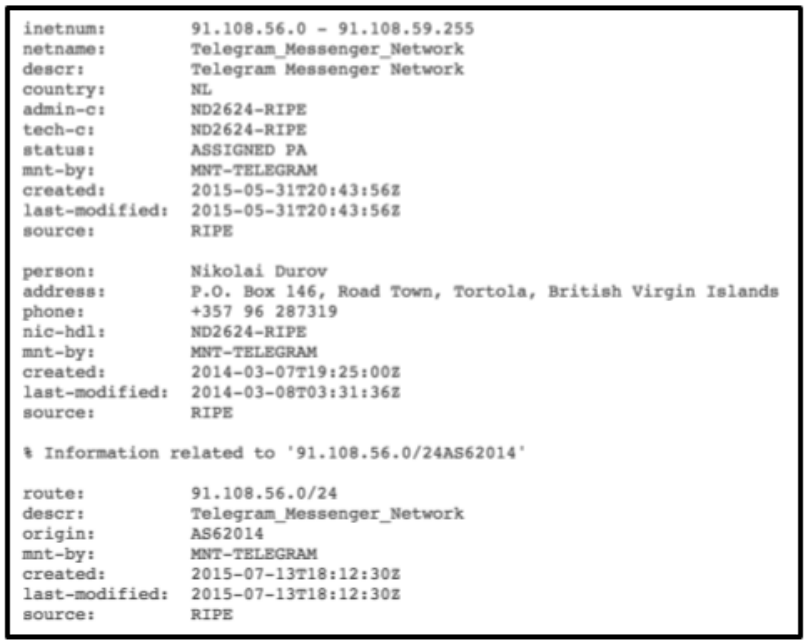
Teniendo esto en cuenta, cualquier estrategia de las descritas anteriormente que se implemente en el sistema para la detección de servicios consumidos va a tener que tener en cuenta este comportamiento, por lo cual **va a necesitan complementarse con una funcionalidad que permita encontrar el nombre del servicio a partir del dominio.**

Detección por propietario

Aparte de la detección de servicios por IP o por dominio, se puede intentar detectar el servicio consumido a partir del propietario de la dirección IP, utilizando el protocolo *Whois.*

“*Whois*” es un protocolo que permite conocer información acerca de los propietarios de dominios o de direcciones IP. Realizando una simple consulta a partir de una dirección IP, se puede conocer información acerca del propietario de esa dirección IP, como el nombre, el *mail* de contacto, dirección de contacto, etc.

Al igual que con el tema de los dominios, existen situaciones en las que es muy difícil poder deducir el servicio que se está consumiendo a partir de la información del propietario de la dirección IP con las que se está comunicando.



```
inetnum:        91.108.56.0 - 91.108.59.255
netname:        Telegram_Messenger_Network
descr:          Telegram Messenger Network
country:        NL
admin-c:        ND2624-RIPE
tech-c:         ND2624-RIPE
status:         ASSIGNED PA
mnt-by:         MNT-TELEGRAM
created:        2015-05-31T20:43:56Z
last-modified:  2015-05-31T20:43:56Z
source:         RIPE

person:         Nikolai Durov
address:        P.O. Box 146, Road Town, Tortola, British Virgin Islands
phone:          +357 96 287319
nic-hdl:        ND2624-RIPE
mnt-by:         MNT-TELEGRAM
created:        2014-03-07T19:25:00Z
last-modified:  2014-03-08T03:31:36Z
source:         RIPE

% Information related to '91.108.56.0/24AS62014'

route:          91.108.56.0/24
descr:          Telegram_Messenger_Network
origin:         AS62014
mnt-by:         MNT-TELEGRAM
created:        2015-07-13T18:12:30Z
last-modified:  2015-07-13T18:12:30Z
source:         RIPE
```

Fig 28. Consulta WHOIS a partir de una dirección IP del servicio de chat Telegram.

Breve referencia al alcance, estrategias y limitaciones del Sistema

Como se menciona en la sección de “Metodología”, **la última parte de la etapa de investigación consistió en la definición, a partir de los conocimientos adquiridos en esta etapa, del alcance y las limitaciones del sistema que se iba a desarrollar**. A continuación se detallan el alcance que va a tener el sistema y se mencionan cuáles de las estrategias explicadas anteriormente van a ser implementadas para lograr llegar a este alcance propuesto; como así también las limitaciones que se sabe que va a tener el sistema previamente al comienzo del desarrollo.

Con respecto al alcance, el sistema a desarrollar a partir del procesamiento de archivos de tráfico de una red privada capturado en la interfaz WAN de un *gateway* NAT debe poder:

- **Identificar posibles dispositivos activos en la red.** Cada uno identificado con una serie de características como puede ser la marca, el modelo, el *build*, el sistema operativo, la versión del sistema operativo, los *drivers* que utiliza, etc.
- **Identificar y asociar aplicaciones a los dispositivos encontrados.** Aplicaciones como navegadores web o *mobile apps* deben poder asociarse a dispositivos como así también caracterizarse por versión, nombre, etc.
- **Identificar servicios consumidos**. Conocer de cada servicio que direcciones IP estuvieron involucradas como así también dominios. En muchos casos el servicio debe poder relacionarse con la aplicación y por ende el dispositivo que lo consumió. Lo que es aún más, cada servicio debe estar caracterizado por el horario en el cual fue consumido y la cantidad de bytes que se intercambiaron en el consumo.

Todas estos aspectos van a poder revelar los eventos de la red que describan los patrones de comportamiento.

Para lograr este objetivo se van a utilizar las siguientes estrategias:

- ***“User Agent Fingerprinting* por HTTP plano”** para la detección de dispositivos.
- **“Detección por IP”, “Cache de búsquedas DNS” y “*header Host* de los *request* HTTP”** para la detección de servicios.

Las otras estrategias que se explicaron en esta sección no van a ser tenidas en cuenta para el desarrollo del sistema y quedan pendientes como futuras extensiones.

En cuanto a las limitaciones, se pueden identificar dos a esta altura:

1. **Dispositivos clones**. En el caso de que existan en la red varios dispositivos que sean exactamente iguales entre sí (modelo de hardware, versión del sistema operativo, build del sistema operativo, etc) resulta imposible poder diferenciarlos por el tráfico que generan en la captura obtenida de la interfaz WAN del *gateway* NAT. Caso ejemplo: 3 de estos dispositivos “clones” estaban navegando con el mismo navegador por 3 páginas web distintas. El tráfico es exactamente el mismo que 1 de esos dispositivos navegando en esas páginas en 3 pestañas de su navegador.

2. **Migración a HTTPS**. Como se explicó recientemente, la detección de dispositivos se da únicamente con la técnica de *fingerprinting* de *User Agents*. Es decir, depende de tráfico HTTP plano (sin encriptar). Cada vez es menos frecuente el tráfico HTTP plano. Este tráfico está migrando a su modo seguro, HTTPS; y como se explicó en secciones anteriores de este informe, este tráfico deja de ser inspeccionable en la captura.

Es importante mencionar que esta definición del alcance y de las limitaciones limitaciones se da únicamente a partir del conocimiento adquirido en la etapa de investigación. No contemplan aspectos que puedan surgir luego en la etapa de desarrollo, la cual va a estar detallada en la sección siguiente. Esto quiere decir que durante la etapa de desarrollo esta definición de alcance y limitaciones va a estar sujeta a cambios propios de la implementación (restricciones en la tecnología disponible, tiempo disponible para el desarrollo, etc). De ser así, los cambios van a figurar en la sección correspondiente.

Sección IV: **Implementación**

Introducción

Esta sección describe la implementación del sistema construido en la etapa de desarrollo. Como se explica anteriormente en la sección de “Metodología”, luego de la etapa de investigación tuvo lugar la etapa de desarrollo, la cual estuvo constituida por todas las actividades que se involucraron en la construcción del sistema de software.

Si bien el sistema desarrollado involucra dos componentes (el *core* y la *GUI*), esta sección (a menos que se lo explicite) va a tratar acerca del *core*, que constituye el *back-end* del sistema (es decir, toda la lógica y procesamiento).

Esta sección incluye detalles sobre la tecnología utilizada, sobre la arquitectura planteada, el flujo de información y los algoritmos creados para llevar a cabo las estrategias elegidas, sobre la performance y sobre el I/O (*input* y *output*).

SonarWAN

El sistema desarrollado tiene el nombre de “**SonarWAN**”. Este nombre, como se puede deducir fácilmente, es una combinación entre “Sonar” y “WAN”. Por un lado, el término WAN (Wide Area Network), como ya se explicó anteriormente en el Marco Teórico, hace referencia a aquellas redes que se extienden por zonas extensas geograficamente y que suelen tener también una cantidad alta de nodos más alta, como por ejemplo la red de un ISP (Internet Service Provider). Por el otro lado, un sonar (acrónimo de **So**und **Na**vigation And **R**anging) es una técnica de navegación que utiliza la propagación del sonido con **el objetivo de detectar objetos**. El objetivo del sistema desarrollado es detectar patrones de comportamiento de una red privada a partir del tráfico en WAN que genera la misma. **Similar a tener un sonar en una red WAN.**

En esta sección y en lo que reste del informe, **SonarWAN** va a ser utilizado como sinónimo del sistema desarrollado. Se recuerda que cuando se mencione al sistema o a **SonarWAN**, se va a estar referenciando al *core* del sistema, y no a la *GUI*, a menos que se explicite.

Repositorios

Para toda la etapa de desarrollo se utilizó la plataforma de GitHub, al ser la plataforma más famoso de desarrollo de proyectos *Open Source*. En dicha plataforma, SonarWAN se presenta como una organización, por ahora compuesta por los integrantes de este proyecto y por el tutor, la cual contiene los repositorios fuente tanto del *core* como de la *GUI*. En estos repositorios es posible ver los avances del sistema, leer el código fuente, reportar problemas en el funcionamiento del sistema, colaborar con el avance código, etc.

El link a la organizacion es github.com/SonarWAN.

Tecnología Utilizada

Siempre que se va a realizar una descripción detallada de una implementación, es esencial mencionar y justificar la tecnología utilizada. En particular en un sistema de software, como es este caso, la tecnología no solo incluye el lenguaje de programación, sino también el paradigma que se utilizó en este lenguaje, las librerías nativas que se utilizaron, las librerías externas, etc.

Python

En primer lugar, se justifica el lenguaje de programación, que como se puede anticipar, fue **Python**. El lenguaje de programación no solo impone la sintaxis con la cual se escribe código, sino también que existen otros factores que lo caracterizan y justifican su elección. Entre dicho factores, se puede identificar:

- **Velocidad**. Qué tan rápido es en ejecución. Esto depende de si es un lenguaje compilado o interpretado, a que lenguaje compila, etc.

- **Comunidad.** Tiene que ser un lenguaje que tenga una buena comunidad (en especial *Open Source)* de programadores. Esta característica permite que el lenguaje reciba actualizaciones y que existan múltiples librerías externas. Lo que es aún más, al ser SonarWAN un proyecto *Open Source*, el lenguaje de programación no debe imponer una barrera de entrada para diferentes programadores que quieran colaborar.

- **Documentación.** El lenguaje debe estar correctamente documentado, para poder hacer uso fácilmente de las herramientas (controles de flujo, funciones de la librería estándar, etc.) que provee.

- **Multiplataforma.** Es esencial que el lenguaje elegido sea multiplataforma, es decir, que pueda correr en cualquier computadora independientemente del sistema operativo que posee.

- **Simpleza.** Es un factor menor pero también importante y se refiere a que sea simple el desarrollo en este lenguaje. Esto se refiere a que sea fácil la instalación del lenguaje en cuestión, que provea buenos manejadores de librerías y versiones, que no dependa de entornos de desarrollo, etc.

Bajo estos factores, se decidió que la mejor opción sea **Python.** Se trata de un lenguaje de programación multiparadigma, ya que la programación puede ser orientada a objetos, puede ser imperativa o hasta también, funcional. Es un lenguaje interpretado cuyo tipado es dinámico y además es multiplataforma.. **Este lenguaje desde ya algunos años viene siendo uno de los lenguajes más utilizados por la comunidad de desarrolladores**. Se caracteriza por ser un lenguaje multipropósito, es decir, no es

específico para un área en particular (como lo es por ejemplo Javascript para desarrollo Web). Posee una de las comunidades de desarrollo más grandes, lo que permite que existan una variedad enorme de librerías, extensiones, frameworks, etc.

Una de los aspectos más fuertes de Python es que su código es fácil de interpretar. Esto quiere decir que cualquier programador va a poder fácilmente entender lo que realiza un código Python y va a poder fácilmente colaborar en el mismo. Este aspecto es crucial en los proyectos colaborativos de código abierto. La filosofía tras este aspecto de "fácil interpretación" se manifiesta en que debe existir una sola manera de realizar las cosas en Python siguiendo ciertas normas de estilo que estandarizan el código (lo que se conoce en la comunidad como *Pythonic Way*).

Al ser un lenguaje interpretado, la única contra que tiene este lenguaje es su *performance*, la cual es mucho más baja que en un lenguaje compilado (como lo es C o C++). Existen, sin embargo, varias implementaciones de Python (PyPy en particular) que pueden optimizar la velocidad. En este trabajo, se trabaja con la implementación más conocida de Python, llamado CPython, que está escrita en C. El hecho de que SonarWAN es un sistema que no necesita un poder de cálculo alto y también todos los aspectos positivos (vasta comunidad, multiplataforma, altamente conocido, etc) que tiene Python hacen de la "baja" performance un aspecto menor.

Por último, es necesario aclarar que la versión de Python utilizada es la 3. Existen dos grandes versiones de Python en uso actualmente, la 2 (versión anterior pero muy estable) y la 3 (versión última y también estable). Se eligió esta última pues es la recomendada por la comunidad para los proyectos nuevos. Si bien ambas versiones se mantienen estables y reciben actualizaciones, el objetivo es que a largo plazo prevalezca únicamente la última versión.

PyShark

El procesamiento que va a realizar el sistema va a ser a partir de la lectura y la inspección de los paquetes presentes en las capturas de tráfico. Por lo tanto, es necesario contar con un módulo en el sistema que sea el encargado de poder leer e inspeccionar secuencialmente los paquetes involucrados. Ante esta necesidad, se analizaron dos propuestas:

1. **Realizar un desarrollo propio de este módulo**. Es decir, desarrollar un componente que pueda tener esta funcionalidad. Para llegar a este objetivo, se iba a tener que estudiar cómo están estructurados los archivos de captura, o mismo aprender el manejo de interfaces en Python para poder proveer soporte analisis en vivo. Todos estos aspectos no tienen que ver con el objetivo del trabajo en sí y consumirían mucho tiempo de investigación/desarrollo.

2. **Utilizar una librería externa que se encargue de esta funcionalidad.** Consiguiendo una librería que satisfaga las necesidades y funcione correctamente ahorraría mucho tiempo de desarrollo.

Como se dijo anteriormente, una de las grandes ventajas que tiene Python es la comunidad de desarrolladores *Open Source* que posee, aspecto que permite la existencia de múltiples librerías *Open Source* externas (no estándar del lenguaje) que están disponibles públicamente para su uso. Por lo tanto, se optó por la segunda opción, es decir, utilizar una librería externa.

En el proceso de elección de esta librería en cuestión, se analizaron varias propuestas. Incluso se hicieron prototipos funcionales utilizando cada una de estas librerías para poder conocer cuál era la que mejor se adaptaba a SonarWAN.

Finalmente, se decidió por **PyShark**, una librería *Open Source* que se encuentra también en la plataforma *GitHub*. **Las razón principal por la cual se eligió esta librería es que es básicamente un *wrapper* de *TShark*.** Un *wrapper* es un programa cuyo objetivo es proveer una interfaz compatible para hacer uso de un segundo programa. En este caso, ese segundo programa es *TShark*, que es *Wireshark* pero sin una interfaz gráfica, es decir, con una interfaz de terminal. Es decir, **lo que realiza PyShark es proveer la capacidad de, a través de Python, hacer uso de *Wireshark*.**

Ya se dijo anteriormente que *Wireshark* es la herramienta más conocida y efectiva para análisis de tráfico. **Poder hacer uso desde SonarWAN de la tecnología de *Wireshark* es una enorme ventaja**, pues asegura **confianza** (los paquetes van a ser correctamente inspeccionados) y **estabilidad** (*Wireshark* está continuamente actualizándose y proveyendo soporte para los usuarios). Todas las otras librerías que se analizaron desarrollaban sus propios analizadores de tráfico y, por lo tanto, generaban cierta desconfianza en su correctitud. **Con esta librería, se está literalmente utilizando *Wireshark* desde Python.**

Lo que es aún más, **PyShark soporta captura en vivo**, y además en el caso de levantar un archivo de captura, no necesita cargarlo enteramente en memoria, sino que **trae a memoria únicamente los paquetes que va leyendo**. Esto último resulta en otra gran ventaja, considerando que los archivos convencionales de captura suelen tener varios GB de tamaño.

Input / Output

Es importante para realizar una caracterización de un sistema desarrollado detallar su *input* y su *output*. El *input* son aquellos datos que provee el usuario del sistema para ser analizados, procesados, etc. Luego del procesamiento, los resultados se encuentran en lo que se conoce como *output.*

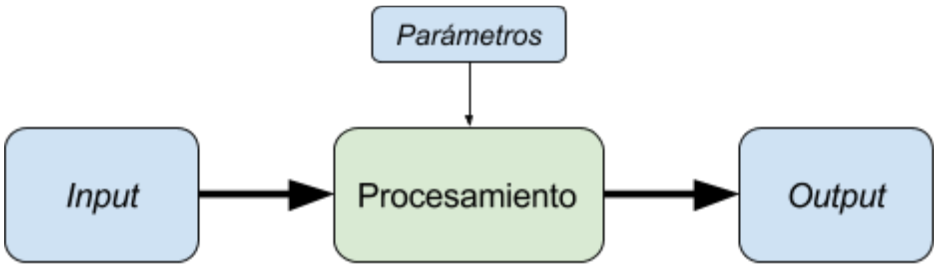


Fig 30. Flujo de un Sistema.

Input

En el caso de SonarWAN, **el *input* son archivos de capturas de tráfico** (el escenario en el que el *input* sea una captura en vivo queda pendiente como una futura extensión del sistema). SonarWAN recibe una lista de archivos en formato “pcap” o “pcapng” que representan las capturas de tráfico que el usuario posee sobre la red privada sobre la cual quiere conocer los patrones de comportamiento. Esos dos formatos compatibles con SonarWAN son el formato estándar para guardar paquetes de capturas de tráfico.

Ahora bien, una captura de tráfico de una red se realiza durante cierto periodo de tiempo (pueden ser horas, días, semanas, meses, etc.). Resulta poco probable que toda la captura esté guardada en un solo archivo (por tamaño, restricciones en el lugar de captura, corte de conexiones, interrupciones en el *sniffer*, etc), por eso SonarWAN permite que el usuario provea **una lista de archivos de capturas ordenada cronológicamente**. En este último escenario, pueden existir momentos de tiempo donde no existan capturas de tráfico, es decir, no se tenga registro de la actividad que tuvo la red. En estos casos, **SonarWAN va a procesar a todos los archivos como si fueran una única captura.**

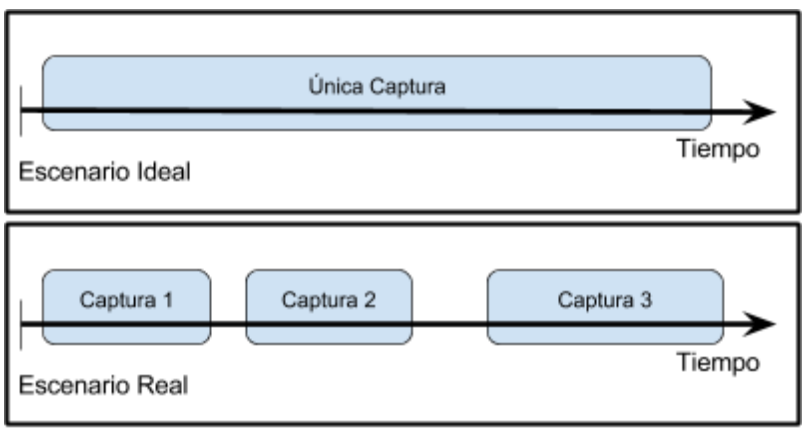


Fig 31. Escenarios de archivos de captura

Acerca de PCAP, “.pcap” y “.pcapng”

Suele haber confusión entre los términos que denotan estos formatos. **PCAP** (**P**acket **Cap**ture) es una API (interfaz de programación de aplicaciones) para capturar tráfico de red. Existen diversas implementaciones de PCAP, entre las que se destacan **libpcap** para dispositivos tipo *Unix*. Estas implementaciones sirve como motor para analizadores, por ejemplo, *Wireshark*.

Estas implementaciones soportan guardar capturas a un archivo y leer luego los paquetes de estos archivos guardados. Estos archivos pueden tener un formato “pcap” o “pcapng” (archivos que tienen extensión “.pcap” o “.pcapng” respectivamente). Este último formato forma parte de una nueva generación (su nombre significa justamente **pcap N**ext **G**eneration) que busca reemplazar a la versión anterior (permite almacenar mayor información y ser extensible). Este último formato pasó a ser la versión defecto para guardar las capturas en *Wireshark.*

Acerca de los parámetros que recibe

SonarWAN recibe **parámetros** por parte del usuario; los cuales también forman parte, de alguna manera, del *Input*. **Estos parámetros pueden ser agrupados en dos grupos: por un lado, aquellos que permiten configurar el *output*; y por otro lado, aquellos que tienen que ver con la extensibilidad del sistema y con la capacidad de aumentarle la base de conocimiento**. Estos últimos van a estar relacionados con un componente que va a ser explicado luego en esta sección y tienen que ver con la capacidad que tiene el usuario para extender la base de conocimiento. Todos los parámetros van a ser detallados en el manual de uso del sistema que aparece en el Anexo de este Informe.

Output

El *output* representa los datos que se obtuvieron luego del procesamiento de los datos del *input*. En el caso de SonarWAN, **el *output* constituye una descripción del escenario y su comportamiento que tuvo la red cuyo tráfico fue capturado.**

En SonarWAN, existen dos maneras de consumir el *output*, que se explican a continuación:

1. **SonarWAN puede utilizarse como una aplicación de línea de comandos**, independiente de cualquier interfaz gráfica. En este caso, el *output* consiste en un documento de texto donde figura una versión legible e interpretable del escenario descubierto.

2. **SonarWAN puede consumirse como una API para una interfaz gráfica.** En este caso, el *output* del sistema es un documento del tipo JSON para que pueda ser interpretado fácilmente por otro programa que exponga los resultados de una manera gráfica. Este es el caso de uso que se cree que le saca mayor provecho al sistema, pues los datos que refieren al horarios con su cantidad de bytes transferidos no tienen otra manera de apreciarse completamente que no sea con una interfaz gráfica que realice visualizaciones a partir de estos datos. Estas

visualizaciones permiten sacar conclusiones (como por ejemplo observar picos de actividad o detectar usos simultáneos de dispositivos) que serían muy difíciles de obtener desde el caso de uso de la línea de comandos. Más adelante cuando figuren algunos ejemplos y capturas de pantalla se va a poder apreciar por qué una interfaz gráfica le saca mayor provecho al *core* desarrollado.

El componente de la *GUI* de SonarWAN es simplemente un cliente que consume al *core* como API y visualiza sus resultados a través de una interfaz más "amigable". La idea del *core* es que la comunidad usuaria de SonarWAN pueda desarrollar sus propias interfaces gráficas según sus necesidades (información a mostrar, tecnología disponible, etc).

La especificación de la información que provee el *output* va a figurar al final de esta sección, pues para comprenderla primero hay que conocer las entidades y los componentes de SonarWAN, que van a ser explicados en un apartado siguiente.

Arquitectura Física

La tecnología que utiliza SonarWAN es multiplataforma y debería correr en cualquier computadora que soporte Python 3. Incluso, podría funcionar en las conocidas "microcomputadoras" embebidas como la *Raspberry Pi.* El sistema en si no presenta ningún tipo de requisito o configuración específica en lo que respecta a la arquitectura física.

Sin embargo, como el *input* de SonarWAN son archivos de captura (o incluso capturas en vivo, en un futuro), es importante mencionar brevemente a los dispositivos que permiten realizar estas capturas, en especial aquellas capturas que van a analizarse en este proyecto (capturas desde la interfaz WAN de un *gateway* NAT de una red privada).

Estos dispositivos son conocidos como *network taps*, y tienen la capacidad de interceptar de una manera invisible (ningún nodo de la red percibe la existencia del *tap*) el tráfico de una red. En esencia, el *tap* presenta 3 puertos, sean A, B y C. El tráfico fluye normalmente entre A y B (como si hubiera un cable en lugar de un *tap*), pero el *tap* copia todo el tráfico pasante en el puerto C, donde un tercero puede *sniffear* el tráfico. Generalmente, este puerto C está conectado a una computadora que puede estar corriendo *Wireshark* o mismo guardando un archivo con el tráfico para luego invocar a SonarWAN.

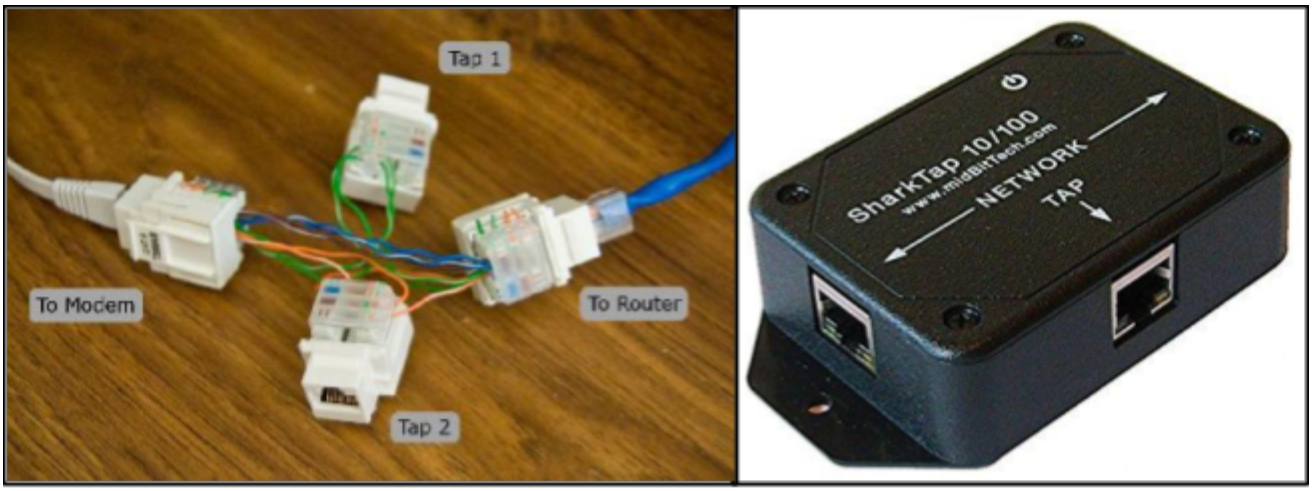


Fig 32. A la izquierda, un tap casero, con dos interfaces para el sniffing, una para tráfico entrante y otra para saliente. A la derecha, un tap comercial. Solo una única interfaz de sniffing donde aparece ya unificado tráfico entrante y saliente.

Otros dispositivos, como un *Raspberry Pi,* o mismo computadoras convencionales pueden simular la misma funcionalidad de un *tap*, es decir, permitir fluir el tráfico a través de sí mismos de una manera transparente pero monitoreando el mismo. La unica restriccion es que deben poseer un mínimo de dos puertos de red (puertos A y B que se mencionan anteriormente).

Arquitectura del *Software*

La arquitectura del *softwar*e está definida por las decisiones de diseño que se efectuaron para **establecer un conjunto de abstracciones lógicas que proporcionan un marco definido y claro para interactuar con el código fuente del software**. La arquitectura definida impone una forma de organizar conceptualmente el código fuente del proyecto. A continuación se van a detallar aquellos aspectos clave que forman parte de esta arquitectura.

Las entidades

Una entidad, también conocido como modelo, **es la representación computacional de un objeto o concepto del mundo real**. En SonarWAN, se modelaron distintos entidades que representan los aspectos que se intenta detectar de la red. Estas entidades son:

- Dispositivos
- Aplicaciones
- Servicios

A continuación se van a detallar brevemente cada una de estas entidades. De las mismas, se va a también especificar la información intrínseca (propia de la entidad) y las referencias que posee con respecto a las otras entidades.

Dispositivos

En esta arquitectura, un dispositivo es un equipo de *hardware* funcional que corre determinado *software*. Ejemplos de dispositivos en este escenario pueden ser celulares inteligentes, tabletas, computadoras de escritorio, computadoras portátiles o televisores inteligentes.

Información Intrínseca

Cada dispositivo tiene un conjunto de características asociadas, las cuales se dividen en dos grupos principalmente, las características de *software* y las características de *hardware*. El primer grupo se refiere a aquellas características se se relacionan con los programas que corre el dispositivo. Por ejemplo, el nombre del sistema operativo, la versión del sistema operativo, los *drivers* que posee, la versión de los *drivers,* el número de *build*, etc. El segundo grupo se refiere a aquellas características del dispositivo que son independientes del *software*, como por ejemplo la marca del dispositivo, el modelo, la versión del modelo, etc. **Este concepto de modelar a los dispositivos como un conjunto de características va a ser esencial para los algoritmos de similitud de dispositivos que se van a explicar luego.**

Además, cada dispositivo posee información acerca del tiempo en el cual estuvo activo en la red y también de la cantidad de *bytes* que se transfirieron desde/hacia este dispositivo en función del tiempo. Esta información se denomina **actividad**.

Referencias a otras entidades

Si bien estas entidades se van a explicar en breve, es importante mencionar que cada dispositivo posee un conjunto de aplicaciones y un conjunto de servicios (servicios que fueron consumidos desde el dispositivo pero no se conoce cuál fue la aplicación contenida en este dispositivo que consumió este servicio en cuestión).

Diagrama

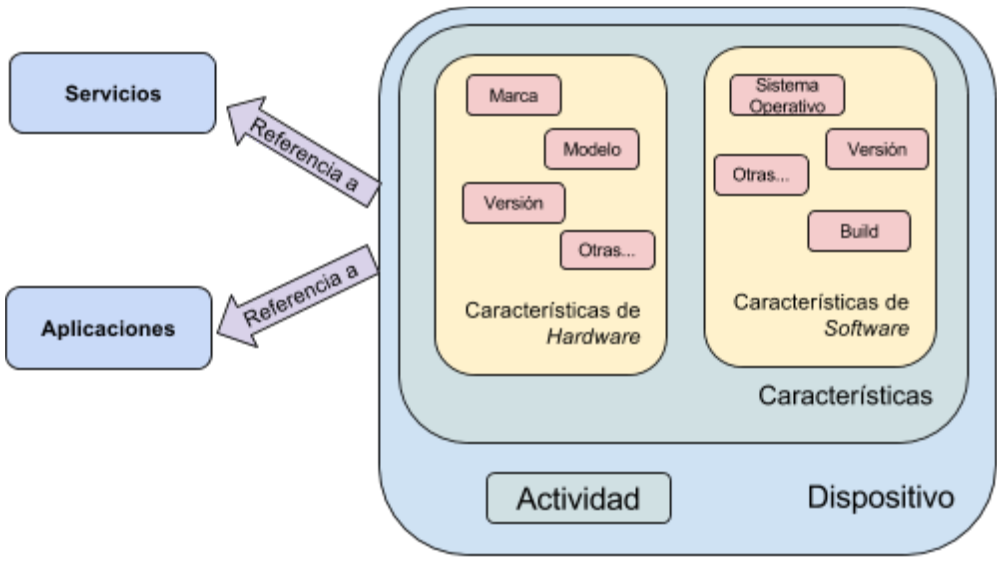


Fig 33. Modelado de un dispositivo

Aplicaciones

En SonarWAN, **una aplicación es un programa de *software* que manifiesta actividad con servicios de Internet y que está asociada con un dispositivo que es el que la ejecuta.** Ejemplos de aplicaciones, en este escenario, pueden ser: navegadores web (o *browsers*), aplicación para móviles de juegos, aplicación para ver películas, etc.

Información Intrínseca

Al igual que los dispositivos, cada aplicación está modelada como un conjunto de características, las cuales pueden ser el nombre de la aplicación, la versión de la aplicación o cualquier otra información atribuible a la aplicación.

El hecho que las aplicaciones también se modelan como un conjunto de características tiene que ver con los algoritmos de *matching* que se desarrollaron y que se van a explicar más adelante.

Referencias a otras entidades

Cada aplicación posee un conjunto de servicios (explicados luego) que fueron consumidos desde la misma.

Diagrama

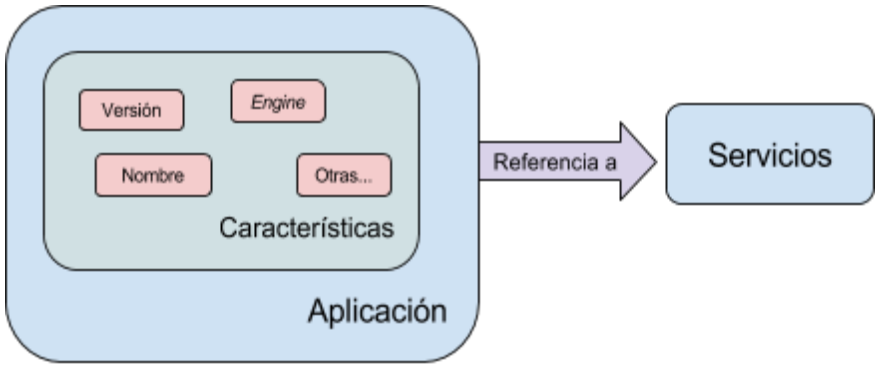


Fig 34. Modelado de una Aplicación

Servicios

Un servicio, en este modelado de entidades, es **un conjunto de recursos disponibles en Internet que buscan satisfacer alguna necesidad o deseo de un cliente.** Estos servicios pueden ser un un diario digital, un portal de streaming de películas (como *Netflix*), un programa de streaming de música (como por ejemplo *Spotify*), o una red social (como *Facebook* o *Twitter*).

En esta arquitectura, se pueden distinguir dos tipos de servicios, los servicios "conocidos" y los servicios "anónimos", según puedan ser atribuibles o no a aplicaciones. Los servicios "conocidos" son aquellos de los cual se sabe cuál fue la aplicación (y por ende el dispositivo) o se sabe cuál fue el dispositivo que los consumió (pero no la aplicación). En cambio, los servicios "anónimos" son aquellos que no se conoce cuál fue la aplicación (ni tampoco el dispositivo) que los consumió.

Información Intrínseca

Cada servicio está representado con un nombre, una categoría (red social, streaming de música, etc.), un conjunto de direcciones IP (aquellas direcciones de los *hosts* que proveen este servicio) y un conjunto de *dominios* (aquellos dominios de los *hosts* asociados al servicio).

Además, cada servicio posee información acerca de su **actividad**, es decir, información acerca del momento en el cual fue consumido este servicio y la cantidad de *bytes* que se intercambiaron en función del tiempo.

Referencias a otras entidades

Un servicio no posee referencia ni a aplicaciones ni a dispositivos, sino que son estos últimos los que poseen referencias al servicio.

Diagrama

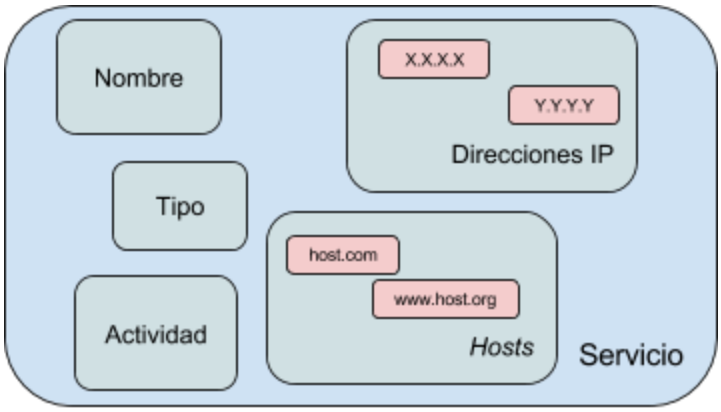


Fig 35. Modelado de un Servicio

Los componentes

Los componentes representan unidades funcionales independientes que se comunican entre para llegar a cabo el objetivo del *software*. **En SonarWAN, se pueden identificar tres componentes principales: el *environment* (o lo que es lo mismo, el escenario), los *handlers* (palabra en inglés que significa "manipuladores") y las *tools* (o en espanol, "herramientas").** A continuación se van a detallar cada uno de los componentes especificando de cada uno la funcionalidad y la responsabilidad dentro del sistema. Como aclaración, cabe mencionar que se va a seguir durante toda esta sección la nomenclatura en inglés (*handlers, environment* y *tools*) de los componentes.

Environment

El *environment* es la representación del escenario en el cual se manifiesta la red analizada a través de sus capturas. Este componente guarda todas las estructura de datos que guardan información acerca de la red de tráfico cuyas capturas se están analizando. **Es el único componente que mantiene un estado.**

Con respecto a las entidades, el *environment* posee:

- **Un conjunto de dispositivos.** Son los dispositivos que se detectaron en la red. Recordar que cada dispositivo además posee servicios y aplicaciones, y estas últimas también poseen servicios.

- **Un conjunto de servicios anónimos.** Son aquellos servicios que fueron consumidos en la red pero que no pueden ser asociados a ningún dispositivo o aplicación.

El objetivo es que el *environment* se vaya modificando y actualizando a medida que se procesan los paquetes de las capturas. Los encargados de mantener actualizado a este componente son los *handlers*, que se explican a continuación.

Handlers

Los *handlers* son los componentes que poseen toda la lógica de comportamiento y de decisión en la que se basa SonarWAN. Estos componentes son los que procesan cada uno de los paquetes que pertenecen a las capturas. **El *handler* es el encargado de actualizar el estado del *environment* de acuerdo a los paquetes que procesa.**

La idea es que cada paquete sea destinado al *handler* correspondiente. El *handler* extrae, del paquete en cuestión, la información necesaria (utilizando las *tools,* que van a ser explicadas luego). Luego, **va a realizar las consultas necesarias al *environment* sobre su estado para poder actualizar al mismo de la mejor manera en base a la información que posee**. Una actualización, podría ser por ejemplo, agregar un nuevo dispositivo, o sumar un servicio a un dispositivo existente, o crear una nueva aplicación para un dispositivo específico, etc.

Ahora bien, los *handlers* están divididos por protocolos y son:

- **DNSHandler**: para paquetes DNS
- **TCPHandler**: para paquetes TCP
- **UDPHandler**: para paquetes UDP
- **HTTPHandler**: para paquetes HTTP

El comportamiento que tiene el análisis de cada uno de estos *handlers* van a ser explicados luego cuando se analice el flujo del programa para los distintos paquetes.

Tools

Como ya se anticipa antes, **las *tools* son las herramientas que utilizan los *handlers* para extraer información acerca de un paquete que está siendo analizado.** Estas *tools* dependen generalmente de archivos que proveen una base de conocimiento para extraer esta información (aspecto explicado luego en esta sección), o bien de librerías externas que sirven para el mismo propósito.

Básicamente, las *tools* son tres:

- ***Inference Engine.*** Significa “motor de inferencias”. Se encarga de inferir información adicional a un conjunto de características. Por ejemplo, si se poseen la característica “familia del sistema operativo” con el valor de “Windows” entonces, este motor puede inferir y agregar la característica de “marca de sistema operativo” con el valor de “Microsoft”.

- ***User Agent Analyzer***. Es la herramienta que permite extraer información a partir de un *header User-Agent* acerca del dispositivo o de la aplicación que haya enviado el *request* HTTP con dicho *User Agent*. La información que extrae es un conjunto de características dividido en dos grupos, por un lado las características del dispositivo y por otro lado las características de la aplicación. Esta información como conjunto de características van a poder

relacionarse con los dispositivos y aplicaciones que posee el *environment* a partir de algoritmos de similitud que van a ser explicados luego. Esta herramienta no solo se basa en un archivos que provee la base de conocimiento, sino que utiliza dos librerías externas (cada una con su base de conocimiento) para poder tener más recursos en la extracción de información.

- ***Service Analyzer.*** Es la herramienta que permite detectar un servicio que está siendo consumido. Esta detección, como se explica en las **estrategias que se van a implementar** que se detallan en la sección anterior, esta detección puede ser por dirección IP o por dominio.

Todos estas herramientas, como se dice anteriormente, dependen de una base de conocimiento provista por el sistema a través de un conjunto de archivos. **SonarWAN, siguiendo uno de sus objetivos de diseño que es la extensibilidad, provee la opción de que un usuario pueda proveer sus propios archivos que colaboren en la base de conocimiento que utilizan estas *tools*.** Este aspecto va a estar desarrollado en el manual de uso presente en el Anexo del Informe.

Comunicación entre componentes

A continuación se muestra un esquema que resume la manera en que se comunican estos componentes.

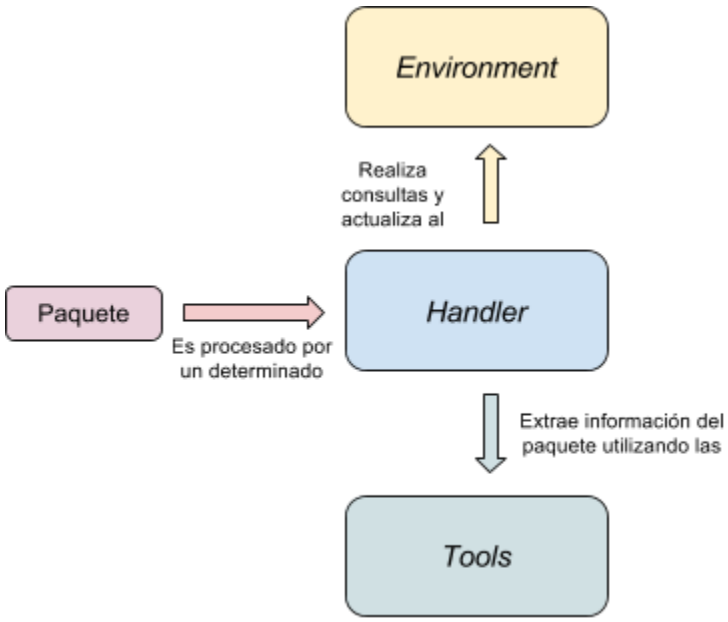


Fig 36. Comunicación entre componentes.

El *stream* como unidad de análisis

SonarWAN va a procesar paquetes secuencialmente provenientes de capturas de tráfico de red. Estos paquetes provienen de dispositivos que se encuentran en este red privada que se está analizando, y pertenecen a cierto evento que está manifestando este dispositivo en cuestión. Para SonarWAN, **un evento es el consumo de un servicio a través de una aplicación.**

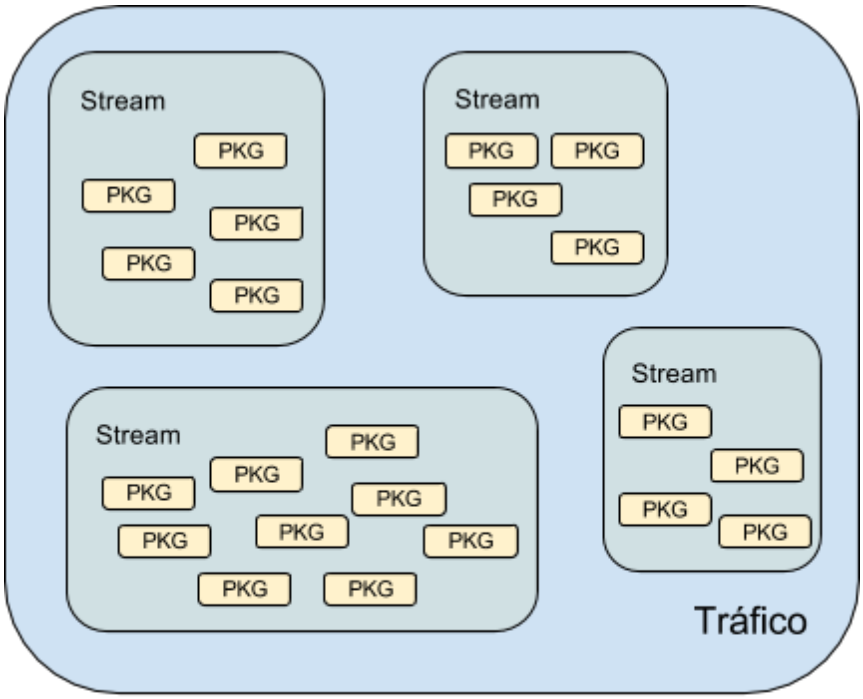


Fig 37. Esquema conceptual del tráfico, visto como un conjunto de streams, donde cada stream es un conjunto de paquetes.

Ahora bien, los paquetes se pueden agrupar a más alto nivel en lo que se denomina *stream*. **Un *stream* es un conjunto de paquetes que representa una "conversación" entre dos duplas [Dirección IP: puerto].** Por ejemplo, siguiendo la nomenclatura de la dupla, un *stream* puede ser un intercambio de paquetes entre [10.16.33.10:100] y [8.8.8.8:53]. **Es importante definir que todos lo paquetes del *stream* involucran a las mismas dos direcciones IP, a los mismos dos puertos, y al mismo protocolo de transporte.**

Un *stream* puede ser:

- **TCP**. En este caso, **el *stream* es una conexión TCP** y va a incluir a todos los paquetes que interactuaron en esta conexión, desde el paquete SYN hasta el paquete FIN.

- **UDP.** En este caso, no existe conexión, pues UDP no es un protocolo orientado a conexiones, pero **el *stream* agrupa aquellos mensajes que**

tienen relación semántica entre sí, como por ejemplo una búsqueda DNS y su respuesta.

Además, un *stream* TCP puede representar también un intercambio *request/response* HTTP entre un cliente y un servidor (o incluso varios *request/response* en el caso de conexiones persistentes); o incluso una comunicación TLS entre nodos.

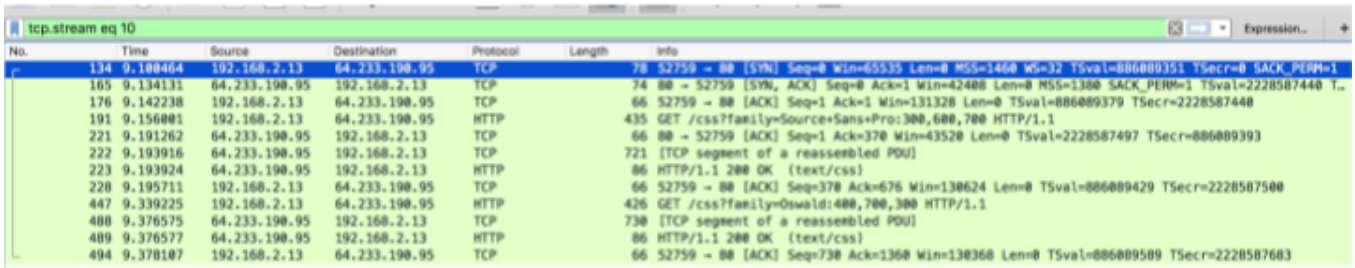


Fig 38. Stream TCP que corresponde a una serie de request/response HTTP. La interfaz corresponde al programa Wireshark

En el escenario de estudio, donde el tráfico es capturado desde la interfaz WAN de un *gateway* NAT, **el *gateway* (o *router*) siempre va a figurar como uno de los dos participantes del *stream***. Esto se debe al mecanismo de NAT que realiza la traducción de direcciones, mecanismo que ya se explicó anteriormente en el trabajo.

Sería ideal poder asegurarse que un *stream* corresponde únicamente a un evento de un dispositivo. Es decir, poder asegurarse que si dos aplicaciones de la red privada buscan consumir el mismo servicio (misma dirección IP y puerto de un *host* de Internet) desde las duplas [A:a] y [B:b], entonces no pueden salir por el mismo puerto del *gateway* NAT y pertenecer al mismo *stream*; sino que el *gateway* se comunica con este *host* externo desde dos puertos distintos, uno para cada comunicación de los *hosts* internos, viendo así dos *streams* en la captura de tráfico NATeado.

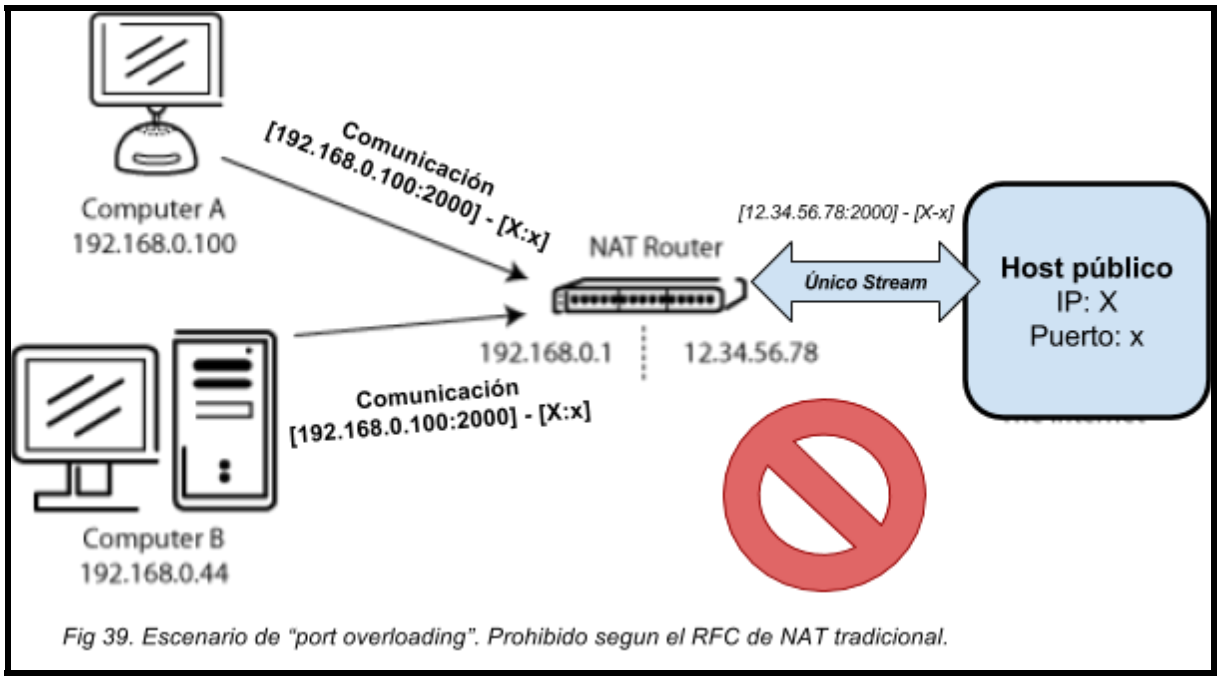


Fig 39. Escenario de "port overloading". Prohibido segun el RFC de NAT tradicional.

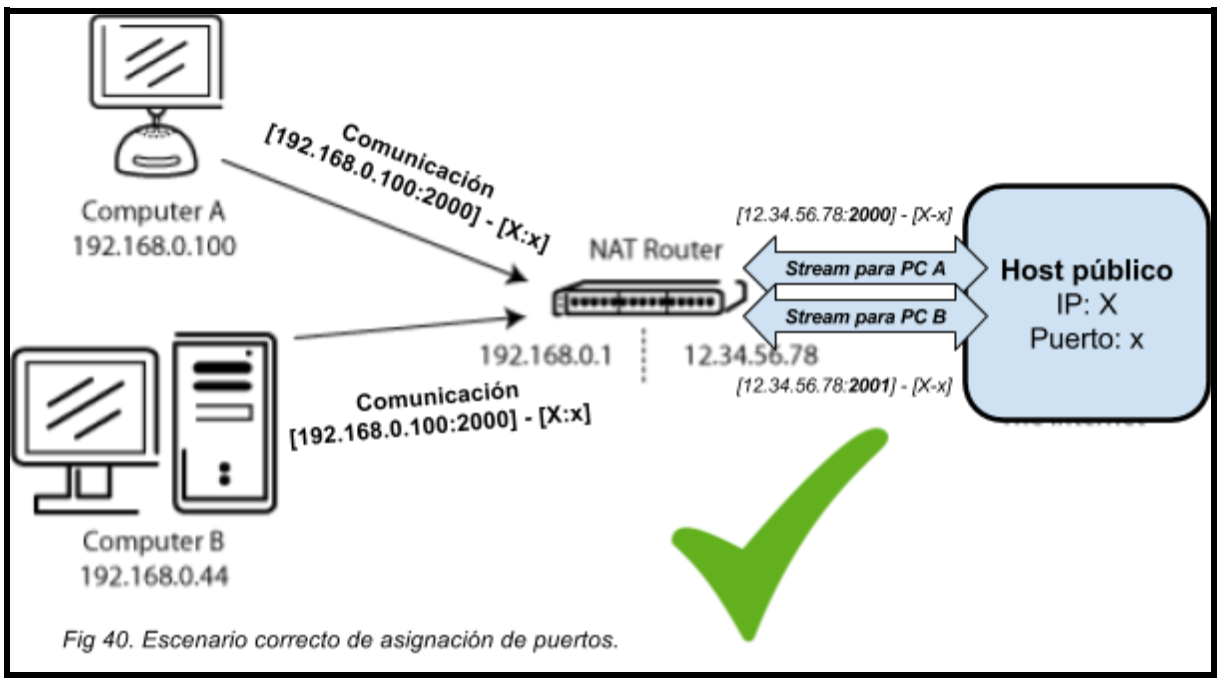


Fig 40. Escenario correcto de asignación de puertos.

En el caso de ser cierto que un *stream* corresponde únicamente a un evento de un dispositivo, **la lógica de procesamiento de paquetes de SonarWAN se vería altamente simplificada.** Por ejemplo, si ya se identificó un evento para un paquete analizado, entonces cuando llegue otra paquete perteneciente al mismo *stream*, se puede indudablemente asociar al evento identificado con el paquete anterior. Es decir, **con poder detectar eventos a partir de un solo paquete, todos los otros paquetes que pertenecen ese *stream* pueden automáticamente asociarse al evento en cuestión.**

Ahora bien, **el hecho que un *stream* corresponde únicamente a un evento de un dispositivo es cierto**, pues de no ser así el dispositivo NAT estaría teniendo un comportamiento de “Port Overloading” en lo que respecta a la asignación de puertos (comportamiento prohibido segun el estándar de NAT tradicional, recordar las observaciones realizadas con respecto al RFC de NAT en la sección anterior).

Por estas razones y por la manera en que se simplifica el procesamiento en SonarWAN, se habla del *stream* como unidad de análisis. El *stream* pasó a ser **la agrupación mayor de paquetes que se puede asegurar sin margen de error que pertenecen al mismo evento**.

Entonces, ¿Cómo aprovecha esta situación SonarWAN? **Todas las entidades o componentes que mantienen referencias a otras entidades (el *environment* que posee referencias a servicios y dispositivos; los dispositivos que poseen referencias a aplicaciones; y las aplicaciones que poseen referencias a servicios), guardan en mapas información de la relación entre el *stream* y la referencia.**

Por ejemplo, el *environment* posee referencias a los dispositivos que pertenecen a la red. Una de sus estructuras de datos es un mapa (llamado *device_stream_map*), donde hay valores *key-value* de la forma *stream-dispositivo*. Entonces, cuando a partir de un paquete se crea un dispositivo nuevo, se lo agrega al *environment* y además se agrega el *stream* al que pertenece el paquete y el nuevo dispositivo al mapa. Cuando luego se tiene que procesar un paquete que pertenece al mismo *stream*, se busca en el mapa si existe un dispositivo asociado a ese *stream*. Al ser la búsqueda exitosa, entonces ya se conoce el dispositivo que emitió (o recibió) ese paquete. Este mismo ejemplo se puede trasladar a todas las otras entidades que poseen estos mapas. **Es importante poner en evidencia el poder que tiene este método. Con un paquete (que quizas por sí solo no tiene manera de asociarse a un evento, porque es por ejemplo un FIN de una conexión TCP) y unas búsquedas rápidas en estos mapas, SonarWAN puede asociar al paquete no solo al dispositivo, sino también a una aplicación y a hasta el servicio que fue consumido.**

Flujo de información y algoritmos implementados

Ya habiendo explicado las entidades y los componentes de SonarWAN, se puede proceder a explicar puntualmente cómo el el flujo de información y de decisión del sistema a medida que va procesando los paquetes que componen las capturas.

El flujo principal es simple, por cada archivo de captura se procesa cada uno de los paquetes secuencialmente. El procesamiento realizado esta representado en la figura 36, donde aparece la comunicación entre componentes.

A continuación se va a especificar el flujo para cada paquete según su protocolo. Como se dijo anteriormente, cada protocolo va a ser procesado por un *handler* distinto. Como se va a poder observar, **en SonarWAN se analizan y se toman en cuenta casi todos los paquetes que forman parte de la captura.** Si bien no todos ellos sirven para detectar nuevos eventos, todos intervienen en la sumatoria de *bytes* y los horarios de actividad.

Paquetes DNS

Cuando llega un paquete DNS, el encargado de procesarlo es el *DNSHandler*. Únicamente se procesan las respuestas DNS, pues contienen la información también de las búsquedas sobre las cuales responden. La única utilidad de estos paquetes es almacenar las búsquedas con los resultados en el *caché DNS*, estrategia explicada antes que se mencionó que iba a ser implementada. Por lo tanto, lo único que realiza el *handler* es guardar la búsqueda con su respuesta en una estructura de datos que se almacena en el *environment* (recordar que es el único componente que guarda estado).

Paquetes TCP y UDP

Estos paquetes son los que poseen en la última capa alguno de estos protocolos (es decir, no llegan a ser identificados como paquetes de capa de aplicación).

La lógica de procesamiento para estos paquetes es simple pero constituye el soporte para poder realizar un buen cálculo de la actividad de la red.

En primer lugar, el *handler* correspondiente (*TCPHandler* o *UDPHanlder*) consulta al *enviroment* si el *stream* al que pertenece el paquete ya fue asociado a alguna entidad (que puede ser un dispositivo o un servicio anónimo). Pueden existir dos casos:

1. **El *stream* al que pertenece el paquete ya fue analizado.** En este caso, el *handler* puede acceder rápidamente a la entidad correspondiente a través del mapa explicado anteriormente. (En el caso de ser un dispositivo, también puede acceder a través de los mapas al servicio consumido desde ese dispositivo correspondiente de ese *stream.*) Con la posesión de las entidades relacionadas a ese *stream*, lo único que tiene que realizar es actualizar la actividad de dichas entidades, es decir, agregarle el horario y la cantidad de *bytes* que aparecen en el paquete en cuestión.

2. **El *stream* al que pertenece el paquete NO fue analizado.** En este caso, lo único que se puede detectar es un nuevo servicio anónimo, pues las estrategias para detectar dispositivos y aplicaciones se puede efectuar únicamente con paquetes HTTP. Por lo tanto, en el caso de detectarse un servicio (por dirección IP o por dominio), este servicio sera anonimo (podría luego dejar de ser anónimo y asociarse a un dispositivo, cuestión que se va a explicar luego). El *stream* entonces realiza la búsqueda del servicio, por dirección IP o por dominio (consultando el *caché DNS*). Si se pudo detectar un servicio, pueden ocurrir otras dos cosas:

    a. El *environment* ya posee este servicio anónimo que fue consumido en otro momento. En este caso no se crea un servicio anónimo y se agrega la actividad al existente. Cabe destacar que la igualdad entre servicios se realiza por nombre.
    b. El *environment* no posee un servicio anónimo igual. En este caso se genera un nuevo servicio anónimo con la actividad en cuestión y se la agrega al *environment.*

En todos los casos, se actualizan los mapas que contienen la información de los *streams* para que puedan utilizarse con otros paquetes. Además, para todos lo paquetes TCP o UDP que no pudieron ser asignados ni a dispositivos ni a servicios anónimos (nuevos o existentes) porque no poseían la información necesaria, su actividad (tiempo y cantidad de *bytes*) se guarda en estructuras temporales asociada al *stream* al que pertenecen. Cuando aparece algún paquete de dicho *stream* del cual si se puede identificar alguna entidad, entonces se tiene en cuenta también la actividad previa del *stream* extrayéndose de estas estructuras auxiliares.

Paquetes TLS

Como ya se mencionó anteriormente, no se va a implementar la estrategia de los *cipher suites* y *compression methods* que permite distinguir dispositivos. Al no proveer ningún tipo de información extra, van a ser procesados como paquetes TCP.

Paquetes HTTP

Como se explica en las estrategias que se decidieron implementar, los paquetes HTTP son los únicos a partir de los cuales se puede extraer información acerca del dispositivo y aplicación en cuestión. La lógica es similar a la de los paquetes TCP o UDP, en el sentido de que primero se busca en el mapa del *enviroment* si existe un dispositivo asociado al *stream* al cual pertenece el paquetes. Existen dos opciones:

1. **El *stream* está asociado a un dispositivo.** En este caso, se actualiza la actividad de este dispositivo, y en el caso de haber también un servicio asociado, se actualiza tanto la actividad como las direcciones IP o *host* que lo caracterizan. Además, en el caso de ser un *request* (la única posibilidad es que sea una conexión persistente y haya habido más de un *request/response* en este *stream*), se extraen las características del *user-agent* y con ellas se actualizan las características del dispositivo.

2. **El stream NO está asociado a un dispositivo.** En este caso, solo se procesan los *request* (que tienen el *header user-agent*). El *handler* extrae las características del *user-agent* (con la *tool* de *UserAgentAnalyzer*) y busca si existe un dispositivo que sea compatible con estas características según el **algoritmo de similitud** que va a ser explicado luego. De no existir, entonces crea un dispositivo nuevo. Ya teniendo posesión del dispositivo (haya sido creado o encontrado por similitud), se actualizan las características y se agrega la actividad. Luego, en el caso de haber características de la aplicación en la extracción inicial, entonces para ese dispositivo se busca (con un algoritmo de similitud similar al anterior) o se crea una aplicación, a la cual se le actualizan también las características. A continuación, se realiza una búsqueda para averiguar si existe un servicio asociado a este paquete HTTP (la técnica para encontrar el servicio es la misma que con los paquetes TCP/UDP a diferencia que se agrega el recurso del *header Host*). De encontrar un servicio, se agrega la actividad correspondiente a este servicio como también la información acerca de la dirección IP y el dominio. Finalmente, el *HTTPHandler* debe consultar al *environment* si posee un servicio anónimo asociado a ese *stream*. En el caso de existir este servicio anónimo, entonces se asigna únicamente la fracción del servicio correspondiente a ese *stream* al dispositivo o a la aplicación del dispositivo. **Este es el caso en el que esta fracción del servicio anónimo deja de ser "anónimo".**

   Es importante mencionar algunas cuestiones:
   - **Siempre que se actualizan las características de un dispositivo, se corre el motor de inferencias para ver si se pueden inferir más características.**
   - Siempre se mantienen actualizados los mapas tanto de los dispositivos como de las aplicaciones.
   - Cuando se genera un nuevo dispositivo a partir de un nuevo *stream*, se agrega toda la actividad de los paquetes TCP previos de ese *stream* que están almacenados en las estructuras temporales del *environment*.

Algoritmo de similitud

El algoritmo de similitud tiene dos usos, para detectar dispositivos compatibles con un conjunto de características o para detectar aplicaciones compatibles con un conjunto de características. Este algoritmo lo va a ejecutar el *HTTPHandler* cuando tenga que buscar un dispositivo y una aplicación para un nuevo *stream* HTTP.

Anteriormente, cuando se específico acerca del modelado de las entidades, se hizo énfasis en representar al dispositivo y la aplicación como un conjunto de características. Bajo este modelado, el algoritmo se reduce al siguiente problema: **teniendo un set de conjuntos de características y dado un conjunto específico de características, encontrar al elemento del set que mas compatible es con ese conjunto dado.** En el caso del uso para detectar dispositivos compatibles, este set va a estar constituido por todos los conjuntos de características de los dispositivos existentes en el *environment* y el conjunto dado va a ser las características del dispositivo extraídas de un *User-Agent*. En el

caso de uso para detectar aplicaciones, este set va a estar conformado por todos los conjuntos de características de las aplicaciones asociadas a un dispositivo y el conjunto dado va a ser las características de la aplicaciones extraídas de un *User-Agent.*

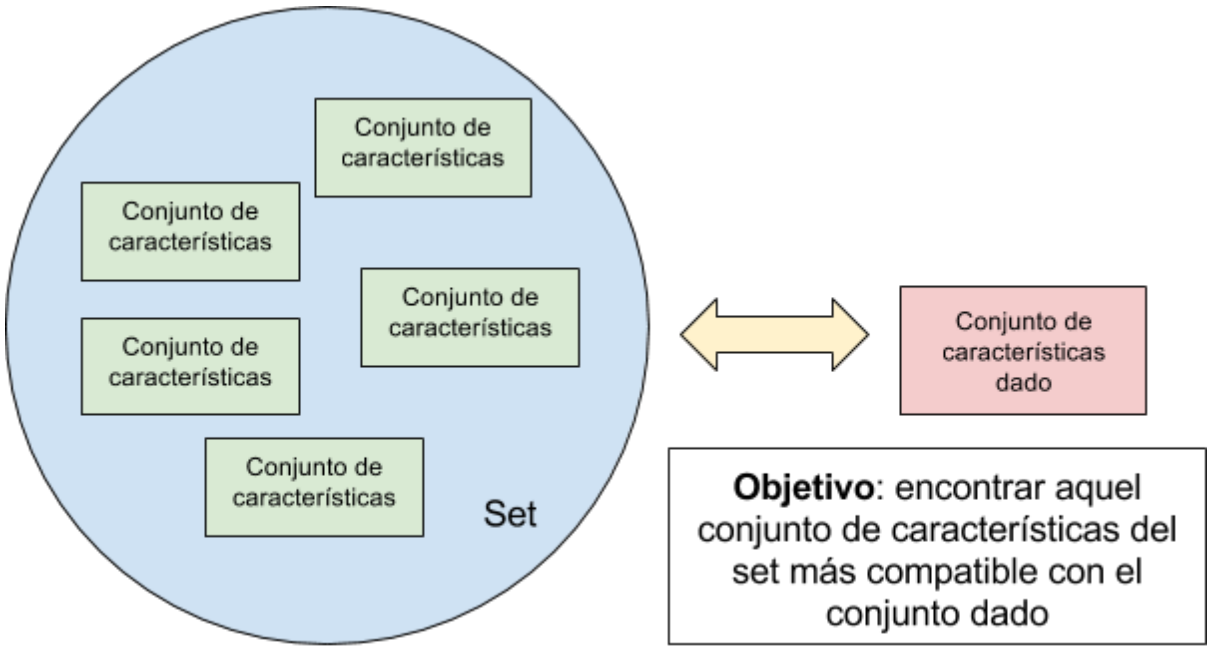


Fig 41. Diagrama descriptivo del algoritmo de similitud.

El algoritmo planteado es un algoritmo de *scoring*, es decir, por cada elemento del set (cada elemento es un conjunto de características) pone un puntaje de compatibilidad con el conjunto de características dado (en el caso de no sea compatible, el puntaje es de -1). Luego, se fija aquel conjunto que tiene el mayor puntaje y lo retorna. En el caso de haber muchos con el mismo puntaje, retorna uno al azar. En el caso de no haber ninguno compatible, no retorna ningún elemento.

El puntaje de cada conjunto de características en relación a un conjunto de características dado se calcula con el *matcheo* de sus caracteres. Por cada característica común entre ambos conjuntos, se suma puntaje si el *string* de esa característica de un conjunto es "*substring inicial"* (por ejemplo, "hola" de "holachau" es "*subtring inicial"*) del otro *string* de la misma característica del otro conjunto, o viceversa. En el caso de no ser *substring inicial*, resultan incompatibles los conjuntos y ese conjunto del set recibe el puntaje de -1. El conjunto que tenga mayor cantidad de *substrings iniciales* con el conjunto dado, va a poseer el puntaje más alto.

Resulta más claro con un ejemplo gráfico:

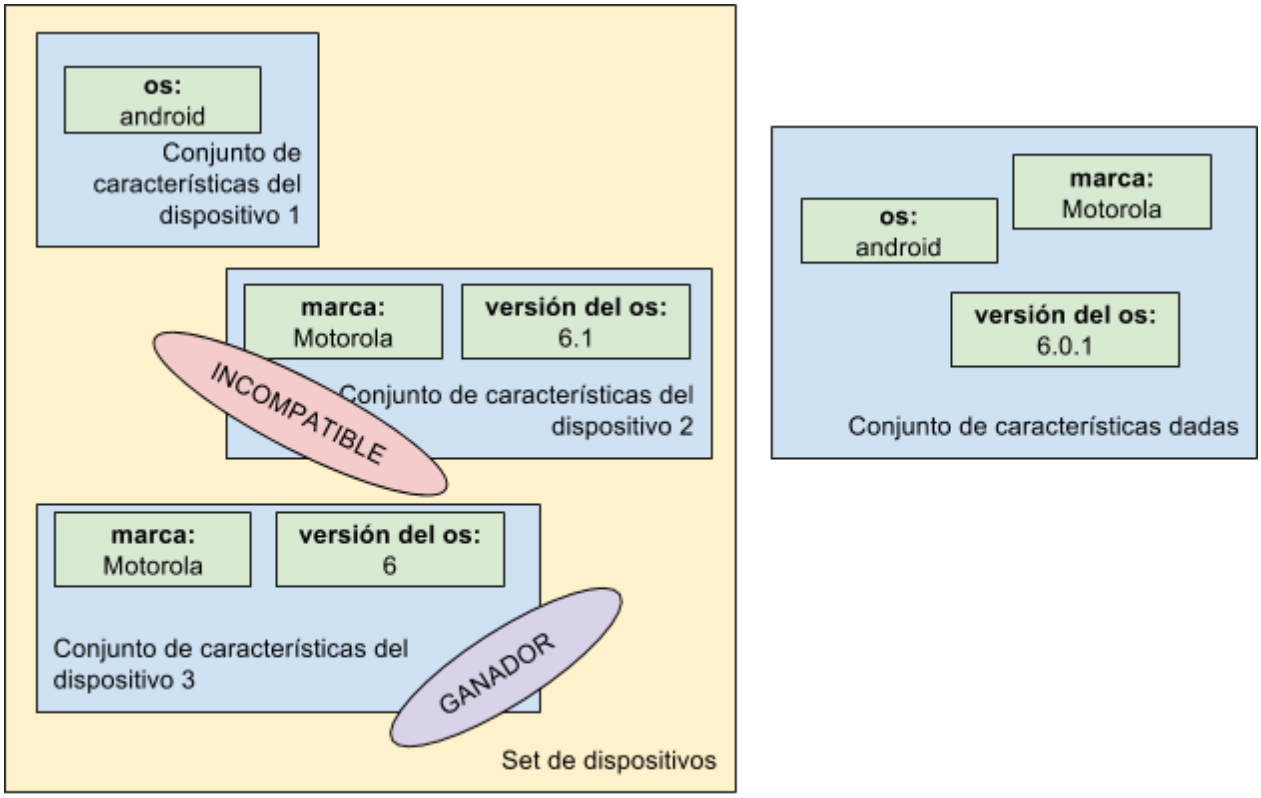


Fig 42. Ejemplo del algoritmo para detección del dispositivo más compatible a partir de un conjunto de características dadas. El dispositivo 2 es incompatible por la versión. Tanto el 1 como el 3 son candidatos pero el 3 obtiene mayor puntaje al matchear más características..

Este algoritmo es el responsable de que los distintos *streams* HTTP que poseen información acerca de los dispositivos y de las aplicaciones se vinculen correctamente a los dispositivos existentes y a las aplicaciones existentes en la red; evitando, por ejemplo, que se SonarWAN detecte dos dispositivos distintos cuando en realidad en la red había uno solo.

Algoritmos de detección de un servicio

Este algoritmo va a ejecutarse en el caso en que se conoce el dominio del servicio consumido, pero la base de conocimiento del *ServiceAnalyzer* no puede especificar puntualmente de qué servicio se trata. Por ejemplo, se cuenta con el dominio “www.lanacion.com”, pero no existe base de conocimiento que pueda decir que ese dominio se relaciona con el servicio cuyo nombre es “La Nacion”.

En estos casos, la deducción del nombre del servicio se hace con el dominio mismo. **El nombre del servicio va a ser la palabra con longitud mayor a 3 caracteres que se encuentre en el dominio, analizando de derecha a izquierda.** Por ejemplo, “www.lanacion.com.ar” va a ser “lanacion”, “photos.argentina.facebook.com” va a ser “facebook”. Sin embargo, existen casos en que este mecanismo no resulta eficiente, pues el dominio no tiene ningún tipo de relación con el nombre del servicio. Para evitar este comportamiento, este dominio debe vincularse al servicio correspondiente en la base de conocimiento disponible para el *ServiceAnalyzer*.

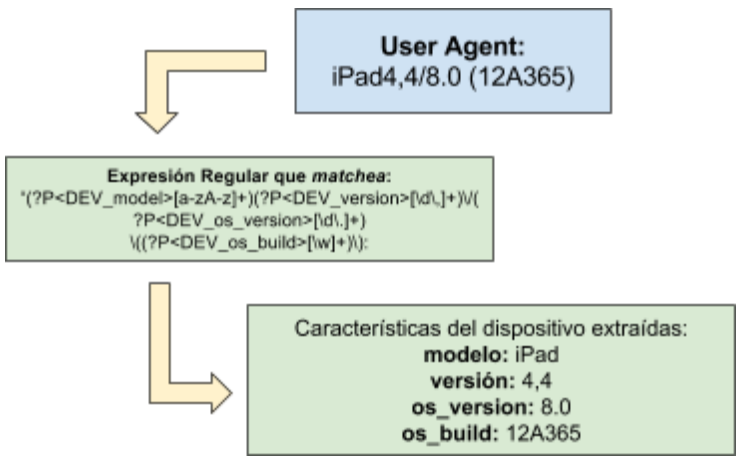
Base de conocimiento

Durante toda esta sección, en particular cuando se habla de las *tools*, se hace referencia a la **base de conocimiento (abreviado BC)**; la cual es un conjunto de archivos que le dan capacidad de resolución al *Inference Engine, User Agent Analyzer* y *Service Analyzer*. A continuación se especifican brevemente cómo son estos archivos para cada uno de estas *tools*:

- **BC para *Inference Engine***. Son archivos *.csv*, donde en la primera línea figuran las características del dispositivo, y luego cada línea representa a una inferencia posible distinta. El *Inference Engine* se crea un motor de inferencias a partir de todos estos archivos que es capaz de inferir características a partir de otras. Ejemplo de archivo:

```
os_family;os_brand
Windows;Microsoft
Android;Google
iOS;Apple
Mac OS X;Apple
```

- **BC para *User Agent Analyzer***. Es un archivo con extensión ".*patterns*" que contiene expresiones regulares siguiendo la sintaxis de Python. En cada expresión regular, se pueden identificar ciertos *values* clave bajo lo que se conoce como *keys*. Estos *key-value* que van a ser las características que se extraen. Para diferenciar las características del dispositivo de los de las aplicaciones, aquellas *keys* que comienzan con "DEV_" van a ser características del dispositivo; en cambio las que comienzan con "APP_" van a ser las de la aplicación. El *User Agent Analyzer* carga al inicio todas las expresiones regulares, y cuando llega un *user-agent*, busca la expresión regular que *matchea*, para extraer las características y retornarlas. Por ejemplo:



- **BC para *Service Analyzer*.** Es un conjunto de archivos *yaml*. En cada uno aparece un servicio distinto, para el cual se puede especificar las direcciones IP asociadas (*header* “ips”), los dominios absolutos (*header* “absolute-urls”) o los dominios no-absolutos (*header* “urls”). Con respecto a los no-absolutos, por ejemplo, “www.facebook.com” matchea con el dominio no-absoluto “facebook.com”, pero si este último fuera absoluto, no habria *match*. Con esta información, el *Service Analyzer* va a poder hacer la detección de servicios por dirección IP y/o por dominio. Ejemplo de un archivo de este tipo:

Breve mención de la *performance*

Si bien la *performance* no es el atributo de calidad más importante de este sistema, siempre es bueno realizar una referencia a este aspecto.

En SonarWAN, la lectura de los paquetes se realiza de una manera secuencial, y **cada paquete se inspecciona una única vez**. Esto quiere decir que el sistema no va recorriendo la captura múltiples veces buscando paquetes para relacionar. En esencia, por cada paquete que se lee, se inspecciona y se actualiza el escenario. Esta característica hace que SonarWAN tenga una buena *performance*, y es una de las razones por las cuales SonarWAN va a poder permitir el análisis con capturas en vivo.

Especificación del *output*

Si bien antes se habla de los formatos en los que se presentan los datos en el *output*, se anticipa que luego de explicarse los componentes y las entidades se va a realizar una mención más específica de la información del *output*. La especificacion de esta información que provee el *output* se puede dividir en:

- **Sumario**: dispone la cantidad de dispositivos que se detectaron y de servicios anónimos (luego van a ser explicados). También se menciona la cantidad de archivos, el conjunto de paquetes que se inspeccionaron y el tiempo de ejecución. que se analizaron.

- **Detalle:**
    - **Por cada dispositivo**, se informa sus características y la actividad que tuvo (cantidad de *bytes* en función del tiempo). Además, se informan:
        - **Servicios asociados a ese dispositivo pero a ninguna aplicación.** De estos dispositivos se menciona la actividad, los *hosts* y las direcciones IP que intervienen.
        - **Aplicaciones.** Cada aplicación asociada a ese dispositivo con sus características. Además, para cada aplicación, aparecen los servicios que fueron asociados a esa aplicación de ese dispositivo (con información de la actividad, los *hosts* y las direcciones IP).
    - **Por cada servicio anónimo,** se informa la actividad (cantidad de *bytes* en función del tiempo), los *hosts* y las direcciones IP involucradas en el consumo de ese servicio.

En el Anexo de este Informe va a figurar un *output* ejemplo a modo de demostración.

Breve mención a la GUI

La interfaz de usuario, como se menciona anteriormente, constituye el *front-end* de Sonarwan. Este componente utiliza al *core* para procesar los archivos que el usuario cargue para analizar. La interfaz de usuario se resume en las siguientes dos funcionalidades:

- **Carga de archivos**. El usuario va a poder seleccionar los archivos de captura a analizar por Sonarwan.
- **Presentación de resultados**. La interfaz va a parsear el *output JSON* del *core* para generar visualizaciones, tablas, líneas de tiempo, etc.

Esta componente es una aplicación de escritorio multiplataforma y esta desarrollada con tecnología web, utilizando el framework *Electron*. Más información de la GUI, tanto capturas como instrucciones de uso, van a figurar en el Anexo.

Nivel de satisfacción del sistema desarrollado

Para este trabajo, SonarWAN ser considera efectivo si garantiza un funcionamiento satisfactorio bajo ciertas condiciones. Un buen funcionamiento equivale a un margen de error bajo, es decir, que el escenario que se ha detectado (dispositivos, aplicaciones y servicios) sea lo más cercano al escenario real que se encuentra en la red.

Estas condiciones son:

- El tráfico debe corresponder a una red doméstica.
- Dicha red debe tener menos de 10 dispositivos
- Dicha red debe poseer dispositivos “comunes”
- Inexistencia de dispositivos clones (varias instancias exactamente similares del mismo dispositivo).

Si bajo estas condiciones se obtiene un escenario con bajo margen de error, SonarWAN se considera que tuvo éxito.

Sección V: **Resultados**

Introducción

En esta etapa se van a exponer cuáles fueron los resultados de la etapa de prueba. Como se explica anteriormente en la sección de Metodología, esta etapa de prueba fue la última de las etapas que tuvo lugar en el proyecto y consistio en **poner a prueba el sistema en escenario reales para ver como responde de acuerdo al alcance y limitaciones definidas anteriormente, y, de ser necesario, realizar los ajustes correspondientes para terminar de consolidar el sistema desarrollado.** Las pruebas que se corrieron fueron:

1. Pruebas con capturas propias.

2. Pruebas con capturas desconocidas facilitadas por el tutor.
3. Pruebas de *stress* con capturas extensas.

Resulta importante realizar la distinción entre las pruebas con escenario conocido y las pruebas con escenario desconocido. En las primeras, se conocen absolutamente todos los eventos que se realizaron en la red. Estas pruebas fueron creadas específicamente para simular estos eventos. Teniendo el *output* para estas pruebas, es posible medir el rendimiento (pues se conoce cual tendría que ser el *output* ideal). En cambio, las segundas pruebas (bajo escenarios desconocidos), no se conocen todos los eventos que se manifestaron en la red. No es posible conocer el *output* ideal y, por lo tanto, no es posible medir el rendimiento de una manera objetiva (se pueden realizar estimaciones de acuerdo a la información disponible de la captura).

Para cada una de las pruebas, se va a especificar el escenario (en el caso que sea conocido) y **se va a resumir el *output* obtenido.** En el caso de ser escenarios conocidas (pruebas con capturas propias), **se va a puntuar objetivamente el rendimiento de SonarWAN.**

Ajustes durante la etapa de prueba con capturas propias

Como se anticipa anteriormente, esta etapa también iba a servir para realizar ajustes necesarios para consolidar al *core* de SonarWAN. Se pueden identificar dos tipos de ajustes que tuvieron lugar en la etapa de prueba. **Ninguno de estos ajustes representó cambios significativos en la lógica de procesamiento de los *handlers* ni en las estructuras de datos que guarda tanto el *environment* como las entidades**. Esta característica pone en manifiesto la efectividad que se obtuvo principalmente en la etapa de desarrollo, en lo que involucra al diseño de algoritmos, de componentes y entidades.

Los dos tipos de ajustes que se mencionan son:

1. **Ajustes para extender/mejorar la base de conocimiento**. La base de conocimiento, como se explica en la sección anterior, es el motor de los *handlers;* y el éxito de SonarWAN depende, en parte, de cuanta información haya disponible en

la base de conocimiento. A medida que se hicieron las pruebas, se observaron resultados que podían agregar o incluso corregir información de esta base de conocimiento (por ejemplo, agregar *urls* a ciertos servicios en sus respectivos archivos *yaml*).

2. **Ajustes menores frente a nuevas situaciones.** Estos ajustes tienen que ver con imprevistos que van surgiendo a medida que se trabaja con nuevas capturas y tiene que ver con comportamientos de los *hosts* que no son *estándar* (o *tradicionales*) para su actividad en red. Por ejemplo, paquetes de una captura de red doméstica que no utiliza protocolo IP en la capa de red, mensajes HTTP que no viajan por TCP, etc. Estos imprevistos sirvieron para realizar pequeños ajustes y evitar que SonarWAN falle cuando se presentaban estas situaciones, que antes se consideraban inexistentes.

Sistema de puntuación

Como se dijo anteriormente, para cada prueba de escenario conocida realizada se asignó un puntaje de acuerdo al nivel de efectividad que tuvo SonarWAN. Dicho puntaje se asigna en un rango de 1 a 6, siendo 1 el nivel más bajo de efectividad y el 6 el nivel más alto. Es importante mencionar que **este criterio de puntuación es independiente de las capacidades potenciales de SonarWAN**, es decir, que la puntuación tiene que ver con el objetivo a cumplir y no tanto con las facultades de SonarWAN. Por ejemplo, si el tráfico no posee tráfico HTTP, SonarWAN (como fue explicado anteriormente en la sección de Implementación) no va a detectar dispositivos. En términos del SonarWAN, este rendimiento es esperado por como fue diseñado; pero en términos del objetivo, este rendimiento es bajo, pues no cumple con las expectativas esperadas (no detecta dispositivos). En este caso, la puntuación es baja pues **el criterio de puntaje es dependiente del objetivo y no de SonarWAN.**

El criterio que guía la puntuación es el siguiente:

1. SonarWAN no detecta ningún tipo de información relevante, ya sean dispositivos, aplicaciones o servicios consumidos.
2. SonarWAN detecta únicamente algunos dispositivos en la red, algunas aplicaciones o algunos servicios consumidos; pero no todos.
3. SonarWAN combina dispositivos, aplicaciones o servicios distintos bajo uno único; o genera dispositivos, aplicaciones o servicios extra.
4. SonarWAN detecta correctamente dispositivos, aplicaciones y servicios, pero existe algún error relevante en términos de asignación o caracterización.
5. SonarWAN detecta los dispositivos, aplicaciones y servicios (anónimos o no) consumidos correctamente.
6. SonarWAN detecta y asigna todo a la perfección, tanto dispositivos, como aplicaciones y servicios consumidos. No existen los servicios anónimos pues todos están asignados.

Resumen de los resultados de las pruebas

Pruebas con capturas propias

Las pruebas se dividieron según la cantidad de dispositivos que estaban involucrados en la red. Se realizaron pruebas con 1 dispositivo, con 2 dispositivos y con 3 o más dispositivos.

Para todas las pruebas realizadas, se caracteriza cada dispositivos como así también los eventos que produce en la red. Aparece además la complejidad de la prueba (que puede ser Baja, Media o Alta), la cantidad de archivos y su tamaño. Luego de ejecutar la prueba, se especifica la cantidad de paquetes analizados, el tiempo de ejecución, las observaciones del *output* y su puntaje.

Toda esta información sobre los resultados de las pruebas se encuentra en el Anexo en forma de tablas. A continuación únicamente se presenta un resumen con ciertos valores clave obtenidos.

Dispositivos en red	**Tiempo**	**Puntaje Promedio**	**Observaciones generales**
1	0,37 seg /100 paquetes	4,4/6 (~7,34/10)	● **Las pruebas eran generalmente de complejidad baja.** ● **Incluye celulares, computadoras, tabletas y hasta un lector electrónico.** ● **Nunca detecta dispositivos extra.**
2	0,45 seg/100 paquetes	4,4/6 (~ 7,34/10)	● **Las pruebas eran generalmente de complejidad media.** ● **Incluye celulares, computadoras y tabletas.** ● **Nunca detecta dispositivos extra.** ● **Nunca combina dispositivos en uno solo.**
3 o más	0,18 seg/100 paquetes	4,34/6 (~ 7,23/10)	● **Las pruebas eran generalmente de complejidad alta.** ● **Hay pruebas de hasta 5 dispositivos.** ● **Incluye celulares, computadoras y tabletas.** ● **Nunca detecta dispositivos extra.** ● **Hubieron combinaciones de dispositivos por ser clones.**

Es posible que el valor del puntaje distorsione la efectividad de SonarWAN, pues se obtuvieron valores cercanos a 7.5/10. **Es necesario volver a remarcar que el criterio de puntaje no es dependiente a las capacidades de SonarWAN (de ser así, siempre se obtendría el puntaje máximo), sino del objetivo del sistema**. **Es posible que no exista sistema alguno que pueda lograr puntaje promedio perfecto (6) bajo el escenario de captura con el que se trabaja en este proyecto.**

Pruebas con capturas desconocidas facilitadas por el tutor

La prueba realizada bajo este escenario se puede resumir en la siguiente tabla:

Prueba	**Archivos**		**Paquetes Analizados**	**Tiempo de Ejecución [seg]**	***Output***
	Cantidad	**Tamaño**			**Breve Descripción**
1	3	10,7 MB	31551	68,5 (Promedio 0.22 seg/ 100 paquetes)	- Detecta 2 dispositivos: - Motorola con Android 6.0 - Computadora Linux con Ubuntu - De la computadora, se detecta actividad con navegadores Chrome y Firefox, donde se visitaron principalmente diarios (página12, infobae, etc) - Se detecta actividad de WhatsApp y (poca) de Telegram

Al ser una prueba desconocida, no se puede puntuar ni describir la calidad exacta de la *performance* de SonarWAN. Se estima haber tenido un puntaje cercano al 5 (según el sistema de puntaje implementado para las pruebas de escenarios conocidos), pues el tutor afirmó que la detección de dispositivos fue correcta, que no hubieron dispositivos sin detectar, que la detección de aplicaciones y servicios asociadas a la computadora es correcta y que los servicios de mensajería detectados también son correctos.

Pruebas de *stress* con capturas extensas

El objetivo de esta prueba fue de ejecutar SonarWAN en un escenario real, es decir, con capturas realistas. En estas capturas, los eventos de los dispositivos no son simulados, sino que se realiza un *sniffing* continuo de una red en su comportamiento normal. **Este tipo de capturas representa el escenario donde efectivamente se va a utilizar SonarWAN.**

Lo desafiante de esta prueba es que las capturas suelen abarcar un periodo de tiempo largo y por lo tanto, son extensas. Los archivos fácilmente superan los GB de tamaño. **Someter a SonarWAN ante tanta carga de paquetes representa lo que se conoce como prueba de *stress*** (prueba de exigencia). De pasar estas pruebas satisfactoriamente, SonarWAN pone en evidencia su capacidad para utilizarse en escenarios reales que involucren capturas extensas.

Para realizar esta prueba, **se capturó tráfico en una casa durante un lapso mayor a una hora.** Un resumen de los resultados se disponen a continuación:

Archivos		**Paquetes Analizados**	**Tiempo de Ejecución [seg]**	***Output***
Cantidad	**Tamaño**			**Breve Descripción**
16	1.12 GB	1114035	3790,5 (Promedio 0.34 seg/ 100 paquetes)	- Detecta 6 dispositivos: - 2 iPad - 2 iPhone - 1 MacbookPro - 1 Nexus 5 - Detecta múltiples aplicaciones para cada dispositivo - Se detecta actividad de WhatsApp y Telegram como servicios de mensajeria. - Se detecta actividad en Netflix, Youtube y Spotify como servicios multimedia.

Si bien es una prueba bajo un escenario desconocido, es decir, que no se sabe con exactitud qué eventos tuvieron lugar en la red; se puede afirmar lo siguiente (basándose en los datos que se obtuvieron preguntando a los usuarios de la red qué habían consumido y bajo qué dispositivo) :

- SonarWAN detectó correctamente los dispositivos que utilizaron la red.
- SonarWAN no detectó dispositivos extra.
- SonarWAN detectó los servicios de mensajería que se utilizaron.
- SonarWAN asoció correctamente aplicaciones a dispositivos
- SonarWAN detectó los servicios multimedia que los usuarios reportaron consumir.

Habiendo obtenido estos resultados, se estima un puntaje de 5 puntos (basándose en el sistema de puntaje implementado para las pruebas con escenarios conocidas).

Conclusiones de las pruebas

Lo más importante para destacar es que **los resultados de las pruebas fueron satisfactorios**. En términos generales, todos los *outputs* de SonarWAN ponían en evidencia el escenario descrito en la prueba, con más o menos exactitud dependiendo de la información disponible en la captura.

El sistema de puntaje arrojó un promedio de 7.28/10 para todas las pruebas de de capturas conocidas. Considerando la aclaración sobre el criterio de puntaje dependiendo del objetivo del sistema y no de las capacidades de SonarWAN, se puede concluir que el sistema propuesta responde con un alto nivel de efectividad al objetivo del proyecto.

No existieron situaciones en las pruebas bajo escenarios conocidos donde SonarWAN no funcionara de la manera esperada. Esta afirmación incluye a las pruebas en las que SonarWAN tuvo fallas en la detección. Estos casos de falla fueron:

- **Casos en los que había dispositivos clones.** Ya se sabia de antemano que la presencia de dispositivos clones, que son dispositivos exactamente iguales entre sí (modelo de hardware, versión del sistema operativo, build del sistema operativo, etc), presentaba una limitacion en SonarWAN. Para estos dispositivos, el comportamiento esperado de SonarWAN es que detecte solo uno de ellos. **Este comportamiento fue el observado en los *outputs* de las pruebas que contenían estos tipos de dispositivos**. Sin embargo, se probaron situaciones donde los dispositivos no eran clones exactos (pues variaban en alguna versión, por ejemplo 10.11.6 y 10.11.4) y SonarWAN pudo realizar la detección de una manera correcta, a pesar que existía una probabilidad alta que los considere dispositivos clones y solo detecte uno solo de ellos.

- **Casos en los que detecta un dispositivo incorrecto.** Hubo casos particulares, como fue la prueba de la “Kindle”, donde SonarWAN detecta un dispositivo iPhone. Como ya a esta altura se debe saber, la detección de dispositivos se hace a partir del tráfico HTTP. Luego de observar el tráfico de este dispositivo, se pudo observar que el *User-Agent* que aparecía en los *requests* del mismo era similar a los que aparecían en un navegador en un iPhone. Este evento vuelve a validar la afirmación anterior que expresa que en todos las pruebas SonarWAN se comportó según lo esperado.

Aparte de estos casos, vale la pena hacer mención a **algunas observaciones que rectifican la satisfacción de las pruebas**. Dichas observaciones son:

- **SonarWAN nunca detectó dispositivos extras en la red.** Es decir, dispositivos que no existían en la red.

- **SonarWAN nunco tuvo fallas internas,** o como se dice en el ambiente del *software*, nunca “rompió”.

- **SonarWAN siempre identifico los principales servicios que consumieron los dispositivos**, ya sea como servicios anónimos o servicios asignados al dispositivo correspondiente.

- **SonarWAN respondió sin problemas a la prueba de stress**, la cual representó las condiciones en las que se va a ejecutar el sistema normalmente. Pudo soportar una carga mayor al millón de paquetes en una captura mayor a 1 GB de tamaño.

- **SonarWAN tuvo tiempos de ejecución coherentes con el nivel de *performance* esperados**. Sus promedio fue de 0.3 segundos/100 paquetes.

Acerca del tráfico automático

Los *outputs* de las pruebas ejecutadas dejaron en evidencia la existencia (en abundancia) de lo que se denomina “tráfico automático”. **Este tráfico no tiene que ver con un consumo directo de un usuario, sino que es tráfico generado por servicios “secundarios” que son consumidos automáticamente cuando un usuario accede a determinados servicios.** Este tipo de tráfico tiene que ver generalmente con:

- **Librerías externas** que necesita una página, como por ejemplo tipografías de letra o archivos de estilo CSS.

- **Servicios de publicidad** que disponen de *banners* o avisos en las páginas que está visitando el usuario, como por ejemplo GoogleAds o Herolens.

- **Servicios de redes sociales** que buscan acceder a las redes sociales a las que el usuario está *logueado* para mostrar determinado contenido.

Por ejemplo, cuando un usuario accede a una página de noticias como puede ser “lanacion.com.ar”, no solo se detecta el servicio de la nacion, sino que SonarWAN también registra servicios consumidos con el dominio “ads.yahoo.com”, “connect.facebook.com”, “fonts.google.com”, “herolens.com”.

Este comportamiento presenta un dilema para el diseño de SonarWAN que consiste en determinar si dicho tráfico debe pertenecer al *output* del sistema o no. En el caso de no pertenecer, es necesario **desarrollar un mecanismo para poder filtrar este tráfico**.

Ahora bien, a menos que haya una base de datos que pueda determinar si un dominio pertenece a un servicio “automático”, **no se ha encontrado un método efectivo para filtrar este tráfico.** La métrica más coherente para el filtro sería la cantidad de *bytes* consumidos en ese servicio. En teoría, estos servicios “automáticos” deberían involucran a una cantidad de tráfico mucho menos que los servicios que si son consumidos por el

usuario. ráfico no es nada sencillo. Sin embargo, esta métrica no siempre es válida (considerar un usuario que entra a un servicio compuesto de una página que no posee demasiada información) y podría significar en el filtro incorrecto de servicios que si consumieron los usuarios y no son “automáticos”.

Al no poder encontrar un método efectivo para poder efectuar el filtro, **se ha optado porque SonarWAN incluya en el *output* a los servicios “automáticos”. SonarWAN, como sistema, va a incluir absolutamente todos los servicios que detecte**. **Será responsabilidad de los sistemas que consuman a SonarWAN como API filtrar este tipo de servicio si no desean incluirlo.**

Sección VI: **Conclusiones**

Palabras iniciales

Esta sección es la última sección del informe. Su objetivo es poder expresar ciertas reflexiones finales para concluir el trabajo. En primer lugar, resulta interesante y vale la pena mencionar ciertas ideas acerca de los cambios que está sufriendo la infraestructura de Internet con la transición a IPv6 y cual es el impacto que tienen estos cambios en las áreas que involucra este proyecto. Luego, para terminar, van a figurar las conclusiones acerca de SonarWAN como producto del trabajo de este proyecto, realizando un análisis acerca de sus ventajas, de sus desventajas, de las posibles extensiones, de las expectativas que se cumplieron y de las expectativas a futuro.

IPv6 y NAT

Resulta interesante reflexionar acerca del estado actual de Internet y las redes de computadoras; y poder analizar qué impacto tiene este estado sobre los temas que se estudiaron en este proyecto.

Actualmente, **tanto Internet como las redes de computadoras están sufriendo cambios causados por la transición de IPv4 a IPv6**. Esta transición no es una simple actualización de protocolos (como fue, por ejemplo, la aparición de TLS 1.0 para reemplazar a SSL 2.0), sino que **representa todo un cambio en la infraestructura de las redes.** El hecho de que cada dispositivo no tenga una sino hasta varias direcciones IP públicas cuestiona la existencia de tecnologías existentes que fueron creadas, en parte, por la poca disponibilidad pública de estas direcciones.

Una de estas tecnologías es la del mecanismo de NAT. Hoy en día no existe un acuerdo sobre el futuro de NAT con IPv6. A pesar de que existen opiniones a favor de la prevalencia de NAT independientemente de IPv6, **la tendencia en las opiniones leídas indica que NAT desaparecería con IPv6 pues el problema que intenta solucionar ya no es un problema**. Si deja de existir NAT con IPv6, las redes privadas dejarían de ser privadas y todos los nodos de Internet pasarían a ser “alcanzables” por cualquier otro nodo.

Con la desaparición de NAT, un proyecto de este tipo dejaría de tener las dificultades que se plantean en este informe. No habría dificultad alguna en identificar a un dispositivo en la red. Pero no solo identificar, sino también analizar su comportamiento y su actividad. Si bien no entra en el *scope* de este trabajo, **vale la pena preguntarse hasta qué punto el avance de IPv6 como nuevo estándar de Internet no representa una amenaza en lo que respecta a la privacidad de los usuarios.**

Conclusiones acerca de SonarWAN

Ventajas y desventajas del sistema desarrollado

La ventaja principal de SonarWAN es un sistema que **aprovecha la información de casi todos los paquetes disponibles en una captura**. Cada paquete, por más simple y poco informativo que sea (por ejemplo, un paquete SYN del protocolo TCP), entra en la lógica de relación y conexión de paquetes que implementa el sistema. Esto quiere decir que **SonarWAN tiene la potencia para poder extraer resultados significativos de todas las capturas que se analicen, independientemente de la cantidad o la calidad del tráfico que posea dichas capturas**. Este comportamiento le asigna la característica de **versátil**, en el sentido de poder adaptarse a cualquier tráfico que se presente a ser analizado.

SonarWAN posee otra ventaja importante que tiene un carácter más abstracto y tiene que ver con que **cada decisión de implementación tomada tiene un fuerte respaldo de estudio.** Por ejemplo, el hecho de poder tomar al *stream* como unidad de análisis o saber que campos inspeccionar a pesar del mecanismo de NAT surge del estudio exhaustivo de los estándares (RFC) de comportamiento para este tipo de redes. Es decir, **todo el diseño del sistema desarrollado tiene su base en los resultados de la etapa de investigación**.

Por otro lado, SonarWAN se destaca por ser *performante*. Esta característica proviene de que el sistema solo lee una sola vez cada paquete de la captura, extrayendo la información necesaria y ejecutando el procesamiento correspondiente. Los resultados que se obtuvieron en la sección anterior en términos de tiempo en función de cantidad de paquetes describen a un sistema ágil que puede soportar el análisis de una captura en vivo.

Por último, **el *core* de SonarWAN es un sistema independiente y autocontenido**. **Esto significa que puede integrarse sin ningún inconveniente a un sistema existente**, ya sea una interfaz gráfica que únicamente muestre los resultados del *core* de una manera más “amigable” o un sistema existente con otras funcionalidades que quiera agregar las funcionalidades de SonarWAN.

La desventaja principal es que la capacidad que tiene SonarWAN para detectar dispositivos y aplicaciones reside en tráfico HTTP. Esto presenta un desventaja teniendo en cuenta que este tráfico cada vez en menor al estar siendo reemplazado por tráfico HTTPS.

Futuras extensiones del sistema

Existen tres extensiones principales que quedan pendientes para realizarle a SonarWAN y que no pudieron ser implementadas por un tema de *scope* y de tiempo. Estas son:

- **Capacidad de realizar capturas en vivo.** La elección de las tecnologías a usar y el nivel de *performance* obtenido deja las puertas abiertas para que se pueda agregar en un futuro esta opción. La capacidad de ejecutar SonarWAN **en vivo** representa un aumento notable tanto la calidad del sistema como en su utilidad.

- **Implementar estrategias pendientes.** En la sección correspondiente a los resultados de la etapa de investigación se mencionan un conjunto de estrategias que no fueron implementadas aún. Sería interesante poder tener la oportunidad de implementarlas y agregarlas a SonarWAN para poder agregarle mayor poder deductivo.

- **Investigar acerca de estrategias matemáticas.** Existe hoy en dia un desarrollo y un fomento muy grande en el campo de la inteligencia artificial, que propone la aplicación de métodos matemáticos (como por ejemplo estrategias de *machine learning*, redes neuronales, etc) para la resolución de ciertos problemas donde no existe un algoritmo exacto de solución. Sería interesante poder investigar de qué manera se pueden introducir estas técnicas en SonarWAN.

Expectativas cumplidas

SonarWAN fue el resultado de un largo trabajo, tanto de investigación y de desarrollo. Especialmente al inicio del proyecto, hubieron momentos en que realmente se puso en duda la viabilidad de construir un sistema como el que se desarrolló. De hecho la consigna del proyecto no fue inicialmente la construcción del sistema, sino que fue estudiar y evaluar la posibilidad de la construcción de un sistema de estas características. La falta de material de investigación y de estrategias disponibles asignaron un grado de dificultad mayor a este proyecto.

Habiendo terminado de desarrollar SonarWAN, se puede decir que se cumplieron las expectativas planteadas. **Se llegó a un sistema que cumple con el objetivo planteado al inicio del proyecto, es decir, que es capaz de caracterizar al escenario que se manifiesta en una red LAN a partir del análisis de tráfico de dicha red capturado desde la interfaz WAN del *gateway* NAT.**

Expectativas futuras

SonarWAN tiene muchas cosas para agregar y para ajustar. Parte de haber optado por una filosofía de “Código Abierto” es la capacidad de seguir el desarrollo de SonarWAN de una manera colaborativa con la comunidad de desarrolladores que estén interesados en el proyecto. **Como expectativa a futuro, la idea es poder lograr que SonarWAN tenga difusión dentro de la comunidad y que su desarrollo se mantenga activo de una manera colaborativa.**

Anexo

Manual de SonarWAN *Core*

Requisitos

- **Python 3** (se puede descargar de www.python.org/downloads/)
- **Virtualenv** como manejador de paquetes (página oficial, virtualenv.pypa.io/en/stable/)
- **Git** para clonar el repositorio (página oficial, https://git-scm.com/)

Instalación

A continuación se detallan los pasos para instalar el sistema:

1) Descargarse los códigos fuente. Para esto, ejecutar:

```
$ git clone https://github.com/SonarWAN/sonarwan-core.git
```

2) Dirigirse a la carpeta donde se encuentran los códigos fuente descargados en el paso anterior.

```
$ cd path/to/sonarwan-core
```

3) Crear el entorno *virtualenv* ejecutando:

```
$ virtualenv -p path/to/python3 env
```

***Nota**: para obtener el path a python3, se puede ejecutar "which python3"*

4) Activar el entorno. Para esto, ejecutar:

```
$ source env/bin/activate
```

5) Instalar las dependencias. Correr:

```
$ pip install -r requirements.txt
```

Uso

IMPORTANTE: Para utilizar SonarWAN siempre se debe activar el entorno *virtualenv*. Para esto, ejecutar:

```
$ source env/bin/activate
```

en la raíz del directorio del sonarwan-core.

SonarWAN se ejecuta de la siguiente manera:

```
$ python sonarwan/main.py    files    [parameters]
```

Donde,

- *"files"* contiene una lista de *paths* a archivos de captura, separados por un espacio.
- "*parameters"* pueden ser:
    - ***"-p FILE"*** o ***"--patterns FILE"***: para agregar un archivo ".patterns" a SonarWAN. "FILE" representa el *path* a dicho archivo.
    - ***"-s DIR"*** o ***"--services DIR"***: para agregar un directorio de archivos *yaml* para la detección de servicios. "DIR" representa el *path* al directorio.
    - ***"-i DIR"*** o ***"--inference DIR"***: para agregar un directorio de archivos *csv* para el motor de inferencia. "DIR" representa el *path* al directorio.
    - ***"-j"*** o ***"--json"***: para que el *output* sea en formato JSON por salida estándar. Ideal para consumir al *core* como API. Por default, el *output* es en texto.
    - ***"--progress"*****:** funciona únicamente con el parámetro "-j". Muestra el progreso del análisis.
    - ***"-o FILE"*** o ***"--output FILE"***: funciona únicamente cuando el parámetro "-j" NO figura. Sirve para especificar el archivo donde aparecerá la salida en formato texto. En el caso de no especificarse, se imprime la salida en salida estándar.

Para más información, ejecutar:

```
$ python sonarwan/main.py  --help
```

Ejemplos de corridas

```
$ python sonarwan/main.py  ../datasets/1.pcapng -o ~/Desktop.report.txt
```

```
$ python sonarwan/main.py  ../datasets/1.pcapng ../datasets/2.pcapng --json --patterns
~/Desktop/extra.patterns --services ~/Desktop/extra_services/
```

```
$ python sonarwan/main.py  ../datasets/capura.pcap -j --progress
```

Ejemplos de *outputs* en formato texto

```
*******
SUMMARY
*******
SonarWAN found 1 devices and 2 authorless services in 1 capture files.
188 packets were analyzed.
Execution time: 0.7967720031738281

*******
DETAILS
*******

*******
Devices
*******

Device 1
========

Characteristics:
----------------  -------
os_build           12A365
brand             Apple
darwin_version     14.0.0
os_brand           Apple
os_version         8.0
os_family          iOS
model             iPad
family            iPad
version           4,4
cfnetwork_version  711.0.6
Associated Apps    1
Unasigned Services  0
----------------  -------

App 1:
----------------  -------------------------
name               Mobile Safari UI/WKWebView
Associated Services  1
----------------  -------------------------

Service 1:
-----  ----------------------------
name   Apple
type   Enterprise
ips    {'181.30.131.33'}
hosts  {'init-p01st.push.apple.com'}
-----  ----------------------------

Activity:
----------------  -----
2016-09-06T15:38:14  10162
----------------  -----
```

```
******************
Authorless Services
******************

Service 1:
-----  ----------------------------
name   Apple
type   Enterprise
ips    {'17.188.145.178'}
hosts  {'23-courier.push.apple.com'}
-----  ----------------------------

Activity:
-------------------  ----
2016-09-06T15:38:17   437
2016-09-06T15:38:16   596
2016-09-06T15:38:18  3130
2016-09-06T15:38:15  5880
2016-09-06T15:38:14   234
-------------------  ----
Service 2:
-----  -------------------------
name   thinkdifferent
type   Generic
ips    {'23.197.215.110'}
hosts  {'www.thinkdifferent.us'}
-----  -------------------------

Activity:
-------------------  ----
2016-09-06T15:38:14  1144
-------------------  ----
```

Manual de SonarWAN *GUI*

Requisitos

- **npm** >= 3.10 como manejador de paquetes (página oficial, https://www.npmjs.com/)
- **Git** para clonar el repositorio (página oficial, https://git-scm.com/)
- **SonarWAN *Core*** (instrucciones para su instalación figuran en el documento anterior)

Instalación

A continuación se detallan los pasos para instalar el sistema:

1) Descargarse los códigos fuente. Para esto, ejecutar:

```
$ git clone https://github.com/SonarWAN/sonarwan-gui.git
```

2) Dirigirse a la carpeta donde se encuentran los códigos fuente descargados en el paso anterior.

```
$ cd path/to/sonarwan-gui
```

3) Instalar las dependencias. Para esto, ejecutar:

```
$ npm install
```

4) Compilar el código.

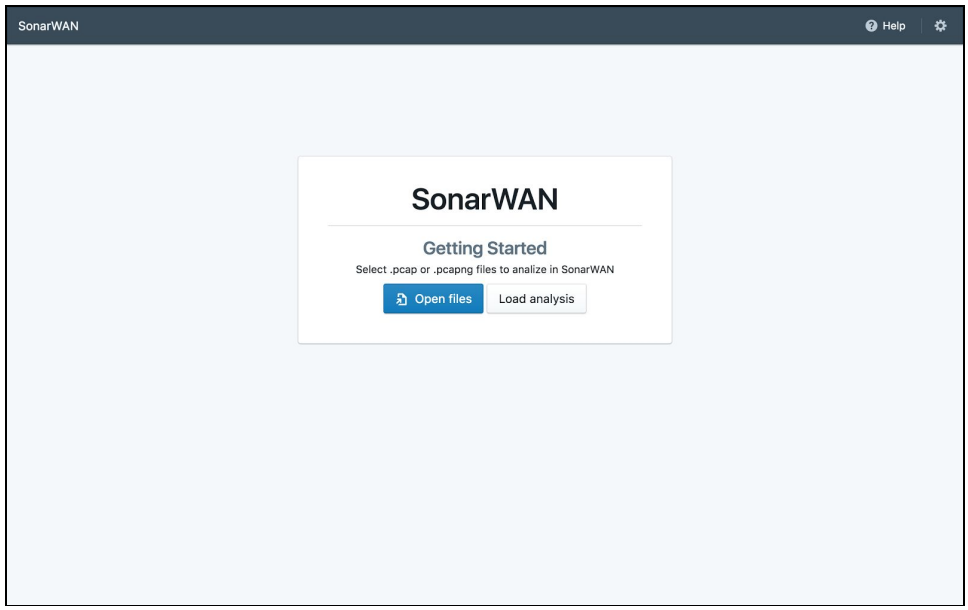
```
$ npm run compile
```

Uso

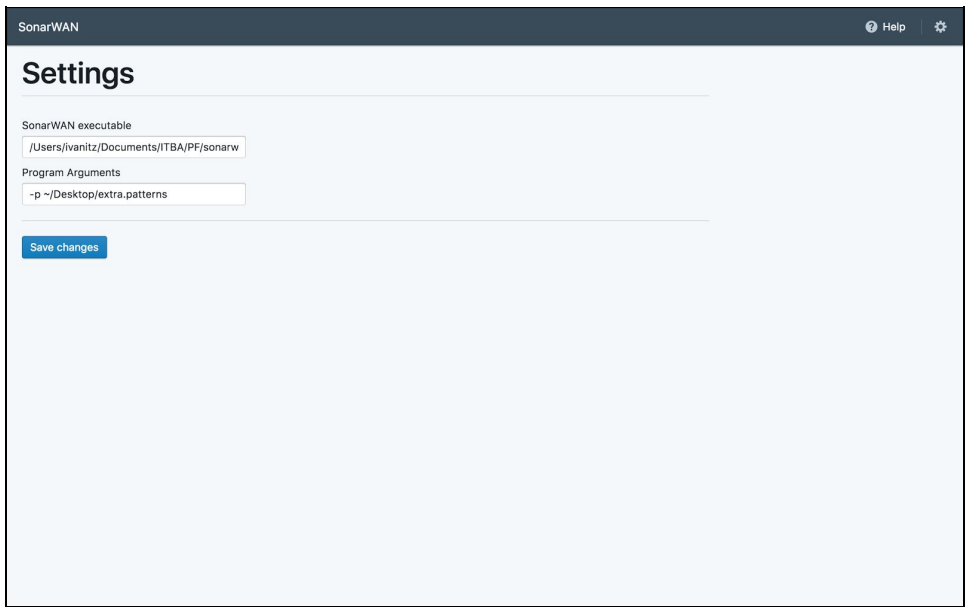
En la carpeta donde se encuentra el repositorio, ejecutar:

```
$ npm start
```

Se debería ver la siguiente pantalla:

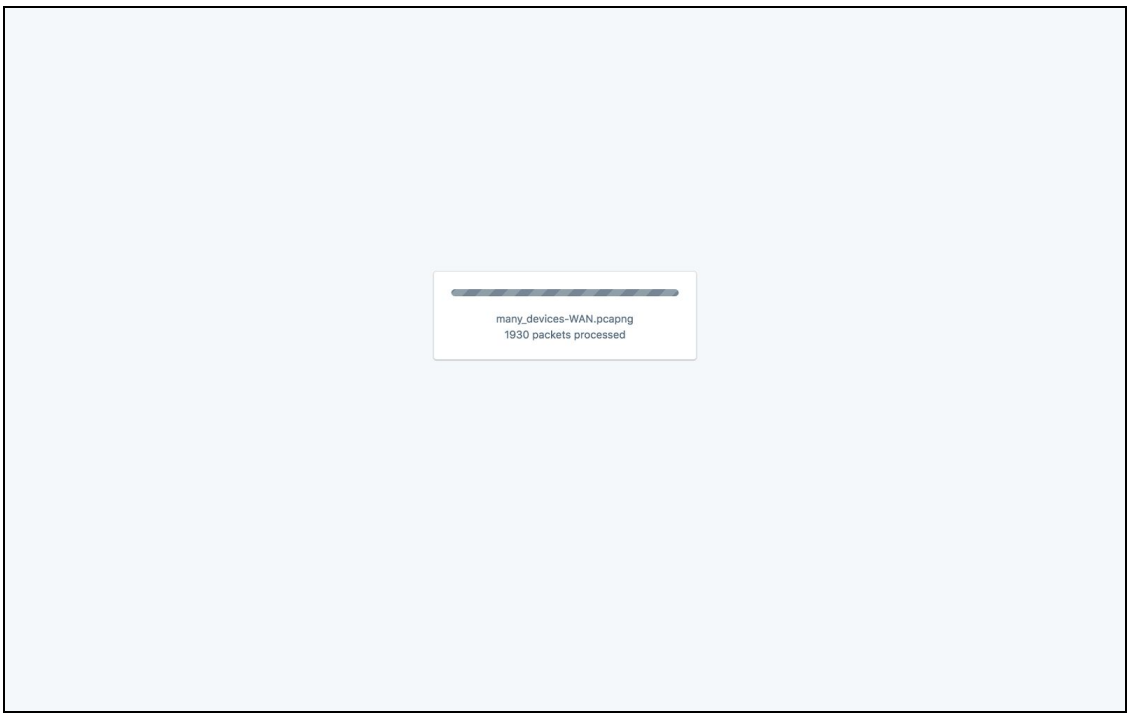


Para ejecutarlo, primero es necesario especificar el *path* al *core*. Para esto, clickear el botón de configuración (arriba a la derecha). Debería ver una imagen como la siguiente:

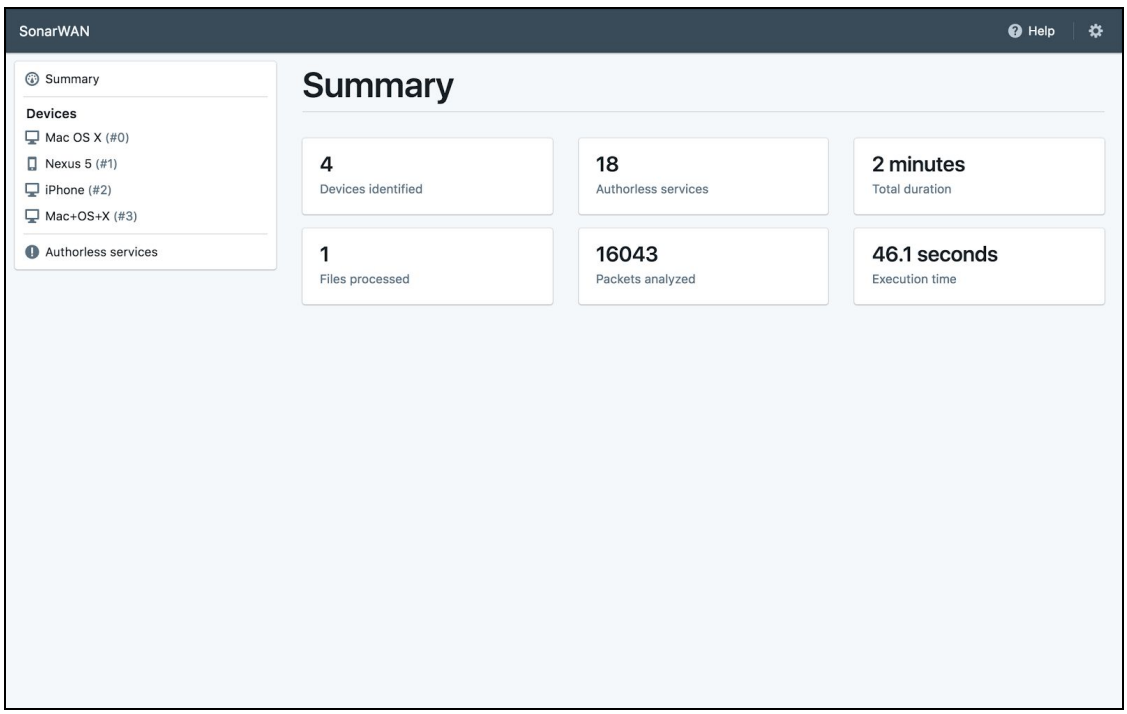


Aquí se debe setear el *path* al *core* (específicamente al archivo *bin/sonarwan* disponible en el repositorio del *core*). También es posible agregarle los parámetros al *core* (por ejemplo '-p PATTERNS'). Al finalizar, guardar los cambios y volver al inicio clickeando el logo 'SonarWAN'.

Una vez en el inicio, seleccionar los archivos a analizar. Debería comenzar a correr el *core* del sistema. La pantalla que debería ver es la siguiente, donde se muestra el progreso del procesamiento:



Cuando finalice, se va a mostrar la siguiente pantalla con un resumen del *output*:



Seleccionando un dispositivo o lo *authorless services* (servicios anónimos), se puede observar su detalle. Por ejemplo:


SonarWAN
Help
Summary
Devices
Mac OS X (#0)
Nexus 5 (#1)
iPhone (#2)
Mac+OS+X (#3)
Authorless services
Mac OS X (#0)
Network I/O
21 kB
20 kB
18 kB
16 kB
14 kB
12 kB
10 kB
8 kB
6 kB
4 kB
2 kB
0 B
:30
:45
05:04
:15
:30
:45
Characteristics
Key
Value
brand
Apple
cfnetwork_version
720.3.13
darwin_version
14.3.0
os_brand
Apple
os_family
Mac OS X
os_version
10.10.3
Apps
Safari
Key
Value

Resultados Pruebas de capturas conocidas - 1 dispositivo

Prueba	Complejidad	Escenario			Archivos		Paquetes Analizados	Tiempo de Ejecucion [seg]	Output	
		Dispositivo	OS	Uso	Cantidad	Tamano			Descripcion	Puntaje
1	Baja	LG Nexus 5	Android 6.0.1	Unicamente ingresa a la red	1	41 KB	121	2,15	- Detecta correctamente al dispositivo con sus caracteristicas. - No detecta dispositivos extras. - Detecta consumo de Telegram como servicio anonimo, que esta instalado en dicho dispositivo.	5
2	Baja	LG Nexus 5	Android 6.0.1	Ya pertenecciente a la red. Utilizad Telegram y WhatsApp	2	19 KB	129	1,5	- Detecta ambos servicios de mensajeria como servicios anonimos. - No detecta ningun dispositivo (no hay trafico HTTP plano, todo encriptado)	2,5
3	Baja	LG Nexus 5	Android 6.0.1	Ya pertenecciente a la red. Utiliza la aplicacion de La Nacion	1	931 KB	1163	4,57	- Detecta correctamente al dispositivo con sus caracteristicas. - Detecta el consumo del servicio La Nacion asociado al dispositivo (pero no a una aplicacion) - No detecta dispositivos extras. - Detecta consumo de Telegram y WhatsApp como servicios anonimos, ambas apps instaladas en el dispositivo.	5
4	Baja	iPad 4.4	iOS 8.0	Unicamente ingresa a la red	1	40 KB	273	1,99	- Detecta correctamente al dispositivo con sus caracteristicas. - No detecta dispositivos extras.	5,5
5	Baja	iPad 4.4	iOS 8.0	Ya pertenecciente a la red. Utiliza la aplicacion de La Nacion y el juego Clash Of Clans	2	1.8 MB	2437	9,47	- Detecta correctamente al dispositivo con sus caracteristicas. - No detecta dispositivos extras. - Detecta correctamente ambas aplicaciones asociadas al dispositivo	5,5
6	Media	MacBook Pro	OSX 10.11.6	Ya pertenecciente a la red. Navega unicamente por paginas con HTTPS.	1	1.1 MB	2146	7,66	- No detecta ningun dispositivo. - Detecta correctamente algunas paginas HTTPS como servicios anonimos	2,5
7	Media - Baja	MacBook Pro	OSX 10.11.6	Ya pertenecciente a la red. Navega con un Chrome y streamea musica con Spotify	1	11.1 MB	9937	32,6	- Detecta correctamente al dispositivo. - No detecta dispositivos extra. - Identifica al Chrome como aplicacion del dispositivo. - Identifica a Spotify como servicio anonimo - Detecta correctamente algunas paginas HTTPS como servicios anonimos	4,5
8	Baja	Dell	Windows 10	Ingresa a la red y navega con un Firefox	1	3.6 MB	5548	17,24	- Detecta correctamente al dispositivo. - No detecta dispositivos extra. - Identifica al Firefox como aplicacion del dispositivo. - Detecta todas las paginas HTTP consumidas con el Firefox	5
9	Baja	Kindle Paperwhite	Kindle	Ingresa a la red y entra a la tienda de libros	1	317 KB	588	2,21	- Detecta un iPhone como unico dispositivo. - Identifica servicios anonimos o asociados al iPhone de Amazon	3,5
10	Baja	iPhone 8.4	iOS 9.3.1	Ingresa a la red unicamente	1	1.7 MB	4831	20,76	- Detecta correctamente al dispositivo con sus caracteristicas. - No detecta dispositivos extra. - Identifica algunas aplicaciones pero que no son de usuario (securityId, etc) - Detecta el consumo de servicios de Apple	5

Resultados Pruebas de capturas conocidas - 2 dispositivos

Prueba	Complejidad	Escenario			Archivos		Paquetes Analizados	Tiempo de Ejecucion [seg]	Output	
		Dispositivo	OS	Uso	Cantidad	Tamano			Descripcion	Puntaje
1	Media	Nexus 5	Android 6.0.1	Ingresa a la red y usa la app de La Nacion	1	35.5 MB	35080	135,45	- Identifica correctamente a los 2 dispositivos. - No detecta dispositivos extra. - Identifica a La Nacion como servicio del Nexus, pero no asociado a una aplicacion. - Identifica netflix como aplicacion en el iPad, pero identifica otras aplicaciones desconocidas. - Identifica el streaming propio del video de netflix como servicio anonimo.	4,5
		iPad 4.4	iOS 8.0	Ya pertenece a la red. Se encuentra viendo una pelicula en la app de Netflix						
2	Madia - Alta	MacBook Pro	OSX 10.11.6	Ya pertenece a la red. Se encuentra viendo videos en YouTube por Chrome	2	10,5 MB	12149	25,06	- Identifica correctamente a los 2 dispositivos a pesar de su similitud. - No detecta dispositivos extra. - Identifica al Chrome en uno de las Mac. - Identifica a YouTube pero como servicio anonimo, no asociado al Chrome	4,5
		MacBook Pro	OSX 10.11.4	Ingresa a la red unicamente						
3	Media - Baja	Dell	Windows 10	Ya pertenece a la red. Navega utilizando Firefox	2	4.1 MB	7098	21,77	- Identifica correctamente a los 2 dispositivos. - No detecta dispositivos extra. - Identifica correctamente los navegadores asociados a los dispositivos. - Identifica las paginas visitadas, algunas como servicios anonimos.	5,5
		Dell	Ubuntu 16.04	Ingresa a la red. Navega utilizando Chrome						
4	Media - Baja	Nexus 5	Android 6.0.1	Ya pertenece a la red. Envia mensajes por WhatsApp	2	1,7 MB	4894	21,77	- Identifica unicamente al iPhone. - El Nexus no es detectado pues no tiene trafico HTTP. - Identifica ciertas aplicaciones de*background* del iPhone. - Identifica a WhatsApp como servicios anonimo	2,5
		iPhone 8.4	iOS 9.3.1	Ya perteneciente a la red pero sin actividad de usuario.						
5	Alta	iPad 4.4	iOS 8.0	Ya pertenece a la red. Utiliza la aplicacion de La Nacion	1	26.8 MB	21349	162,43	- Identifica correctamente a los 2 dispositivos a pesar de su similitud. - No detecta dispositivos extra. - Identifica correctamente a la aplicacion de La Nacion correspondiente al primer iPad. - Identifica netflix como servicio anonimo tambien - Identifica a Netflix pero bajo una IP, no detecta el nombre del servicio	5
		iPad 4.4	iOS 7.1	Ya perteneciente a la red. Se encuentrea viendo una pelicula en Netflix.						

Resultados Pruebas de capturas conocidas - 3 o más dispositivos

Prueba	Complejidad	Escenario			Archivos		Paquetes Analizados	Tiempo de Ejecucion [seg]	Output	
		Dispositivo	OS	Uso	Cantidad	Tamano			Descripcion	Puntaje
1	Alta	MacBook Pro	OSX 10.11.6	Ingresa a la red y navega por Internet con Chrome	1	19.9 MB	23935	64.78	- Detecta una MacBookPro y un iPhone. SON DISPOSITIVOS CLONES! - Detecta el Chrome asociado a la MacBookPro. - Agrupa el trafico de las MacBooks al de la unica MacBook y lo mismo con el de los iPhones .	3
		MacBook Pro	OSX 10.11.6	Ingresa a la red y navega por Internet con Chrome						
		iPhone 6	iOS 10.0.2	Ingresa a la red y se mantiene inactivo						
		iPhone 5	iOS 10.0.2	Ya pertenece a la red. Se mantinene inactivo						
2	Media - Alta	MacBook Pro	OSX 10.10.3	Perteneciente a la red. Navega con Safari	1	12.8 MB	16043	50,13	- Detecta los 4 dispositivos correctamente. - No detecta dispositivos extra. - Asocia correctamente el Safari a la primera MacBookPro.	5,5
		Nexus 5	OSX 10.11.6	Ingresa a la red y se mantiene inactivo						
		iPhone 6	iOS 10.0.2	Ingresa a la red y se mantiene inactivo						
		MacBook Pro	OSX 10.11.4	Ya pertenece a la red. Se mantinene inactivo						
3	Alta	iPad 4.4	iOS 7.0	Perteneciente a la red. Ve videos en YouTube	1	26.6 MB	27611	72,12	- Detecta los 5 dispositivos correctamente. - Detecta al iPod Touch como iPhone. - No detecta dispositivos extra. - Detecta el Edge en Windows 10. - Detecta la aplicacion de La Nacion en el iPad. - Detecta el Safari en el iPod Touch. - Detecta WhatsApp como servicio anonimo.	4,5
		Ipad 4.4	iOS 8.1	Perteneciente a la red. Ejecuta la aplicacion de La Nacion						
		Nexus 5	Android 6.0.1	Ingresa a la red y habla por WhatsApp						
		iPod Touch	iOS 9.1	Ya pertenece a la red. Navega con Safari						
		Asus	Windows 10	Ingresa a la red. Navega con Edge.						

Bibliografía

Papers consultados:

- “A Technique for Counting NATted Hosts”. Bellovin. 2002.
- “Characteristics of Wide-Area TCP/IP Conversations”. Caceres, Danzig, Jamin, Mitzel. 1991.
- “Source attribution for network address translated forensic captures”. Cohen. 2009.
- “Network Forensics Analysis”. Corey. 2002
- “Network Address Translation (NAT) Behaviour: Final report”. Zincir-Heywood, Gokcen, Aghaevi, Howes. 2014.
- “‘No NAT’d User left Behind’: Fingerprinting Users behind NAT from NetFlow Records alone”. Vincenzo Verde, Ateniese,Gabrielli,Mancini, Spognardi. 2014.
- “Passive Network Forensics: Behavioural Classification of Network Hosts Based on Connection Patterns”. McHugh, McLeod, Nagaonkar. 2008.
- “Network forensic frameworks: Survey and research challenges”. Pilli, Joshi, Nigoyi. 2010.
- “A Robust Classifier for Passive TCP/IP Fingerprinting”. Beverly. 2004.

Recursos Web para la Sección I:

- https://www.tlm.unavarra.es/file.php/86/rc1.jpg
- http://blyx.com/public/docs/pila_OSI.pdf
- https://es.wikipedia.org/wiki/Protocolo_de_comunicaciones
- https://www.moma.org/collection/works/110263
- https://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg
- https://tools.ietf.org/html/rfc2663

Recursos Web para la Sección III:

- https://en.wikipedia.org/wiki/Idle_scan
- https://www.ietf.org/rfc/rfc2616.txt
- https://tools.ietf.org/html/rfc5246
- https://es.wikipedia.org/wiki/Transport_Layer_Security#Handshake_b.C3.A1sico
- https://tools.ietf.org/html/rfc3986
- https://tools.ietf.org/html/rfc1035
- https://tools.ietf.org/html/rfc2663
- https://tools.ietf.org/html/rfc3022
- https://tools.ietf.org/html/rfc2766
- https://tools.ietf.org/html/rfc5382
- https://tools.ietf.org/html/rfc4787

Recursos Web para la Sección IV:

- http://pypy.org/
- https://github.com/KimiNewt/pyshark
- https://docs.python.org/3/
- https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html
- https://github.com/serbanghita/Mobile-Detect
- https://github.com/ua-parser/uap-python
- https://github.com/anvileight/pymobiledetect
- http://electron.atom.io/