UNIVERSITY OF VALLADOLID
E.T.S.I. TELECOMUNICACIÓN

*Master Thesis*

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

# *Teaching Networking, Hands-On Labs*

*Author:*

***Mr. Bruno Olivar Trinchet***

*Tutor:*

***Prof. Dr.-Ing. Udo Garmann***

*Deggendorf, 3rd February 2015*

| | |
|---|---|
| *TITLE:* | ***Teaching Networking, Hands-on Labs*** |
| *AUTHOR:* | ***Mr. Bruno Olivar Trinchet*** |
| *TUTOR:* | ***Prof. Dr.-Ing. Udo Garmann*** |
| *DEPARTAMENT:* | ***Elektro- und Medientechnik*** |
| | ***Technische Hochschule Deggendorf*** |

***TRIBUNAL***

| | |
|---|---|
| *PRESIDENT:* | |
| *VOCAL:* | |
| *SECRETARY:* | |
| *Date:* | ***3rd February 2015*** |
| *Qualification:* | |

# Abstract and Key Words

This project has been focused on the design and implementation of a hands-on laboratory, where students would be able understand and learn the different network protocols operations. Once dealt with the installation, we have done some tests focused on Wide Area Network emulation and, as a further step, the results have been analyzed and discussed.

In order to complete said tests, we have implemented some network traffic analysis tools, such as *Traceroute*, *Ping* or *Wireshark*. The latter in particular, due to its ease to capture, show and filter traffic from many different network protocols.

First of all, the theoretical basic concepts, essential to understand network protocols operation, have been properly described.

Then, once explained the theoretical fundamentals, the entire laboratory installation has been developed; starting with the hardware components assembly and then following with the software part installation. This second section also includes the setup of each service or protocol needed for the proper network operation.

Finally, in the last main section, tests performed in the laboratory have been detailed, once fully configured the network. These tests have been focused on Wide Area Network emulation through the WANem software tool, through which we have introduced some delays and packet loss into our network. At the same time, with the above mentioned network tools (*Traceroute, Ping or Wireshark*) we have tested protocols' behaviour when experiencing these delays or packet loss.

Key Words: Network, Protocols, Laboratory, Emulation, WANem, Wireshark, TCP/IP.

# Resumen y Palabras Clave

Este proyecto se ha centrado en el diseño e implementación de un laboratorio de docencia en el que los alumnos puedan comprender y estudiar el funcionamiento de los diferentes protocolos de red. Tras la instalación se han realizado diferentes pruebas, centrándose en la emulación de red de área amplia (*WAN – Wide Area Network*) para, posteriormente, analizar los resultados de dichas pruebas.

Para llevar a cabo las diferentes pruebas se han empleado herramientas de análisis, tales como Traceroute, Ping o Wireshark, esta última en mayor medida, ya que permite capturar tráfico de una gran cantidad de protocolos y mostrar por pantalla el contenido del tráfico capturado.

Primeramente, se han descrito los aspectos teóricos fundamentales, necesarios para comprender el funcionamiento de los protocolos empleados.

A continuación, una vez sentada la base teórica esencial, se ha desarrollado al completo la implementación del laboratorio, comenzando por el montaje de la parte hardware y siguiendo con la instalación de la parte software, tanto de las herramientas de análisis de tráfico como los servicios o protocolos fundamentales para el correcto funcionamiento de la red.

Finalmente, en el último bloque principal se detallan las pruebas realizadas en el laboratorio, una vez que la red ha sido puesta en funcionamiento. Estas pruebas han girado entorno a la emulación de red de área amplia (*WAN – Wide Area Network*) mediante la herramienta WANem, a través de este software se han introducido en la red retardos y pérdidas de paquetes. A su vez, mediante las herramientas mencionadas anteriormente, se ha analizado el comportamiento de los diferentes protocolos de red (IP, TCP o HTTP) al sufrir estos retardos y pérdidas de paquetes.

Palabras clave: Redes, Protocolos, Laboratorio, Emulación, WANem, Wireshark, TCP/IP.

*Acknowledgements*

*First and foremost, I have to thank my thesis tutor, Mr. Udo Garmann. Without his trust placed on me, this project would have never been accomplished.*

*Most importantly, none of this could have happened without my family. To my parents and my sister – Thanks for being always positive, for being by my side whenever I needed.*

*I would also like to show gratitude to Ramón J. Durán to make possible this great Erasmus experience.*

*Finally, I would never have arrived here without help and support of my best colleague Nadia Nohales.*

*“Innovation distinguishes between a leader and a follower”*
*Steve Jobs*

# 1. – Introduction

## 1.1. – Motivation and Description

Nowadays, network related subjects need hands-on laboratories to learn and understand protocols' behaviour. To comprehend these protocols, we need to create a suitable environment where to test and verify how they behave under different conditions. The environment which will be implemented, enables us to modify networks' conditions, like delay or packet loss.

The best way to modify these networks' conditions is by using a software tool, which will allow us to do it in a simple way; this tool is a Wide Area Network Emulator (also known as WANem) and it is distributed as a modified Linux-based Operative System.

Once these elements are modified, some tests will be done to analyze the strength of our network. Through these modifications we implement a Wide Area Network into our Local Area Network, so we will be able to emulate a real-life situation where some packets are delayed and other packets get lost.

This kind of emulation could be really useful for teaching purposes, but also for other areas like software developing. When a new webpage or application is launched, developers often want to test its behaviour in an unusual situation, like a delayed connection between server and client or a connection where some data packets are randomly lost.

## 1.2. – Project goals

The main goal will be to **build a complete environment for the students**, enabling them to:

- Obtain hands-on learning about networking protocols.
- Emulate a real Wide Area Network with delays and packet loss as well as to check out how these protocols behave under different network conditions.
- Perform network tests and analyze their results, making use of network tools, such as *Traceroute*, *Ping* and *Wireshark*.

This project could also have another secondary purpose:

- Learn about how to design and implement a hands-on teaching-networking laboratory, assembling all the essential hardware and installing all the needed software to have a Linux-based environment working properly.
- Review some theoretical concepts related to basic network protocols, which are essential to build an operating Local Area Network.

## 1.3. – Structure

The project will be split in three main sections. The first one will be focused on the main network protocols and its theoretical concepts. The second section aims to explain the laboratory installation, starting by the hardware components assembly and following by the software part installation. This section also includes the set up of every server, service or protocol needed for the proper network operation. Finally, in the third main part, network tests will be performed and the analysis of results will be discussed.

Following to the aforementioned three main sections, some conclusions drawn from the findings are presented and they will be focused on the whole thesis. We will also discuss some future lines, which could be developed in an upcoming complementary project, with some additional network tests and their appropriate analysis of results.

# 2. – Theoretical Concepts

*This first section aims to explain the main protocols and its theoretical concepts. We need to understand these protocols and then, in the following sections, we will be able to test and analyze them. We will describe all the basic protocols, even ARP and DHCP, although these protocols are not going to be tested later, but they are essential for network's operation. The other protocols will be IP, TCP, DNS and HTTP.*

## 2.1. – Introduction

### 2.1.1. – Architectural Principles

The TCP/IP protocol suite allows computers, smartphones, and embedded devices of all sizes to communicate with each other. It forms the basis for what is called the global Internet, or the Internet, a wide area network (WAN) of about two billion. Although many people consider the Internet and the World Wide Web (WWW) to be interchangeable terms, we ordinarily refer to the Internet in terms of its ability to provide basic communication of messages between computers. We refer to WWW as an application that uses the Internet for communication. It is perhaps the most important Internet application.

### 2.1.2. – Design and Implementation

Although a protocol architecture may suggest a certain approach to implementation, it usually does not include a mandate. Consequently, we make a distinction between the protocol architecture and the implementation architecture, which defines how the concepts in a protocol architecture may be implemented.

Design philosophy for networking protocols involves multiple layers of implementation (and design). This approach is now called layering and is the usual approach to implementing protocol suites. With layering, each layer is responsible for a different facet of the communications. Layers are beneficial because a layered design allows developers to evolve different portions of the system separately, often by different people with somewhat different areas of expertise.

#### OSI Model

The most frequently mentioned concept of protocol layering is based on a standard called the Open Systems Interconnection (OSI) model as defined by the International Organization for Standardization (ISO). Figure 1 shows the standard OSI layers, including their names, numbers, and a few examples. The Internet's layering model is somewhat simpler, as we will see later.

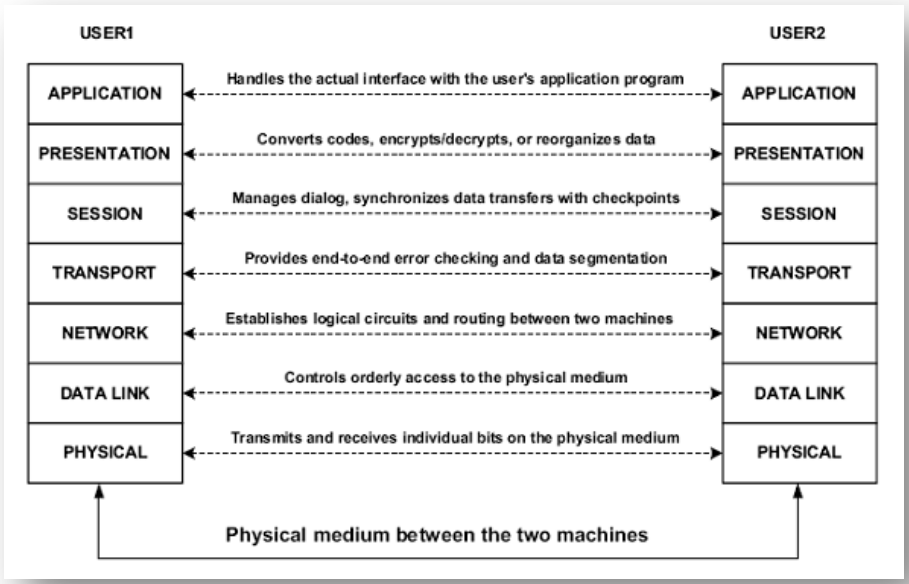

USER1
USER2
APPLICATION
PRESENTATION
SESSION
TRANSPORT
NETWORK
DATA LINK
PHYSICAL
Handles the actual interface with the user's application program
Converts codes, encrypts/decrypts, or reorganizes data
Manages dialog, synchronizes data transfers with checkpoints
Provides end-to-end error checking and data segmentation
Establishes logical circuits and routing between two machines
Controls orderly access to the physical medium
Transmits and receives individual bits on the physical medium
Physical medium between the two machines

Figure 1. OSI Model [1]

Although the OSI model suggests that seven logical layers may be desirable for modularity of a protocol architecture implementation, the TCP/IP architecture is normally considered to consist of five. There was much debate during the early 1970s, but finally TCP/IP "won".

## TCP/IP Model

Figure 2 depicts the layering inspired by the ARPANET reference model, which was ultimately adopted by the TCP/IP suite. The structure is simpler than the OSI model, but real implementations include a few specialized protocols that do not fit cleanly into the conventional layers.

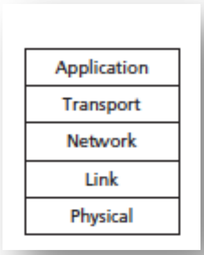


Figure 2. TCP/IP Model

Starting from the top of Figure 2, we could see the Application layer. The application layer is where network applications and their application-layer protocols reside. The Internet's application layer includes many protocols, such as the HTTP protocol (which provides for Web document request and transfer), SMTP (which provides for the transfer of e-mail messages), and FTP (which provides for the transfer of files between two end systems). We'll see that certain network functions, such as the translation of human-friendly names for Internet end systems like *www.example.com* to a 32-bit network address, are also done with the help of a specific application-layer protocol, the domain name system (DNS).

Then we have the Transport layer, which transports application-layer messages between application endpoints. In the Internet there are two transport protocols, TCP and UDP, either of which can transport application-layer messages. TCP provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control (that is, sender/receiver speed matching). TCP also breaks long messages into shorter segments and provides a congestion-control mechanism, so that a source throttles its transmission rate when the network is congested.

The UDP protocol provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control, and no congestion control. We will refer to a transport-layer packet as a segment.

After that we could find the Internet's network layer, which is responsible for moving network-layer packets (known as datagrams) from one host to another. The Internet transport-layer protocol (TCP or UDP) sends a transport-layer segment and a destination address to the network layer. The network layer provides the service of delivering the segment to the transport layer in the destination host. This layer includes the IP Protocol, which defines the fields in the datagram. There is only one IP protocol, and all Internet components that have a network layer must run the IP protocol.

The Internet's network layer routes a datagram through a series of routers between the source and destination. To move a packet from one node (host or router) to the next node in the route, the network layer relies on the services of the link layer. In particular, at each node, the network layer sends the datagram down to the link layer, which delivers the datagram to the next node along the route.

With TCP/IP, each link-layer interface on each computer (including routers) has at least one IP address. IP addresses are enough to identify a host, but they are not very convenient for humans to remember or manipulate (especially the long addresses used with IPv6). In the TCP/IP world, the DNS is a distributed database that provides the mapping between host names and IP addresses (and vice versa).

While the job of the link layer is to move entire frames from one network element to another network element, the job of the physical layer is to move the individual bits into the frame from one node to the next. The protocols in this layer depend again on the actual transmission medium of the link (for example, twisted-pair copper wire, single-mode fiber optics).

## 2.2. – Basic IP Address Structure

### 2.2.1. – Classful Addressing

IPv4 has 4,294,967,296 possible addresses in its address space and because of the large number of addresses, it is convenient to divide the address space into different classes. IP addresses are grouped by type and size. Most of the IPv4 address groups are subdivided to a single address and used to identify a single network interface of a computer. These addresses are called unicast addresses.

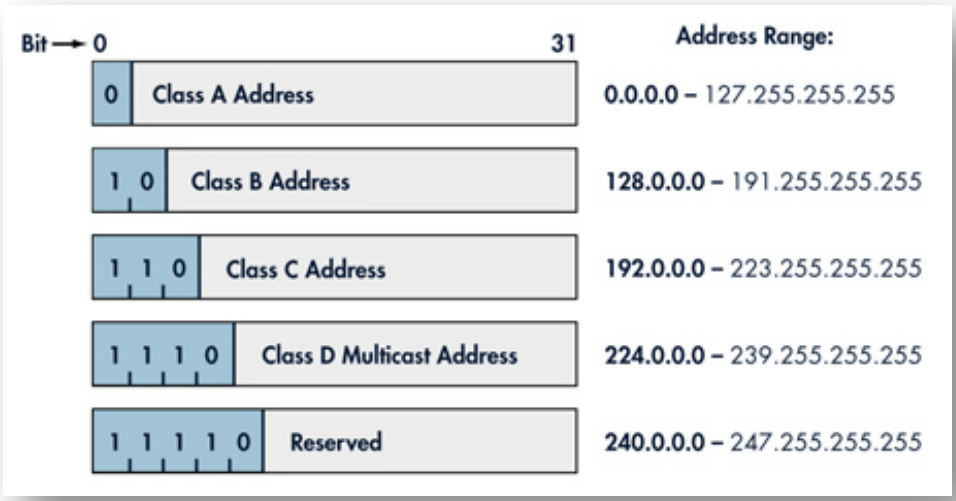
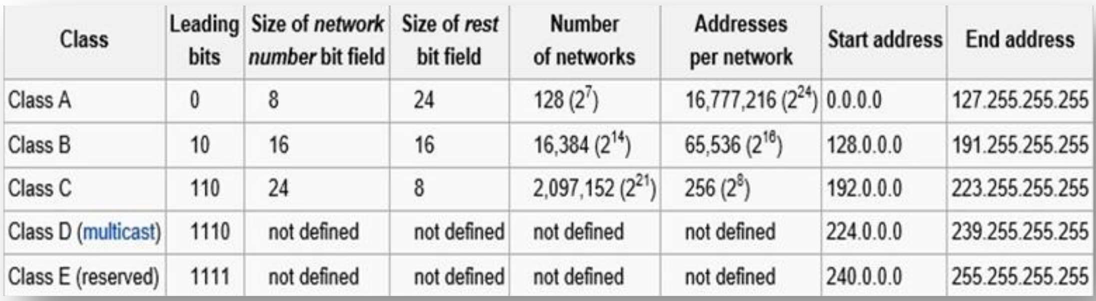


Figure 3. IPv4 Addresses

Here (Figure 3) we see that the five classes are named A, B, C, D, and E. The A, B, and C class spaces were used for unicast addresses. If we look more carefully at this addressing structure, we can see how the relative sizes of the different classes and their corresponding address ranges really work.



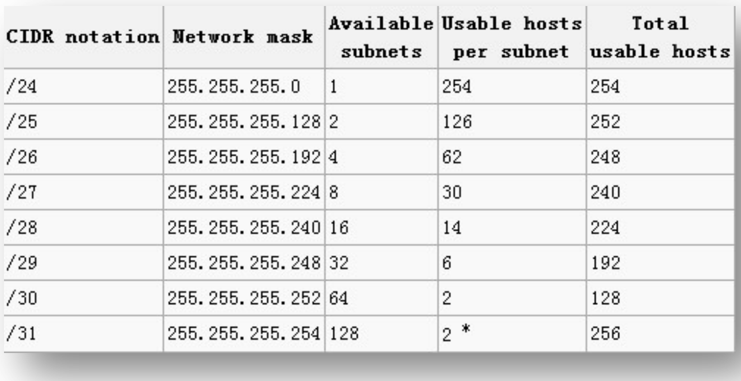
| Class | Leading bits | Size of *network number* bit field | Size of *rest* bit field | Number of networks | Addresses per network | Start address | End address |
|---|---|---|---|---|---|---|---|
| Class A | 0 | 8 | 24 | 128 ($2^7$) | 16,777,216 ($2^{24}$) | 0.0.0.0 | 127.255.255.255 |
| Class B | 10 | 16 | 16 | 16,384 ($2^{14}$) | 65,536 ($2^{16}$) | 128.0.0.0 | 191.255.255.255 |
| Class C | 110 | 24 | 8 | 2,097,152 ($2^{21}$) | 256 ($2^8$) | 192.0.0.0 | 223.255.255.255 |
| Class D (multicast) | 1110 | not defined | not defined | not defined | not defined | 224.0.0.0 | 239.255.255.255 |
| Class E (reserved) | 1111 | not defined | not defined | not defined | not defined | 240.0.0.0 | 255.255.255.255 |

Figure 4. IPv4 space partitioning

The partitioning into classes introduces a trade-off between the number of available networks and the number of hosts that can be assigned to the given network.

### 2.2.2. – Subnet Masks

The subnet mask is an assignment of bits used by a host or router to determine how the network and subnetwork information is partitioned from the host information in a corresponding IP address. Subnet masks for IP are the same length as the corresponding IP addresses (32 bits for IPv4). They are typically configured into a host or router in the same way as IP addresses—either statically (typical for routers) or using some dynamic system such as the Dynamic Host Configuration Protocol (DHCP). For IPv4, subnet masks may be written in the same way an IPv4 address is written (i.e., dotted-decimal). Figure 5 represents some examples of subnet masks in IPv4.



| CIDR notation | Network mask | Available subnets | Usable hosts per subnet | Total usable hosts |
|---|---|---|---|---|
| /24 | 255.255.255.0 | 1 | 254 | 254 |
| /25 | 255.255.255.128 | 2 | 126 | 252 |
| /26 | 255.255.255.192 | 4 | 62 | 248 |
| /27 | 255.255.255.224 | 8 | 30 | 240 |
| /28 | 255.255.255.240 | 16 | 14 | 224 |
| /29 | 255.255.255.248 | 32 | 6 | 192 |
| /30 | 255.255.255.252 | 64 | 2 | 128 |
| /31 | 255.255.255.254 | 128 | 2 * | 256 |

Figure 5. IPv4 subnet mask examples

### 2.2.3. –Broadcast Addresses

Broadcast address is the last address in the network, and it is used for addressing all the nodes in the network at the same time. It means that IP packet, where the destination address is broadcast address, is sent to all nodes of the IP network. It is important for remote announcements in network segment. The subnet broadcast address is formed by setting the network/subnetwork portion of an IPv4 address to the appropriate value and all the bits in the Host field to 1.

The subnet broadcast address is built by inverting the subnet mask and performing a bitwise OR operation with the address of any of the computers on the subnet (or, equivalently, the network/subnetwork prefix). Keep in mind that the result of a bitwise OR operation is 1 if either input bit is 1.

## 2.3. – ARP (Address Resolution Protocol)

For TCP/IP networks, the Address Resolution Protocol (ARP) [RFC0826] provides a dynamic mapping between IPv4 addresses and the hardware addresses used by various network technologies. It is important to note here that the network-layer and link-layer addresses are assigned by different authorities. For network hardware, the primary address is defined by the manufacturer of the device and is stored in permanent memory within the device, so it does not change. On the other hand, the IP address assigned to a network interface is installed by the user or network administrator. The IP addresses assigned to a portable device may be changed when it is moved.

Address resolution is the process of discovering the mapping from one address to another. For the TCP/IP protocol suite using IPv4, this is accomplished by the ARP, which is a generic protocol and it is designed to support mapping between many different types of addresses. In practice it is almost always used to map between 32-bit IPv4 addresses and Ethernet-style 48-bit MAC addresses.

ARP provides a dynamic mapping from a network-layer address to a corresponding hardware address. It happens automatically and adapts to changes over time without requiring reconfiguration by a system administrator. If a host changes its network interface card, it MAC address will change (but retaining its assigned IP address), ARP would continue to operate properly after some delay. ARP operation normally does not need any system administrator changing its configuration.

### 2.3.1. – ARP working

In the example above you see an example of an ARP table on a Computer A. As you can see there is only one entry, this computer has learned that the IP address 192.168.1.2 has been mapped to the MAC address 00:0C:29:63:AF:D0.

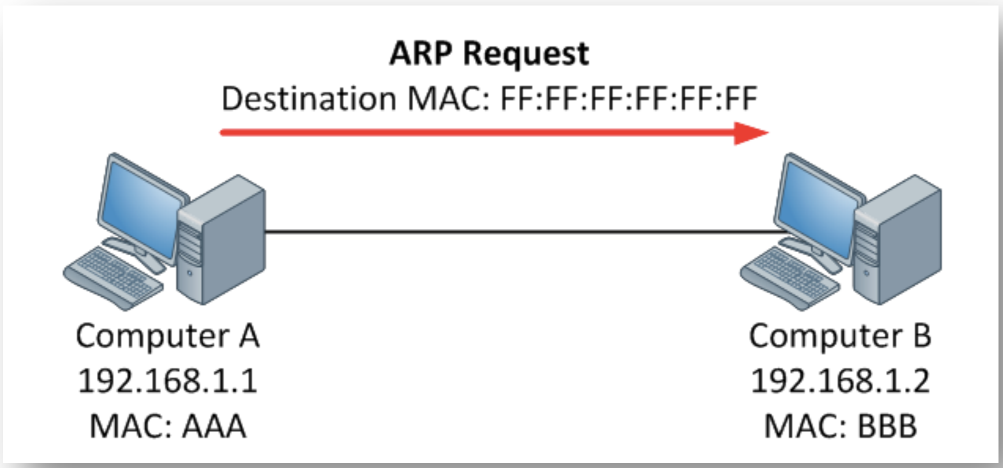


Figure 6. ARP Request

Here (Figure 6) we have two computers and we could see their IP and MAC address. Computer A want to send a ping to computer B but the ARP table is empty so we have no clue what the MAC address of computer B is. First of all computer A will send an ARP Request. This message basically says "Who has 192.168.1.2 and what is your MAC address?" Since we don't know the MAC address we will use the broadcast MAC address for the destination (FF:FF:FF:FF:FF:FF). This message will reach all computers in the Local Area Network.

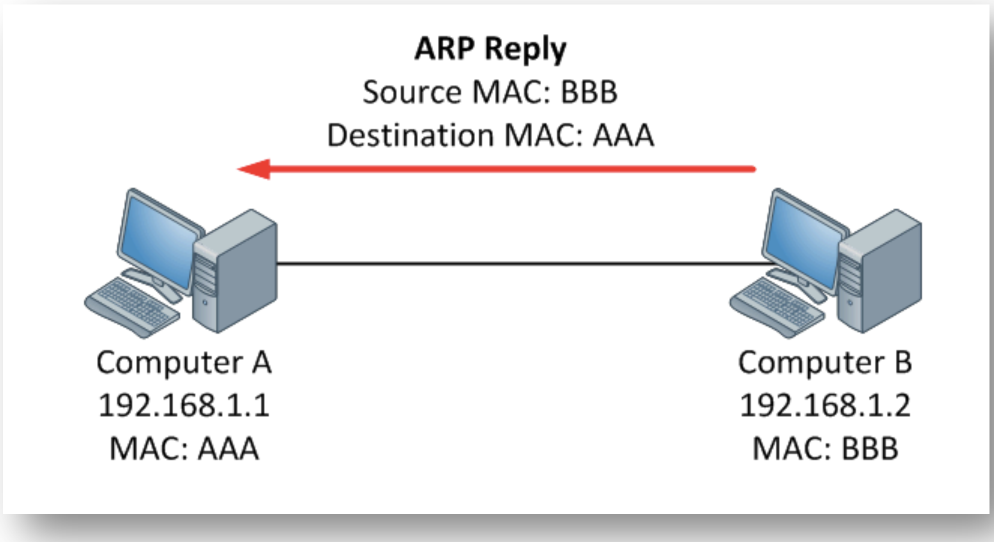


Figure 7. ARP Reply

Computer B will answer with a ARP reply (Figure 7). This reply will say "that's me! And this is my MAC address". Computer A can now add the MAC address to its ARP table and start forwarding data to computer B.

## 2.4. – Internet Protocol (IP)

IP is the most important protocol of the TCP/IP protocol suite. All TCP, UDP, ICMP, and IGMP data are transmitted as IP datagrams. IP provides a best-effort, connectionless datagram delivery service. When we say “best-effort”, we mean there are no guarantees that an IP datagram arrives to its destination successfully. When something goes wrong, such as a router temporarily running out of buffers, IP has a simple error-handling algorithm: throw away some data (usually the last datagram that arrived). Any required reliability must be provided by the upper layers (e.g., TCP).

IP does not maintain any connection state information about related datagrams within the network elements. Each datagram is handled independently from all other others so this also means that IP datagrams can be delivered out of order. These IP datagrams may be duplicated in transit or they may have their data altered as the result of errors. Again, some protocol above IP (usually TCP) has to manage all of these potential problems in order to provide an error-free delivery for applications.

IP supports a number of options that may be selected on a per-datagram basis. Most of these options were introduced in [RFC0791] at the time IPv4 was being designed, when the Internet was considerably smaller and when threats from malicious users were less important. As a consequence, many of the options are no longer practical or desirable because of the limited size of the IPv4 header or concerns regarding security. Most of the standardized options are rarely or never used in the Internet today. Options such as Source and Record Route, for example, require IPv4 addresses to be placed inside the IPv4 header.

### 2.4.1. – IPv4 Header

The normal size of the IPv4 header is 20 bytes, unless options are present (which is rare). The 4 bytes in a 32-bit value are transmitted in the following order: bits 0–7 first, then bits 8–15, then 16–23, and bits 24–31 last. Figure 8 shows the format of an IPv4 datagram.

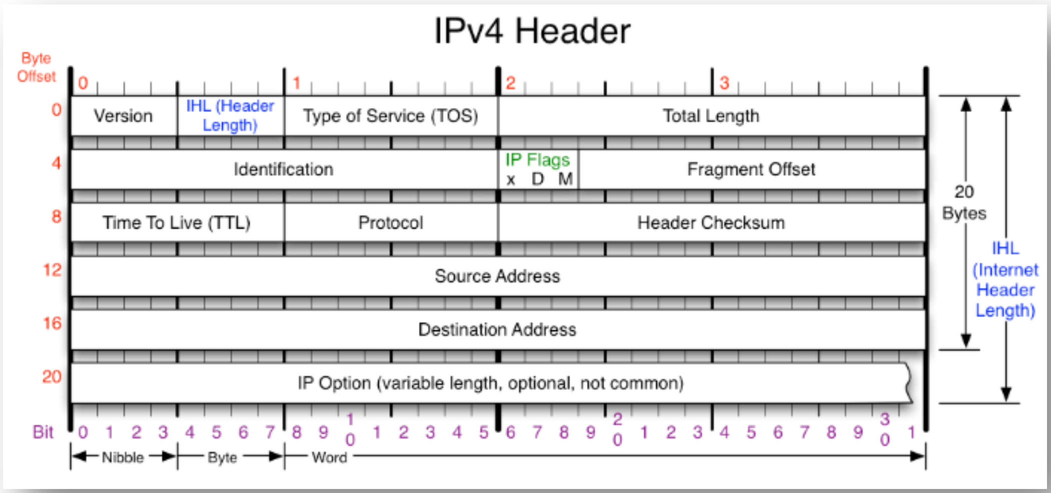


Figure 8. IPv4 Header

The first field (only 4 bits) is the *Version field.* It contains the version number of the IP datagram: 4 for IPv4. The Internet *Header Length (IHL)* field is the number of 32-bit words in the IPv4 header, including any options. Because this is also a 4-bit field, the IPv4 header is limited to a maximum of fifteen 32-bit words or 60 bytes. The normal value of this field (when no options are present) is 5.

Following the header length, the original specification of IPv4 [RFC0791] specified a *Type of Service* (ToS) byte. Use of these never became widespread, so eventually this 8-bit field was split into two smaller parts and redefined by a set of RFCs ([RFC3260], [RFC3168], [RFC2474] and others). The first 6 bits are now called the Differentiated Services Field (DS Field), and the last 2 bits are the Explicit Congestion Notification (ECN) field or indicator bits. These fields are used for special processing of the datagram when it is forwarded.

The *Total Length* field is the total length of the IPv4 datagram in bytes. Using this field and the IHL field, we know where the data portion of the datagram starts, and its length. Because this is a 16-bit field, the maximum size of an IPv4 datagram (including header) is 65,535 bytes. If the Total Length field were not provided, the IPv4 implementation would not know how much of a 46-byte Ethernet frame was really an IP datagram, as opposed to padding, leading to possible confusion.

The *Identification* field helps to identify each datagram sent by an IPv4 host. To ensure that the fragments of one datagram are not confused with those of another, the sending host normally increments an internal counter by 1 each time a datagram is sent and copies the value of the counter into the IPv4 Identification field. This field is most important for implementing fragmentation.

The *Time-to-Live* field, or TTL, sets an upper limit on the number of routers through which a datagram can pass. It is initialized by the sender to some value (64 is recommended [RFC1122], although 128 or 255 is not uncommon) and decremented by 1 by every router that forwards the datagram. When this field reaches 0, the datagram is thrown away, and the sender is notified with an ICMP message. This prevents packets from getting caught in the network forever if an unwanted routing loop occurs.

The *Protocol* field in the IPv4 header contains a number indicating the type of data found in the payload portion of the datagram. The most common values are 17 (for UDP) and 6 (for TCP). This provides a demultiplexing feature so that the IP protocol can be used to carry payloads of more than one protocol type. Although this field originally specified the transport-layer protocol the datagram is encapsulating, it is now understood to identify the encapsulated protocol, which may or not be a transport protocol.

The *Header Checksum* field is calculated over the IPv4 header only. This is important to understand because it means that the payload of the IPv4 datagram (e.g., TCP or UDP data) is not checked for correctness by the IP protocol. To help ensure that the payload portion of an IP datagram has been correctly delivered, other protocols must cover any important data that follows the header with their own data-integrity-checking mechanisms.

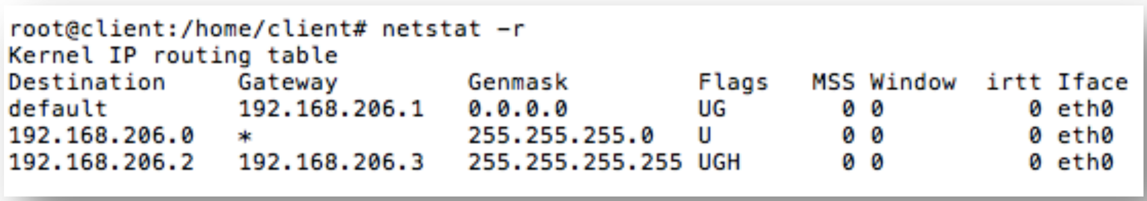
Every IP datagram contains the *Source IP Address* of the sender of the datagram and the *Destination IP Address* of where the datagram is destined. These are 32-bit values for IPv4.

### 2.4.2. – IP Forwarding

IP forwarding is simple, especially for a host. If the destination is directly connected to the host (e.g., a point-to-point link) or on a shared network (e.g., Ethernet), the IP datagram is sent directly to the destination—a router is not required. Otherwise, the host sends the datagram to a single router (called the default router – default input into the IP routing table) and lets the router deliver the datagram to its destination. This simple scheme manages most host configurations.

We begin by noting that most hosts today can be configured to be routers as well as hosts, and many home networks use an Internet-connected PC to act as a router. What differentiates a host from a router to IP is how IP datagrams are handled: a host never forwards datagrams it does not originate, whereas routers do.

IP protocol can receive a datagram either from another protocol on the same machine (TCP, UDP, etc.) or from a network interface. The IP layer has some information in memory, usually called a *IP routing table* or *forwarding table*, which it searches each time it receives a datagram to send. When a datagram is received from a network interface, IP first checks if the destination IP address is one of its own IP addresses (i.e., one of the IP addresses associated with one of its network interfaces) or some other address for which it should receive traffic such as an IP broadcast or multicast address. If so, the datagram is delivered to the protocol module specified by the Protocol field in the IPv4 header. If the datagram is not destined for one of the IP addresses being used locally by the IP module, then: if the IP layer was configured to act as a router, the datagram is forwarded (that is, handled as an outgoing datagram); otherwise the datagram is silently discarded.



```
root@client:/home/client# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         192.168.206.1   0.0.0.0         UG        0 0          0 eth0
192.168.206.0   *               255.255.255.0   U         0 0          0 eth0
192.168.206.2   192.168.206.3   255.255.255.255 UGH       0 0          0 eth0
```

Figure 9. IP routing table

As we can observe in this capture (Figure 9), IP forwarding table contains the following information fields, at least conceptually:

- Destination: This contains a 32-bit field (or 128-bit field for IPv6) used for matching the result of a masking operation (see the next bulleted item). The destination can be as simple as zero, for a “default route” covering all destinations, or as long as the full length of an IP address, in the case of a “host route” that describes only a single destination.

- Gateway (Next-hop): This contains the 32-bit IPv4 address of the next IP entity (router or host) to which the datagram should be sent. The next-hop entity is typically on a network shared with the system performing the forwarding lookup, meaning both share the same network prefix.

- Genmask: This contains a 32-bit field applied as a bitwise AND mask to the destination IP address of a datagram being looked up in the forwarding table. The masked result is compared with the set of destinations in the forwarding table entries.

- Interface: This contains an identifier used by the IP layer to reference the network interface that should be used to send the datagram to its next hop. For example, it could refer to a host's 802.11 wireless interface (wlanX), a wired Ethernet interface (ethX).

## 2.5. – Dynamic Host Configuration Protocol (DHCP)

### 2.5.1. – Introduction

DHCP or Dynamic Host Configuration Protocol is a client/server protocol used to dynamically assign IP-address parameters to a DHCP client (8). DHCP is very widely used, in both enterprises and home networks. Even the most basic home router devices support embedded DHCP servers. DHCP clients are incorporated into all common client operating systems and a large number of embedded devices such as network printers and VoIP phones. These devices usually use DHCP to get their IP address, subnet mask, router IP address, and DNS server IP address.

The design of DHCP is based on an earlier protocol called the Internet Bootstrap Protocol (BOOTP) [RFC0951][RFC1542], which is now effectively obsolete. DHCP extends the BOOTP model with the concept of leases and can provide all information required for a host to operate. Leases allow clients to use the configuration information for an agreed-upon amount of time. A client may request to renew the lease and continue operations, subject to agreement from the DHCP server. BOOTP and DHCP are backward compatible in the sense that BOOTP-only clients can make use of DHCP servers and DHCP clients can make use of BOOTP-only servers. BOOTP, and therefore DHCP as well, are carried using UDP/IP. Clients use port 68 and servers use port 67.

DHCP provides configuration parameters to Internet hosts and it consists of two different parts:

- A protocol for delivering host-specific configuration parameters from a DHCP server to a host
- A mechanism for allocation of network addresses to hosts.

### 2.5.2. – Architecture

DHCP is built on a client-server model, where designated DHCP server hosts allocate network addresses and deliver configuration parameters to dynamically configured hosts. The term "server" refers to a host providing initialization parameters through DHCP, and the term "client" refers to a host requesting initialization parameters from a DHCP server.
DHCP supports three mechanisms for IP address allocation (9):

- Automatic allocation: DHCP assigns a permanent IP address to a client
- Dynamic allocation: DHCP assigns an IP address to a client for a limited period of time
- Manual allocation: The network administrator assigns an IP address to the client. DHCP is used only to transfer the assigned address to the client.

In dynamic allocation, a DHCP client requests the allocation of an IP address. The server answers with one address selected from a pool of available addresses. Typically, the pool is a contiguous range of IP addresses allocated specifically for DHCP’s use. The address given to the client is allocated for only a specific amount of time, called the lease duration.

The client is permitted to use the IP address until the lease expires, although it may request extension of the lease as required. In most situations, clients are able to renew leases they wish to extend.

In a particular network one or more of these different mechanisms can be used. For instance, in our environment, fixed hosts (like web-server, network-server or WANem) have been manually allocated, but DHCP server dynamically assigns IP addresses to the clients.

## 2.5.3. – DHCP Protocol Operation

DHCP messages are essentially BOOTP messages with a special set of options. When a new client attaches to a network, it first discovers what DHCP servers are available and what addresses they are offering. Then, it decides which server will use and which address it desires and requests it from the offering server (while informing all the servers of its choice). Unless the server has allocated this IP address the address in the meantime, it responds by acknowledging the address allocation to the requesting client. In the following image (Figure 10), we will see the time sequence of events between a typical client and server.

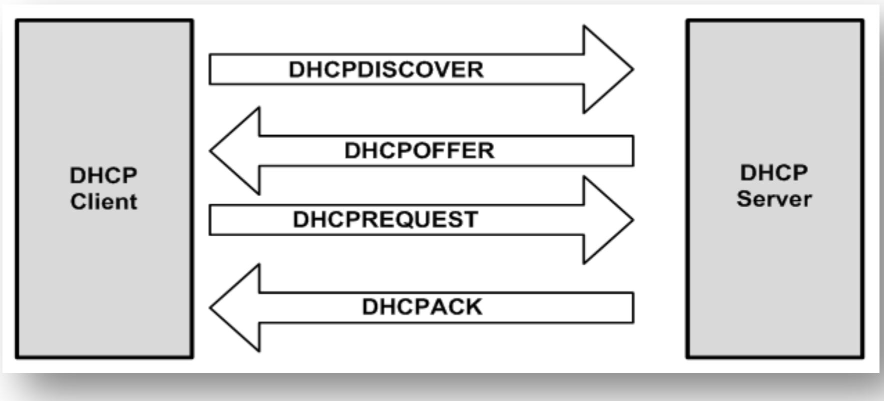


Figure 10. Typical DHCP exchange

In the above image (Figure 10) we could find a typical DHCP exchange, where a client first broadcasts a DHCPDISCOVER message. Then, the server receiving the request, may respond with a DHCPOFFER message, including an offered IP address in the “Your” IP Address field. (Other configuration options, such as IP address of DNS server or subnet mask are often included). The offer message includes the lease time (T), which provides the upper bound on the amount of time the address can be used if it is not renewed. The message also contains the renewal time (T1), which is the amount of time before the client should attempt to renew its lease with the server from which it acquired its lease, and the rebinding time (T2), which bounds the time in which it should attempt to renew its address with any DHCP server. By default, T1 = (T/2) and T2 = (7T/8).

After receiving one or more DHCPOFFER messages from one or more servers, the client determines which offer it will accept and broadcasts a DHCPREQUEST message including the Server Identifier option. The Requested IP Address option is set to the address received in the selected DHCPOFFER message.

After handling the binding, the selected server responds with a DHCPACK message, indicating to the client that the address binding can now be used. Otherwise, in the case where the server cannot allocate the address contained in the DHCPREQUEST message (e.g., it has been allocated in some other way or is not available), the server responds with a DHCPNAK message.

Once the client receives the DHCPACK message and other associated configuration information, it may probe the network to ensure that the address provided is not in use (e.g., by sending an ARP request for the address). If the client determines that the address is already in use, the client stops using the address and sends a DHCPDECLINE message to the server to indicate that the address cannot be used. After a recommended 10s delay, the client will retry. If a client elects to renounce to its address before its lease time expires, it sends a DHCPRELEASE message and this IP address gets freed.

## 2.6. – Domain Name System (DNS) and Name Resolution

### 2.6.1. – Introduction

DNS is a distributed client/server networked database that is used by TCP/IP applications to map between host names and IP addresses (and vice versa). We use the term distributed because no single site on the Internet knows all of the information. Each site (university department, campus, company, or department within a company, for example) maintains its own database of information and runs a server program that other systems across the Internet (clients) can query. The DNS provides the protocol that allows clients and servers to communicate with each other and also a protocol for allowing servers to exchange information.

The set of all names used with DNS constitutes the DNS name space. This space is partitioned hierarchically and is case insensitive. The current DNS name space is a tree of domains with an unnamed root at the top. The top echelons of the tree are called top-level domains (TLDs), which include generic TLDs (gTLDs), country-code TLDs (ccTLDs), and internationalized country-code TLDs (IDN ccTLDs), plus a special infrastructure TLD called, for historical reasons, ARPA [RFC3172].

A domain name consists of a sequence of labels separated by periods. The name represents a location in the name hierarchy, where the period is the hierarchy delimiter and descending down the tree takes place from right to left in the name.

### 2.6.2. – Name Servers and Zones

The unit of administrative delegation, in the language of DNS servers, is called a zone. A zone is a subtree of the DNS name space that can be administered separately from other zones. Every domain name exists within some zone, even the TLDs that exist in the root zone. Whenever a new record is added to a zone, the DNS administrator for the zone allocates a name and additional information (usually an IP address) for the new entry and enters these into the name server’s database.

A DNS server can contain information for more than one zone. At any hierarchical change point in a domain name, a different zone and containing server may be accessed to provide information for the name. This is called a delegation. At a small campus, for example, one person could do this each time a new server is added to the network, but in a large enterprise the responsibility would have to be delegated (probably by departments or other organizational units), as one person could not keep up with the work.

Name servers contain information such as name-to-IP-address mappings that may be obtained from three sources. The name server obtains the information directly from the zone database, as the result of a zone transfer (e.g., for a slave server), or from another server in the course of processing a resolution.

In the first case, the server is said to contain authoritative information about the zone and may be called an authoritative server for the zone. Such servers are identified by name within the zone information. Most name servers (except some of the root and TLD servers) also cache zone information they learn, up to a time limit called the time to live (TTL). They use this cached information to answer queries. Doing so can greatly decrease the amount of DNS message traffic that would otherwise be carried on the Internet.

When answering a query, a server indicates whether the information it is returning has been derived from its cache or from its authoritative copy of the zone. When cached information is returned, it is common for a server to also include the domain names of the name servers that can be contacted to retrieve authoritative information about the corresponding zone.

### 2.6.3. – Domain Name System Protocol

DNS protocol consists of two main parts: a query/response protocol used for performing queries against the DNS for particular names, and another protocol for name servers to exchange database records (zone transfers). It also has a way to notify secondary servers that the zone database has evolved and a zone transfer is necessary (DNS Notify), and a way to dynamically update the zone (dynamic updates). The most typical usage is a simple query/response to look up the IPv4 address that corresponds to a domain name.

Most often, DNS name resolution is the process of mapping a domain name to an IPv4 address. DNS query/response operations are supported over the distributed DNS infrastructure consisting of servers deployed locally at each site or ISP, and a special set of root servers. There is also a special set of generic top-level domain servers used for scaling some of the larger gTLDs, including COM and NET.

As of mid-2011, there are 13 root servers named by the letters A through M. There are also 13 gTLD servers, also labeled A through M. By contacting a root server and possibly a gTLD server, the name server for any TLD in the Internet can be discovered. These servers are mutually coordinated to provide the same information. Some of them are not a single physical server but instead a group of servers (over 50 for the J root server) that use the same IP address.

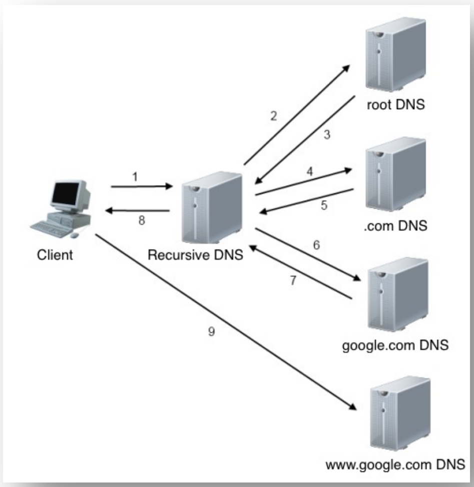


Figure 11. DNS Recursive Query

### 2.6.4. – DNS Message Format

There is one basic DNS message format [RFC6195]. It is used for all DNS operations (queries, responses, zone transfers, notifications, and dynamic updates), as we could see in Figure 12. The basic DNS message begins with a fixed 12-byte header followed by four variable-length sections: questions (or queries), answers, authority records, and additional records.

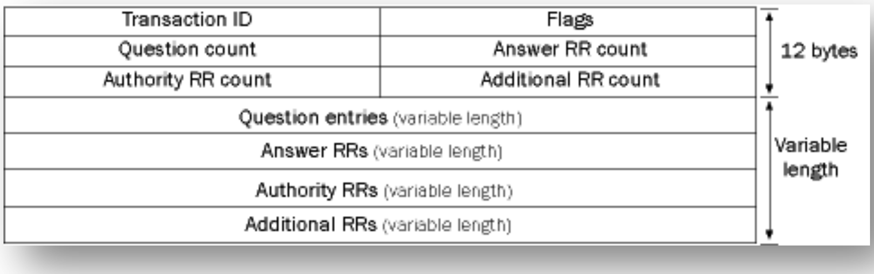
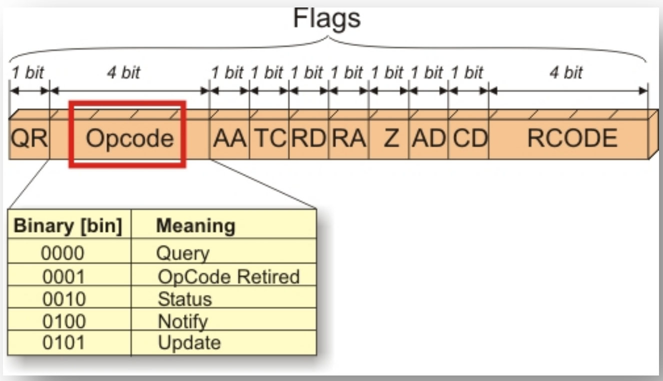

Transaction ID
Flags
Question count
Answer RR count
Authority RR count
Additional RR count
12 bytes
Question entries (variable length)
Answer RRs (variable length)
Authority RRs (variable length)
Additional RRs (variable length)
Variable length

Figure 12. DNS Message Format

In the fixed-length header, the Transaction ID field is set by the client and returned by the server. It lets the client match responses to requests. The second 16-bit word includes a number of flags and other subfields (Figure 13).



| Binary [bin] | Meaning |
|---|---|
| 0000 | Query |
| 0001 | OpCode Retired |
| 0010 | Status |
| 0100 | Notify |
| 0101 | Update |

Figure 13. DNS Flags

Beginning from the left bit, *QR* is a 1-bit field: 0 means the message is a query; 1 means it is a response. The next is the *OpCode*, a 4-bit field. The normal value is 0 (a standard query) for requests and responses. Other values are: 4 (notify), and 5 (update). Next is the *AA* bit field that indicates an “authoritative answer” (as opposed to a cached answer). *TC* is a 1-bit field that means “truncated”. With UDP, this flag being set means that the total size of the reply exceeded 512 bytes, and only the first 512 bytes of the reply were returned. *RD* is a bit field that means “recursion desired”. It tells the server to perform a recursive query. If the bit is not set, and the requested name server does not have an authoritative answer, the requested name server returns a list of other name servers to contact for the answer. *RA* is a bit field that means “recursion available”. This bit is set in the response if the server supports recursion. Root servers generally do not support recursion, and the force clients to perform iterative queries to complete name resolution.

The *Z* bit field is reserved for future use. The *AD* bit field is set to true if the contained information is authenticated, and the CD bit is set to true if security checking is disabled. The *Response Code* (or *RCODE*) field is a 4-bit field with the return code of possible values. The common values include 0 (no error) and 3 (name error or “nonexistent domain,” written as NXDOMAIN). A list of the first 11 error codes could also be checked in DNS documentation. A name error is returned only from an authoritative name server and means that the domain name specified in the query does not exist.

### 2.6.5. – DNS with TCP or UDP

The well-known port number for DNS is 53, for both UDP and TCP. The most common format uses the UDP/IPv4 datagram structure. When a resolver issues a query and the response comes back with the TC bit field set (“truncated”), the size of the true response exceeded 512 bytes, so only the first 512 bytes are returned by the server. The resolver may issue the request again, using TCP, which now must be a supported configuration [RFC5966]. This allows more than 512 bytes to be returned because TCP breaks up large messages into multiple segments.

If UDP is used, both the resolver and the server application software must implement their own timeout and retransmission. RFC1536 suggests starting with a timeout of at least 4s, and that subsequent timeouts result in an exponential increase of the timeout.

## 2.7. – Transmission Control Protocol (TCP)

### 2.7.1. – Introduction

TCP is the Internet's transport-layer, connection-oriented and reliable transport protocol. In this section, we'll see that in order to provide reliable data transfer, TCP relies on many of the underlying principles discussed previously, including error detection, retransmissions, cumulative acknowledgments, timers, header fields for sequence and acknowledgment numbers. TCP is defined in RFC 793, RFC 1122, RFC 1323, RFC 2018, and RFC 2581.

### 2.7.2. – TCP Segment Structure

The TCP segment consists of header fields and a data field. The data field contains a part of application data. The MSS limits the maximum size of a segment's data field, so when TCP sends a large file, such as an image as part of a Web page, it typically breaks the file into chunks of size MSS (except for the last chunk, which will often be less than the MSS).

Interactive applications, however, often transmit data chunks that are smaller than the MSS; for example, with remote login applications like Telnet, the data field in the TCP segment is often only one byte. Because the TCP header is typically 20 bytes (12 bytes more than the UDP header), segments sent by Telnet may be only 21 bytes in length.

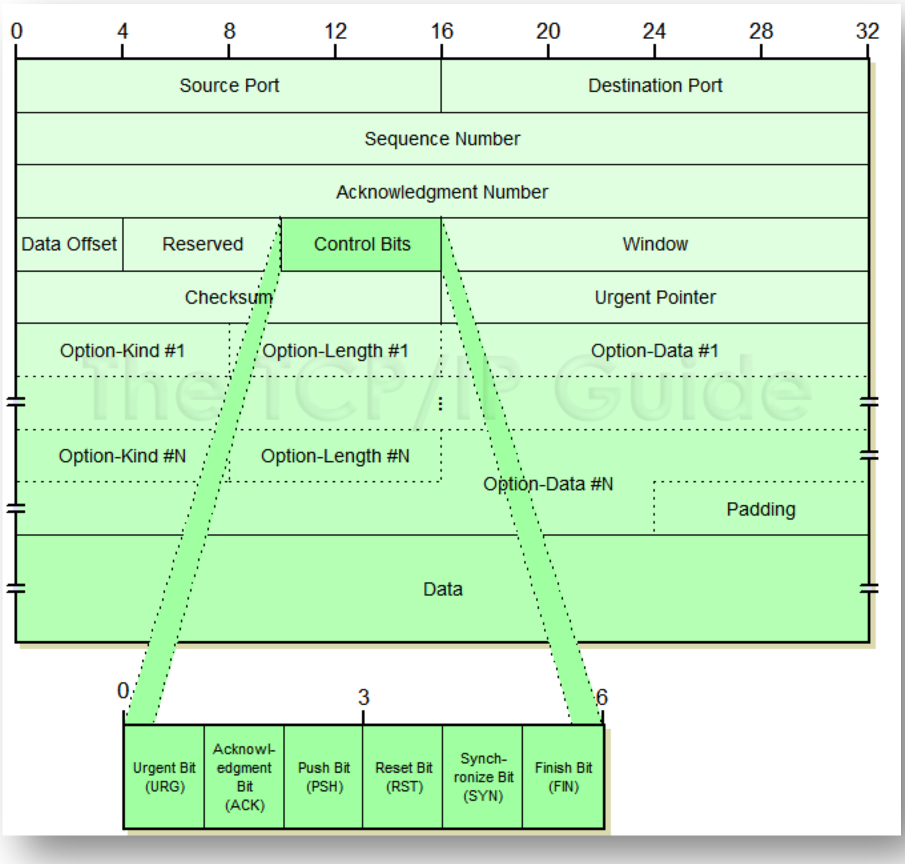


Figure 14. TCP Segment Format

Figure 14 shows the structure of the TCP segment. The header includes source and destination port numbers, which are used for multiplexing/demultiplexing data from/to upper-layer applications. Also, the header includes a checksum field. A TCP segment header also contains the following fields:

- The 32-bit *sequence number* field and the 32-bit *acknowledgment number* field are used by the TCP sender and receiver in implementing a reliable data transfer service.

- The 16-bit *receive window* field is used for flow control. We will see shortly that it is used to indicate the number of bytes that a receiver is willing to accept.

- The 4-bit *header length* field (Data Offset) specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field. (Typically, the options field is empty, so that the length of the typical TCP header is 20 bytes.)

- The optional and variable-length *options* field is used when a sender and receiver negotiate the maximum segment size (MSS) or as a window scaling factor for use in high-speed networks. A time-stamping option is also defined. In RFC 854 and RFC 1323 we could find additional details.

- The *flag* field (Control Bits) contains 6 bits. The *ACK* bit is used to indicate that the value carried in the acknowledgment field is valid; that is, the segment contains an acknowledgment for a segment that has been successfully received. The *RST*, *SYN*, and *FIN* bits are used for connection setup and teardown. Setting the *PSH* bit indicates that the receiver should pass the data to the upper layer immediately. Finally, the *URG* bit is used to indicate that there is data in this segment that the sending-side upper-layer entity has marked as "urgent." The location of the last byte of this urgent data is indicated by the 16-bit urgent data pointer field. TCP must inform the receiving-side upper-layer entity when urgent data exists and pass it a pointer to the end of the urgent data. (In practice, the *PSH*, *URG*, and the urgent data pointer are not used.)

### 2.7.3. – TCP Connection

TCP is connection-oriented protocol because before one application process can begin to send data to another, the two processes must first "handshake" with each other—that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer. As part of TCP connection establishment, both sides of the connection will initialize many TCP state variables associated with the TCP connection.

The TCP "connection" is not an end-to-end TDM or FDM circuit as in a circuit-switched network. It is not a virtual circuit, as the connection state resides entirely in the two end systems. Because the TCP protocol runs only in the end systems and not in the intermediate network elements (routers and link-layer switches), the intermediate network elements do not maintain TCP connection state. In fact, the intermediate routers are completely ignorant to TCP connections; they see datagrams, not connections.

A TCP connection provides a full-duplex service: If there is a TCP connection between Process A on one host and Process B on another host, then application-layer data can flow from Process A to Process B at the same time as application-layer data flows from Process B to Process A. A TCP connection is also always point-to-point, that is, between a single sender and a single receiver.

## TCP Connection: Establishment and Termination

A TCP connection is defined to be a 4-tuple consisting of two IP addresses and two port numbers. More precisely, it is a pair of endpoints or sockets where each endpoint is identified by a (IP address, port number) pair. A TCP connection typically goes through three phases: setup, data transfer (called established), and teardown (closing). As we will see, some of the difficulty in creating a robust TCP implementation is handling all of the transitions between and among these phases correctly. A typical TCP connection establishment and close (without any data transfer) is shown in Figure 15.

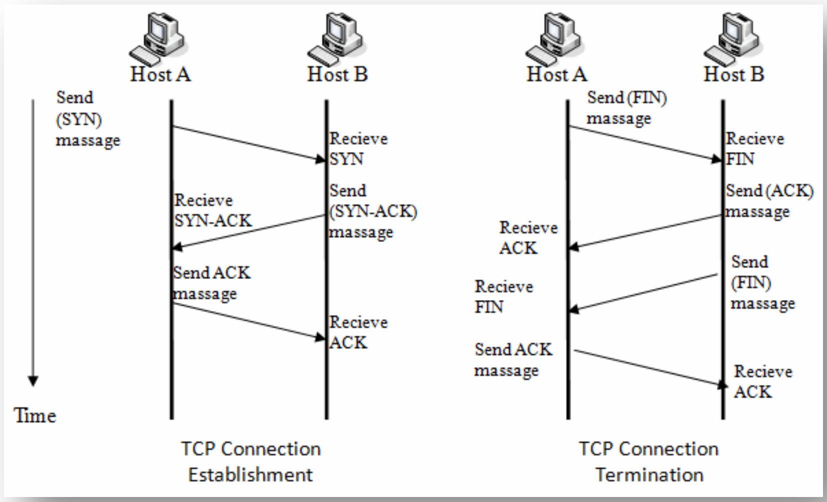


Figure 15. TCP connection (establishment and close)

In this image (Figure 15) we could see a normal TCP connection establishment and termination. The client starts a three-way handshake to exchange initial sequence numbers carried on SYN segments for the client and server. The connection terminates after each side has sent a FIN and received an acknowledgment for it, so the final exchange is a four-way handshake.

Now we will see the initial **three-way** handshake in more detail:

- First, the *active opener* (normally called the client – Host A) sends a SYN segment (a TCP/IP packet with the SYN bit field turned on in the TCP header) specifying the port number of the peer to which it wants to connect and the client's initial sequence. This is segment 1.

- After that, the server (Host B) responds with its own SYN segment containing its initial sequence number. This is segment 2. The server also acknowledges the client's SYN by ACKing plus 1. A SYN consumes one sequence number and is retransmitted if lost.

- Finally the connection is established when the client (Host A) acknowledge this SYN from the server by ACKing plus 1. This is segment 3.

When the connection is completely established, data transfer (request and reply in Figure 14) begins. After this data exchange ends, the connection is closed with a **four-way** handshake:

- The *active closer* (Host A) sends a FIN segment specifying the current sequence number the receiver expects to see. The FIN also includes an ACK for the last data sent in the other direction.
- The *passive closer* (Host B) responds by ACKing value M + 1 to indicate its successful receipt of the active closer's FIN. At this point, the application is notified that the other end of its connection has performed a close. Typically this results in the application initiating its own close operation.

- The *passive closer* (Host B) then effectively becomes another active closer and sends its own FIN. The sequence number is equal to N.

- To complete the close, the final segment contains an ACK for the last FIN. Note that if a FIN is lost, it is retransmitted until an ACK for it is received.

## 2.8. – HyperText Transfer Protocol (HTTP)

### 2.8.1 – Introduction

The HyperText Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web. It is defined in [RFC 1945] and [RFC 2616]. HTTP is implemented in two programs: a client program and a server program. The client program and server program, executing on different end systems, talk to each other by exchanging HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages.

A Web page consists of objects. An object is simply a file—such as an HTML file, a JPEG image, a Java applet, or a video clip—that is addressable by a single URL. Most Web pages consist of a base HTML file and several referenced objects. The base HTML file references the other objects in the page with the objects' URLs. Each URL has two components: the hostname of the server that houses the object and the object's path name. If a Web page contains HTML text and five JPEG images, then the Web page has six objects: the base HTML file plus the five images.

Web browsers (such as Internet Explorer and Firefox) implement the client side of HTTP. In the context of the Web, browser and client are used indistinctly. Web servers, which implement the server side of HTTP, are addressable by a URL. Some popular Web servers include *Apache*, which we will use later in the *Laboratory Installation* section, and *Microsoft Internet Information Server*.

HTTP uses TCP as its underlying transport protocol (rather than running on top of UDP). The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces. On the client side the socket interface is the door between the client process and the TCP connection; on the server side it is the door between the server process and the TCP connection.

TCP provides a reliable data transfer service to HTTP. This implies that each HTTP request message sent by a client process eventually arrives intact at the server; similarly, each HTTP response message sent by the server process eventually arrives intact at the client. Here we see one of the great advantages of a layered architecture—HTTP need not worry about lost data or the details of how TCP recovers from loss or reordering of data within the network. That is the job of TCP and the protocols in the lower layers of the protocol stack.

Because an HTTP server maintains no information about the clients, HTTP is said to be a *stateless protocol*. If a particular client asks for the same object twice in a period of a few seconds, the server resends the object, as it has completely forgotten what it did earlier.

### 2.8.2. – HTTP Connections: Persistent and Non-Persistent

Depending on the application and on how the application is being used, the series of requests may be made back-to-back, periodically at regular intervals, or intermittently.

When this client-server interaction is taking place over TCP, the application developer needs to make an important decision – should each request/response pair be sent over a separate TCP connection, or should all of the requests and their corresponding responses be sent over the same TCP connection?

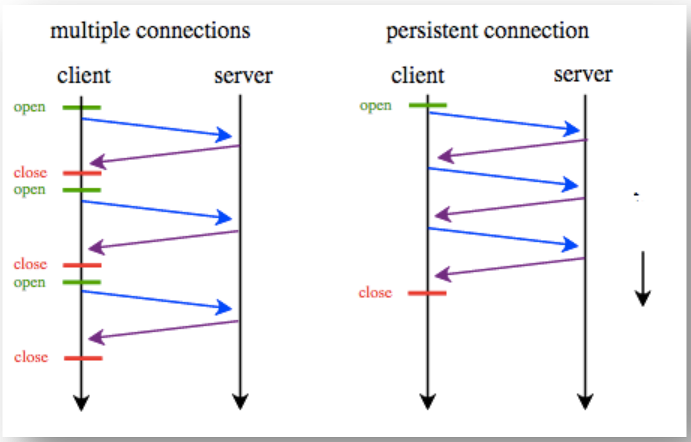


Figure 16. HTTP Non-Persistent and Persistent Connections

As we can observe in Figure 16 – Left, with the **non-persistent** (multiple) connections we could find the following behaviour:

- The HTTP client process initiates a TCP connection to the web server on port number 80, which is the default port number for HTTP.
- The HTTP client sends an HTTP request message to the server. (We will discuss HTTP messages in some in the *Wireshark Labs* section)
- The HTTP server process receives the request message, retrieves the object /from its storage, encapsulates the object in an HTTP response message, and sends the response message to the client.
- The HTTP server process tells TCP to close the TCP connection. (But TCP doesn't actually terminate the connection until it knows for sure that the client has received the response message properly).
- The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, examines the HTML file, and finds references to the other HTML objects.
- The first four steps are then repeated for each of the rest of the objects.

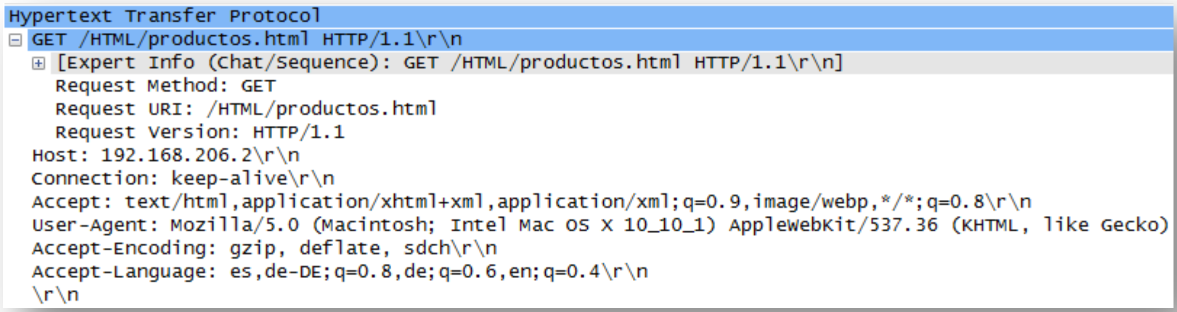
Otherwise, with **persistent** connections (Figure 16 – Right), the server leaves the TCP connection open after sending a response, so requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page (for example, the base HTML file and the rest of HTML files) can be sent over a single persistent TCP connection. Moreover, multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.

Typically, the HTTP server closes a connection when it isn't used for a certain time (a configurable timeout interval). When the server receives the back-to-back requests, it sends the objects back-to-back. The default mode of HTTP uses persistent connections with pipelining.

### 2.8.3. – HTTP Message Format

#### Request Message

The first line of an HTTP request message is called the request line, the next lines are called the header lines. The request line has three fields: the method field, the URL field, and the HTTP version field. The method field can take on several different values, including GET, POST, HEAD, PUT, and DELETE. The most of HTTP request messages use the GET method. The GET method is used when the browser requests an object, with the requested object identified in the URL field.



```
Hypertext Transfer Protocol
⊟ GET /HTML/productos.html HTTP/1.1\r\n
  ⊞ [Expert Info (Chat/Sequence): GET /HTML/productos.html HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /HTML/productos.html
    Request Version: HTTP/1.1
  Host: 192.168.206.2\r\n
  Connection: keep-alive\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko)
  Accept-Encoding: gzip, deflate, sdch\r\n
  Accept-Language: es,de-DE;q=0.8,de;q=0.6,en;q=0.4\r\n
  \r\n
```

Figure 17. HTTP – GET Request

First of all, in Figure 17 we could see that the message is written in ordinary ASCII text. Second, we see that the message consists of seven lines, each followed by a carriage return and a line feed. The last line is followed by an additional carriage return and line feed. Although this particular request message has seven lines (the main lines), a request message can have many more lines or as few as one line.

Now let's look at the header lines in the example. The header line Host: 192.168.206.2 specifies the host on which the object resides. It's a local host, so here we could find directly its IP address instead of its Domain Name. By including the `Connection: keep-alive` header line, the browser is telling the server that it wants to continue with persistent connections; it does not want the server to close the connection after sending the requested object.

The `User-agent:` header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser. This header line is useful because the server can actually send different versions of the same object to different types of user agents.

Finally, the `Accept-language:` header indicates that the user prefers to receive a Spanish, English or German version of the object, if such an object exists on the server; otherwise, the server should send its default version. The `Accept-language:` header is just one of many content negotiation headers available in HTTP.

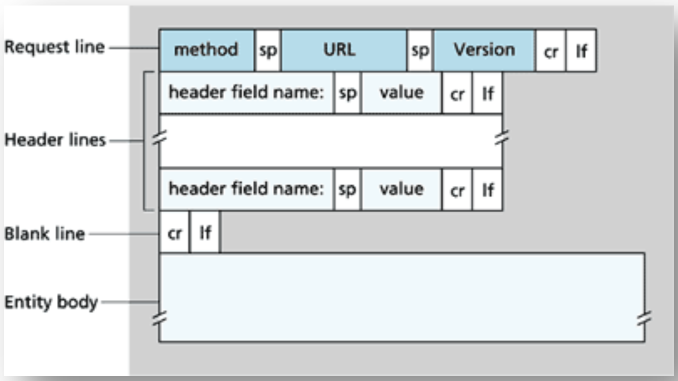


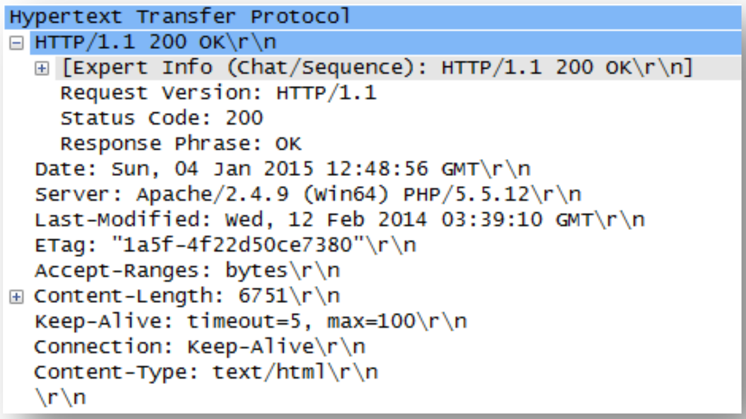
Figure 18. HTTP – General Format of a Request

We see that the general format closely follows our earlier example. However, after the header lines (and the additional carriage return and line feed) there is an "entity body". The entity body is empty with the GET method, but is used with the POST method. An HTTP client often uses the POST method when the user fills out a form—for example, when a user provides search words to a search engine. With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.

The HEAD method is similar to the GET method. When a server receives a request with the HEAD method, it responds with an HTTP message but it leaves out the requested object. Application developers often use the HEAD method for debugging. The PUT method is often used in conjunction with Web publishing tools. It allows a user to upload an object to a specific path (directory) on a specific Web server. The PUT method is also used by applications that need to upload objects to Web servers. The DELETE method allows a user, or an application, to delete an object on a Web server.

## Response Message

Now we will see a typical example of a HTTP response message. It has three sections: an initial status line, header lines, and then the entity body. The entity body contains the requested object itself. The status line has three fields: the protocol version field, a status code, and a corresponding status message. In this example, the status line indicates that the server is using HTTP/1.1 and that everything is OK (Success Code: 200)



```
Hypertext Transfer Protocol
⊟ HTTP/1.1 200 OK\r\n
  ⊞ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Request Version: HTTP/1.1
    Status Code: 200
    Response Phrase: OK
  Date: Sun, 04 Jan 2015 12:48:56 GMT\r\n
  Server: Apache/2.4.9 (Win64) PHP/5.5.12\r\n
  Last-Modified: Wed, 12 Feb 2014 03:39:10 GMT\r\n
  ETag: "1a5f-4f22d50ce7380"\r\n
  Accept-Ranges: bytes\r\n
⊞ Content-Length: 6751\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html\r\n
  \r\n
```

Figure 19. HTTP – Response Message

The server uses the `Connection: Keep-Alive` header line to tell the client that it is not going to close the TCP connection after sending the message. The `Date:` header line indicates the time and date when the HTTP response was created and sent by the server. Keep in mind that this is not the time when the object was created or last modified; it is the time when the server retrieves the object from its file system, inserts the object into the response message, and sends the response message. The Server: header line indicates that the message was generated by an Apache Web server; it is analogous to the `User-agent:` header line in the HTTP request message.

The `Last-Modified:` header line indicates the time and date when the object was created or last modified. The `Last-Modified:` header is critical for object caching, both in the local client and in network cache servers (also known as proxy servers). The `Content-Length:` header line indicates the number of bytes in the object being sent. The `Content-Type:` header line indicates that the object in the entity body is HTML text. (The object type is officially indicated by the `Content-Type:` header and not by the file extension.)

Now we have some common status codes and associated phrases include [18]:

- 200 OK: Request succeeded and the information is returned in the response.
- 301 Moved Permanently: Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.
- 400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.
- 404 Not Found: The requested document does not exist on this server.
- 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

# 3. – Laboratory Installation

*This section focuses on explaining how we have the laboratory installed. As a first step, the available hardware will be detailed and as a next step we will explain the set up process. Next, we will talk about the software tools we have chosen, how to install and how they have been configured.*

*The main protocols, such as DHCP, DNS or IP, and their basic configuration will be also explained. These protocols need to be set up correctly in order to get our LAN working properly.*

*Then we will detail the Wide Area Network emulator software used, the available versions and its basic configuration. As WANem will be our basic tool in the 'Wireshark Labs' section, it is important to have this server well configured.*

## 3.1. – Hardware

When installing a laboratory, we first need some knowledge about the available equipment. We need to decide which machine is the most appropriate for each service, verifying that the hardware meets the requirements. Once clear about the laboratory's structure and the hardware distribution, we are ready to begin with the hardware installation. Once successfully installed, we begin with the software part (i.e. Operative System, protocols and servers).

### 3.1.1. – Available Equipment

First of all we will describe which is the hardware available in the laboratory. We can split hardware in two different parts:

- Hosts: clients and servers (computer case, screens, keyboards and mice)
- Other devices: routers, switches, external Ethernet cards, RJ45 Ethernet LAN cable (cat 5 and cat 6)

Then we have to assign a function to each machine and distribute them properly. Once we have all the hardware in its correct place, we need to connect the Local Area Network cables to the servers, hosts and other devices. As there will be a lot of RJ45 cables, we will need some colour code to identify said connections. The chosen schema has been the as follows:

- Yellow cable: Client ➔ Switch
- Red cable: Internal servers ➔ Switch
- Green cable: Server ➔ Outgoing to Internet

The exact hardware to be used by each device, its brand name and what is the model number should be as follows.

- Plug & play switch used has to be:
  - Netgear JGS524
- Manual switch, allowing us to choose between LAN or Internet, is:
  - Delock Ethernet Switch RJ45 10/100 Mb/s 4-Port manual
- The computers used for the clients are:
  - HP Compaq dx6100 MT
- The main server, which is running inside the two virtual machines, is:
  - Dell Vostro
- The Wide Area Network emulator software is running on:
  - HP Compaq dx6100 MT

### 3.1.2. – The Environment

Once we know what the available equipment is, we have to think about the network's distribution. There are several possibilities however we need to select the one which closely fits our network. We are going to study two options, first a simple one and then another one more complicated but also more useful.

#### First Layout

Here we have a complete network's diagram with the first layout. We can see every host with their own IP and MAC addresses (Those addresses are not changing when implementing the second layout).

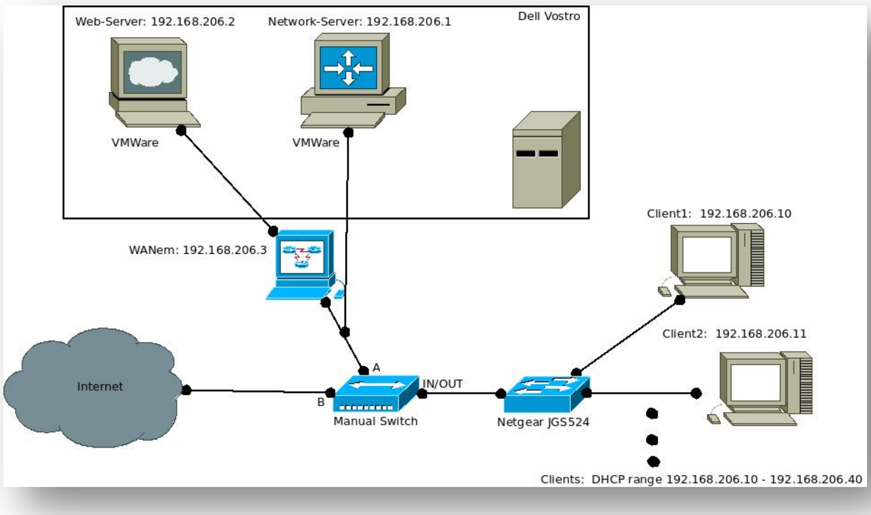


Figure 20. Environment – First layout (with manual Switch)

If we observe the diagram (Figure 20), we see that Dell Vostro is the main “real” host and it runs inside two virtual machines: the Network-Server and the Web-Server. Both virtualized machines are connected to the LAN, but the Web-Server is connected through the WANem host and the Network-Server is connected directly to this LAN.

The other devices shown are a manual switch and a non-configurable switch. The second one, Netgear JGS524, is connecting the clients to the rest of the network (being LAN or Internet). The first one allows us to choose (manually) if we want to work with the emulated WAN or with Internet.

## Second Layout

There is another option to implement this environment. In this other way, the manual switch is removed and we add a second NIC (Network Interface Card) to Network-Server machine. This host will be configured to receive outgoing traffic from the Local Area Network and send it out of the laboratory through the new NIC. We have to edit the IP routing table in Network-Server, adding a new default route. In this case we would have this other network layout:

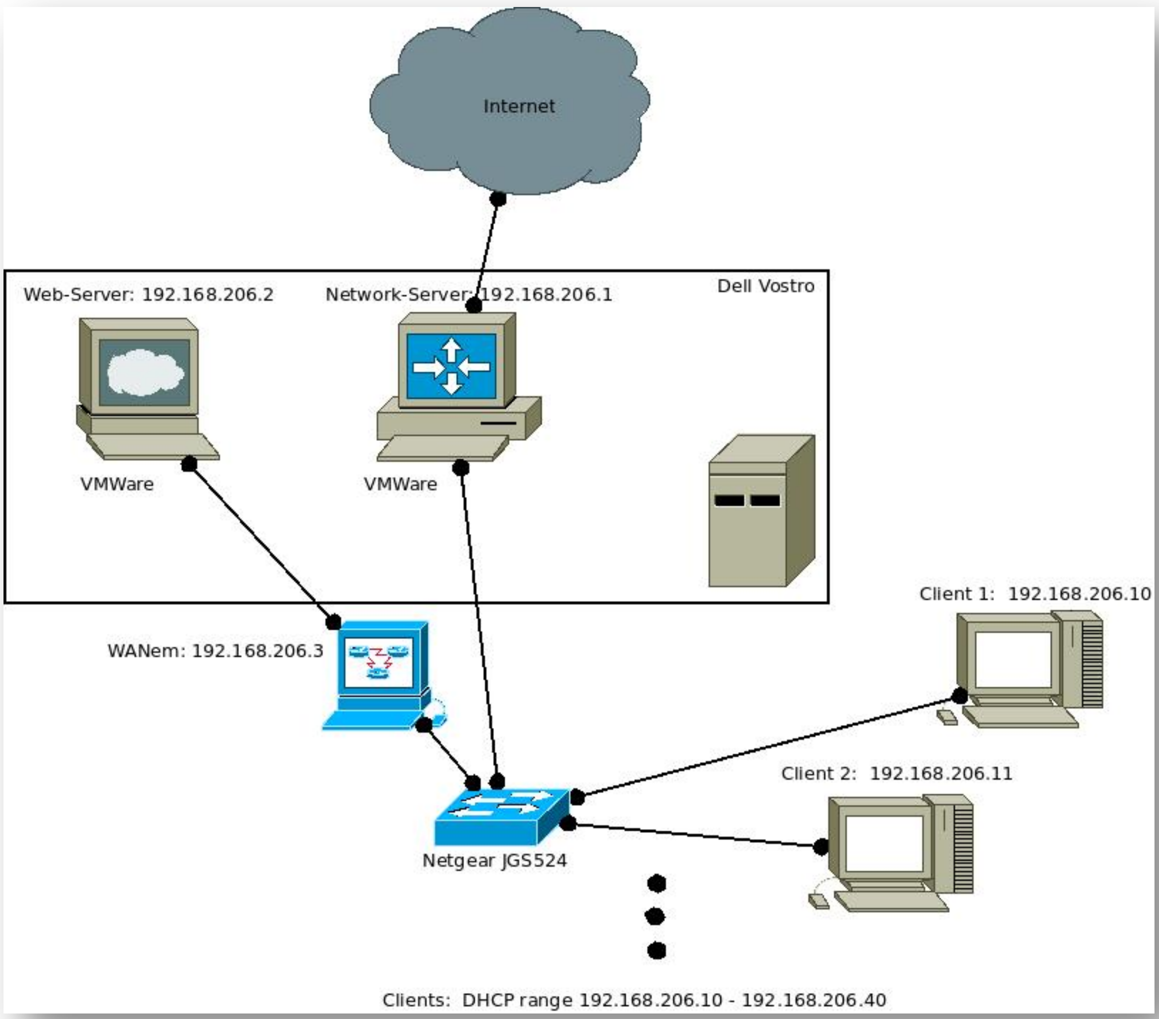


Figure 21. Environment – Second layout

This second environment (Figure 21) could at first sight seem simple to implement, however as our Network-Server is a virtual machine it will be rather more complicated. It requires some configuration by the administrator, but it is worth it for the following reasons.

If we choose to implement our network following this second option, when a client in the LAN types on a web browser “www.j103.lan” it will connect with the local Web-Server through the WANem host. On the other hand, if a client types, for example, “www.google.com” on its web browser, the Network-Server will realize this connection goes outside the Local Area Network and it will send it to the next hop.

When possible, it is always advisable to implement this second network layout. The advantages are clear: we don’t have to choose between one network or another, we can work with both at the same time. Maybe one client wants to work with the Local Area Network to do some tests with the WANem host, but another client wants to work with Internet. This would only be possible with this second layout. If we have to choose manually between one network or another, all the clients must work with the same network.

### 3.1.3. – Setting up Network-Server as a Router

#### Network-Server – Two Network Interface Cards

To implement this second network layout, we need to configure Network-Server as a router. It will distinguish between internal and outgoing traffic and it will send this second one outside the laboratory, through its second NIC (Network Interface Card).

Both Network-Server and Web-Server have been installed as virtual machines into Dell Vostro host. This provides many advantages, such as opportunity to add different virtual network interfaces and assign virtual IP and MAC addresses to each one in an easy way. So we can have only one real NIC (Network Interface Card) and share it between Network-Server and Web-Server, and of course each one could have its own independent addresses.

On the other hand, having these hosts virtualized adds some difficulties to our environment. One of the most important problems appears when we try to add external devices into a virtualized machine, such as USB Network Interface Cards.

We are using VMware as virtualization software, and this software does not allow the virtual machine to manage USB ports until the “guest” system has completely booted up. So when the guest machine’s kernel is booting up, it cannot “see” any USB device (such as the USB Network Interface Card).

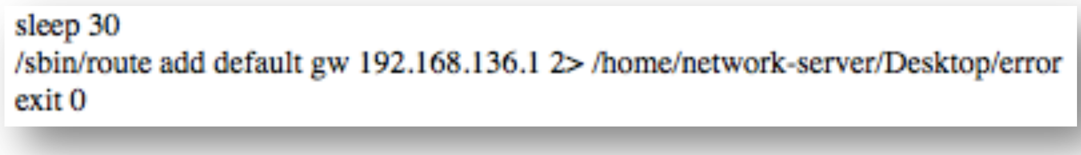
In Network-Server we need one interface for the LAN’s traffic and another one for the outgoing traffic. If we observe the file “FICHERO INTERFACES” we can see how, as the kernel didn’t see this USB NIC while system was booting up, the Network-Manager has written the IP routing table irrespective of the USB Network Interface Card. As a result, Network-Server doesn’t have outgoing connection from the laboratory.

The solution could seem simple: if we write a new IP routing table including the USB NIC, once the machine has detected it, everything should be fixed. By using “route” command we could add the proper line into the IP routing table to send outgoing traffic through the USB Network Interface Card.

First we need to remove the existing default route (“route del default”) and then, simple typing “`route add default gw 192.168.136.1`” we will add the proper line to the IP routing table. This line will set up USB NIC as default route (for outgoing traffic). [More options about this command could be checked simply typing “man route”].

Unfortunately this could only be a short-term solution, it is not definitive. It’s not really useful if we have to type manually the IP routing table each time we boot Network-Server. Keep in mind that if the Network-Server machine doesn’t have any outgoing connection from the laboratory, nobody else will have it because Network-Server is working as laboratory’s router.

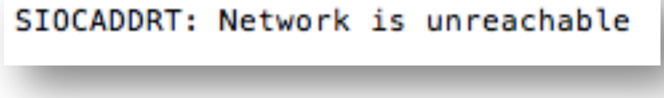
Another solution, and more useful, could be to use “rc.local” file. This file is located in “/etc/rc.local” and, it allows us to execute command lines written inside, each time the system is booted. We can edit “rc.local” file and add a new default route to send outgoing traffic through the USB NIC. This line could be: “sbin/route add default gw 192.168.136.1” (Figure 22). Keep in mind that we need to activate SUID on “route” command if we want to execute it from the “rc.local”.



```
sleep 30
/sbin/route add default gw 192.168.136.1 2> /home/network-server/Desktop/error
exit 0
```

Figure 22. File “*rc.local*”

Now another problem related to USB NIC pops up: “*route*” command can only write a default route if that network is completely reachable at the time of writing. When “*rc.local*” executes “route” command, that network is not reachable yet (Figure 23), so we need to add “*sleep 30*” before “route” command is executed. Thus, “*rc.local*” waits 30 seconds and then it writes a new default route into the IP routing table, once the wished network is reachable.



```
SIOCADDRT: Network is unreachable
```

Figure 23. Error – Network Unreachable

To sum up, Network-Server machine is now configured and, after the system has booted up only with the internal network card detected (eth0), "*rc.local*" file waits 30 seconds until the outgoing network is reachable and then it writes the default outgoing route (through eth1) into the IP routing table.

## Network-Server – Forwarding traffic

In this final step to get Network-Server working as a router, we need to configure it to forward all the traffic. In this Ubuntu distribution, the *ip forward* byte is disabled by default because normally said option is not needed. We will enable it only in Network-Server host, which is going to work as a router.

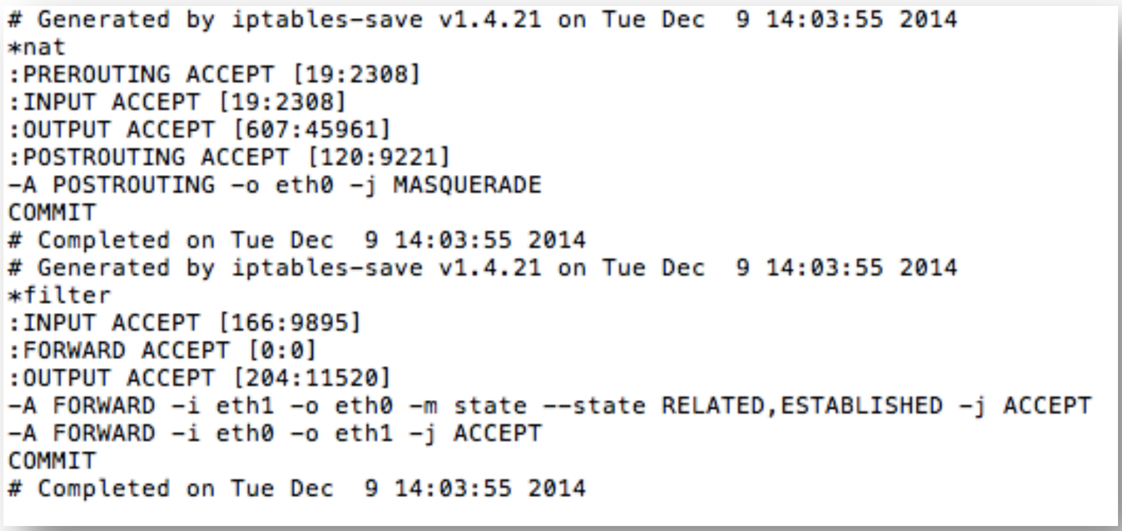
To enable this option we need to type in a terminal: "`sudo echo 1 > /proc/sys/net/ipv4/ip forward`" and we also have to uncomment the following line into the file */etc/sysctl.conf* : "`# net.ipv4.ip forward=1`". Keep in mind that if we wish to uncomment a line, we only need to remove the *#* symbol at the beginning.

## Network-Server – IP Masquerading

Once we have *ip forwarding* activated, we need to enable the IP masquerading. With this function we will make reachable the networks outside of the laboratory from the Local Area Network through the Network-Server. So Network-Server will receive traffic from the LAN through its first Network Interface Card and it will send it out from the laboratory through its second Network Interface Card.

Needed command lines for IP Masquerade activation will be described below:

```
"sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE"

"sudo iptables -A FORWARD -i eth1 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT"

"sudo iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT"
```



```
# Generated by iptables-save v1.4.21 on Tue Dec  9 14:03:55 2014
*nat
:PREROUTING ACCEPT [19:2308]
:INPUT ACCEPT [19:2308]
:OUTPUT ACCEPT [607:45961]
:POSTROUTING ACCEPT [120:9221]
-A POSTROUTING -o eth0 -j MASQUERADE
COMMIT
# Completed on Tue Dec  9 14:03:55 2014
# Generated by iptables-save v1.4.21 on Tue Dec  9 14:03:55 2014
*filter
:INPUT ACCEPT [166:9895]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [204:11520]
-A FORWARD -i eth1 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth0 -o eth1 -j ACCEPT
COMMIT
# Completed on Tue Dec  9 14:03:55 2014
```

Figure 24. IP Tables (ipv4)

As we can see in this capture (Figure 24), we have already introduced the ip tables in Network-Server, but we have to save IPTables rules permanently. Unless they are saved, they will be lost after the next system reboot, as they are stored in volatile memory. [19] We can store it with the following command line:

“`iptables-save > /etc/iptables/rules.v4`”

At last, we have Network-Server working as a full router. It differences between internal and external traffic and it will send it through one NIC or another in each case.

### 3.1.4. – Wide Area Network emulator (WANem)

#### Introduction

This part will briefly explain what are the main options to be set up in the WANem server to get it working properly. We will also describe some details, such as the version that we use or some essential options we have to configure the first time we boot this WANem server.

#### Installation

In the official web page we have two available versions: One is the stable version (v2.3) but it is quite old, it was last modified in 2011. The other one is the Beta 2 version (v3.0 Beta 2) and it was uploaded in 2014, February. Although is a Beta version, it is stable enough to work with it.

We also need this version because the other one (v2.3) is not possible to install in the hard drive, we would use it as a live distribution. It would be really useless, as each time we boot our WANem server, we would have to configure it again.

Once we have the *WANem v3.0 Beta 2* ISO file downloaded, we have to burn it into a USB drive or a CD and then we are ready to start with the installation. To install WANem into the hard drive, we have to boot live version first and then, in the desktop, we will find an option to install it. By simply by following the instructions, we should get WANem server properly installed into the hard drive.

## 3.2. – Protocols and Servers

In this part, software installation and the configuration of every protocol will be explained. First of all, we be start with the choice of the Operative System and which version we have installed. Then we will explain what protocols are needed for a functional Local Area Network, which server has been chosen for each protocol and how we can install it.

### 3.2.1. – Operative System

The laboratory will be built on an Ubuntu environment, which is a Linux distribution based on Debian [20]. The compilation number is Ubuntu 14.04.1 LTS and both x86 and x64 versions will be used, depending on the architecture of each machine. As desktop environment we will use Gnome (Figure 25) instead of Unity because our available hardware resources will work much better with it.

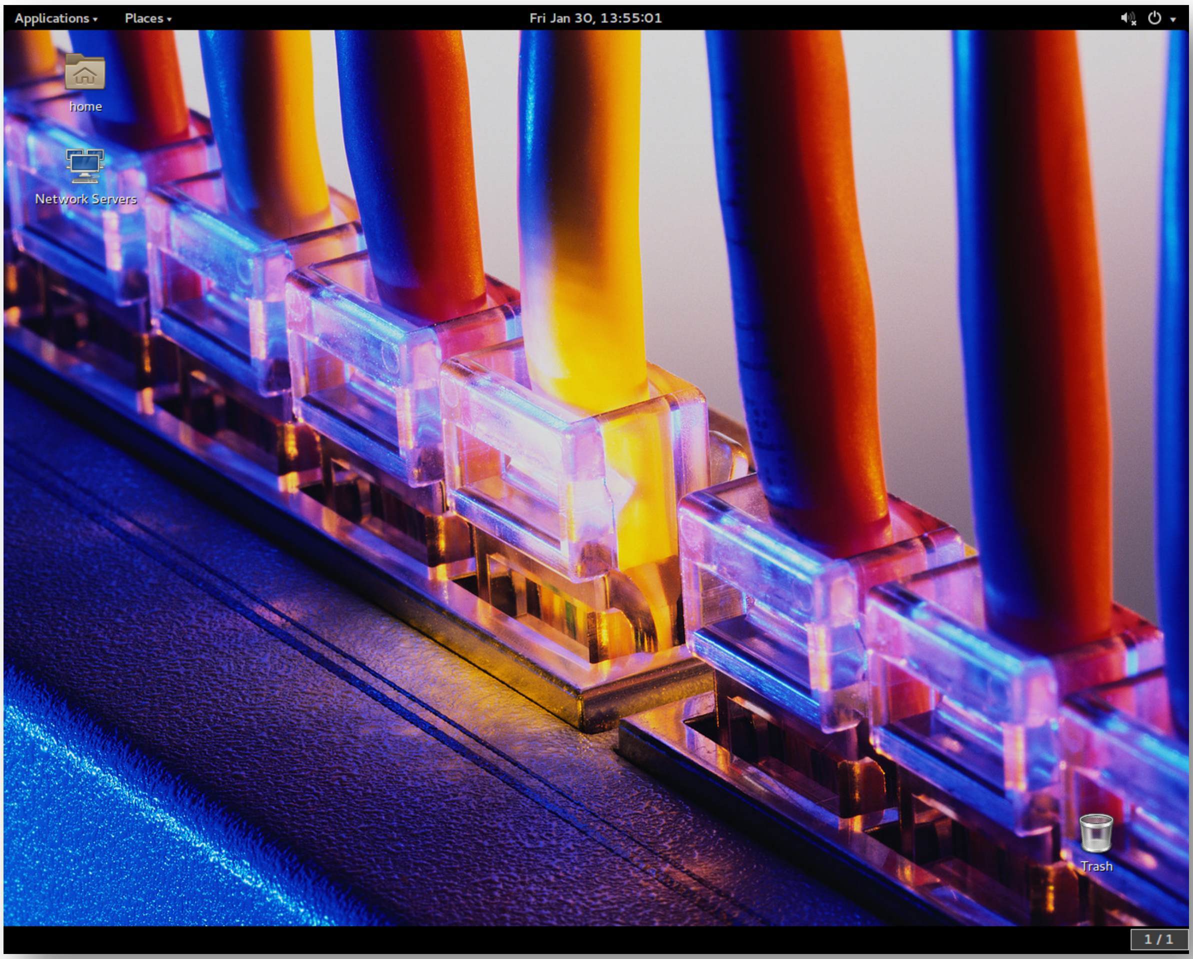


Figure 25. Ubuntu 14.04.1 LTS x64 – Gnome Desktop Environment

WANem will be the only Linux server, which will not use Ubuntu. It is distributed with its own Linux compilation based on Knoppix (although Knoppix is also based on Debian). WANem distribution (version 3.0 Beta 2) uses LXDE as its default desktop environment (Figure 26), which is a really lightweight interface. So it will run without problems in a rather old machine.

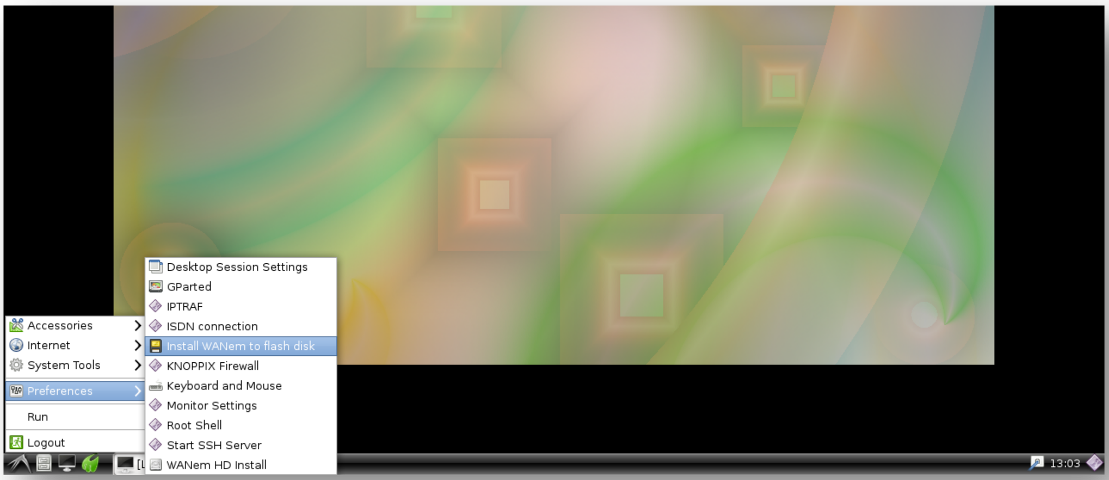


Figure 26. WANem distribution (version 3.0 Beta 2) – LXDE Desktop Environment

Dell-Vostro machine has Windows already installed with VMware as virtualization software, so the virtual machines (guests) are going to use Ubuntu as Operative System but the host runs Windows 7 x64. This will not become a problem because from the Local Area Network we will see two separated machines: a Network-Server and a Web-Server, each one with its own IP and MAC addresses.

## 3.2.2. – Servers

This section will be focused on describing the essential protocols we need for an operational Local Area Network and the main software tools we have to install to manage this protocols. Each description will consist of three different parts: an introduction to that protocol, how to install it and finally how to set it up.

### DHCP – Dynamic Host Configuration Protocol

#### *Introduction*

To implement a new network, one of the first tasks should be how to allocate and manage the IP addresses. To tackle this, we have to choose which host will handle this IP addresses and what service we need to do it. We have to install a DHCP server and configure it to work in our network. Keep in mind that DHCP (Dynamic Host Configuration Protocol) is the protocol which takes care of IP addresses management in every network.

## *Installation*

The chosen server for IP addresses management in our Local Area Network is "ISC DHCP". As we can read on its web page, it is open source software so we are free to use it on our environment. *"ISC DHCP is open source software that implements the Dynamic Host Configuration Protocol for connection to an IP network." [21]*

The chosen host to work as DHCP server has been Network-Server, one of the two virtual machines running into Dell-Vostro. This host runs "*Ubuntu 14.04.1 LTS x64*" as Operative System, which is Debian based, so to install the "ISC DHCP" server we only have to type in a terminal: "`sudo apt-get install isc-dhcp-server`" (Keep in mind that we need administrator privileges to install any server). If everything went successful, the dhcp server should now be installed and ready to be configured.

## *Configuration*

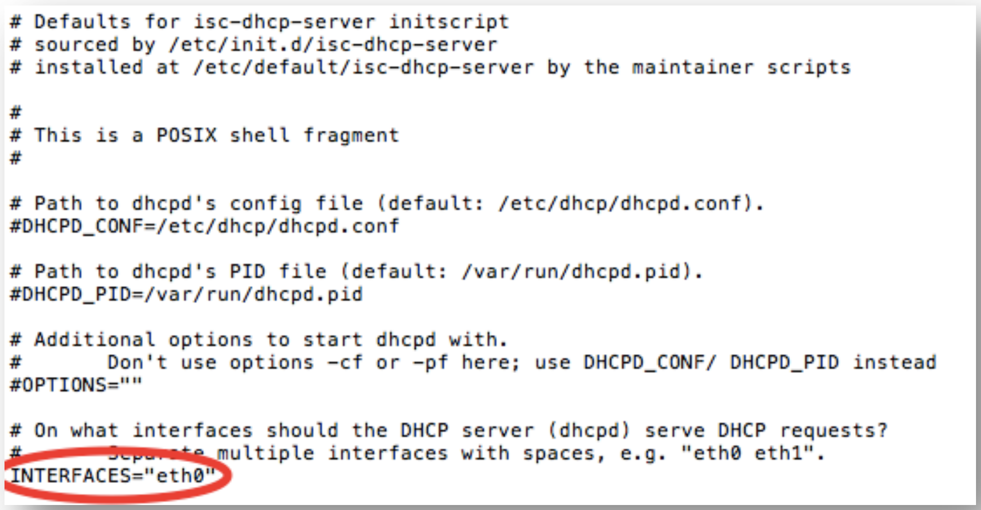
With the "ISC DHCP" server installation, many files related to this server have been created in our machine. The most important ones are:

- **/etc/dhcp/dhcpd.conf:** This is the main configuration file and in it we can modify most of the options of the DHCP server.
- **/var/lib/dhcp/dhcpd.leases:** Here we can find a log with the IP addresses allocated to each client.
- **/etc/default/isc-dhcp-server:** In this file we can configure the interface we want to be listened by this DHCP server.
- **/var/run/dhcpd.pid:** This file contains DHCP server's PID.

We have also installed in our host all the documentation related to DHCP server, so we can look up into the man pages if we have any doubt about "ISC DHCP" server's working. On these pages, we find many options as well as details and even some examples are shown.

### Default interface

First of all, as we have more than one network interface card installed in Network-Server, we need to verify DHCP server is listening on the correct one. To check this, we will open "**/etc/default/isc-dhcp-server**" and at the end of the file we could find this option. In our environment, the network interface, which is listening to the Local Area Network, is "eth0". In the following capture (Figure 27) I will show this file and the option.



```
# Defaults for isc-dhcp-server initscript
# sourced by /etc/init.d/isc-dhcp-server
# installed at /etc/default/isc-dhcp-server by the maintainer scripts

#
# This is a POSIX shell fragment
#

# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPD_CONF=/etc/dhcp/dhcpd.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPD_PID=/var/run/dhcpd.pid

# Additional options to start dhcpd with.
#       Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="eth0"
```

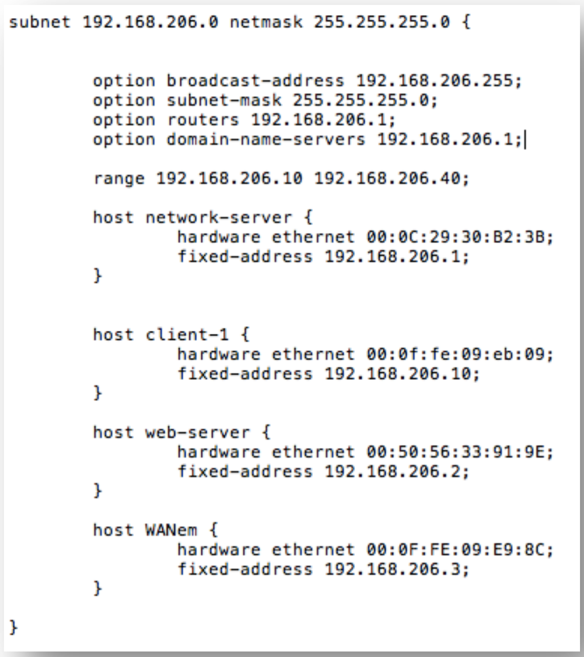
Figure 27. "**/etc/default/isc-dhcp-server**

As we observe in Figure 27, we have set up our DHCP server to listen on interface "eth0". It means that DHCP will only assign IP addresses if the requests come from the Local Area Network, and not from the outside of the laboratory.

This step could seem unnecessary, however if we send IP addresses outside of the laboratory, we may cause a big problem on the rest of University's network. Keep in mind that in a subnet there can only be one DHCP primary server and if this rule is omitted, many IP addresses conflicts will appear. In practice, this problem can be also solved by a configurable switch, which could be configured to reject DHCP assignments if they are coming from a non-allowed port.

## DHCPD.CONF

Now we have to set up the main configuration file of this server: "dhcpd.conf". As we have seen before, this file is located on ***"/etc/dhcp/dhcpd.conf***" (although this location might change depending on the Linux distribution we are using). In the next image (Figure 28), we will see how I have configured this file to manage and allocate IP addresses in our Local Area Network:



```
subnet 192.168.206.0 netmask 255.255.255.0 {


        option broadcast-address 192.168.206.255;
        option subnet-mask 255.255.255.0;
        option routers 192.168.206.1;
        option domain-name-servers 192.168.206.1;

        range 192.168.206.10 192.168.206.40;

        host network-server {
                hardware ethernet 00:0C:29:30:B2:3B;
                fixed-address 192.168.206.1;
        }


        host client-1 {
                hardware ethernet 00:0f:fe:09:eb:09;
                fixed-address 192.168.206.10;
        }

        host web-server {
                hardware ethernet 00:50:56:33:91:9E;
                fixed-address 192.168.206.2;
        }

        host WANem {
                hardware ethernet 00:0F:FE:09:E9:8C;
                fixed-address 192.168.206.3;
        }

}
```

Figure 28. */etc/dhcp/dhcpd.conf*

The first thing we have to specify is the subnet's IP address, in this case IP address is "192.168.206.0" and the netmask we use is "255.255.255.0". Then we will write some needed DHCP options, such as broadcast address, subnet mask, router address and domain name server address.

After that, we will define the DHCP assignment's range, simply typing "range" and then the first IP address followed by the last IP address of the chosen range.

Finally we are going to declare fixed IP addresses, which will always be assigned to the same host. It is a useful option and allows us to identify the most important hosts or servers in our network. These machines with a fixed IP address will be Network-Server, Web-Server and the WANem host. (Here we have also given client1 a fixed IP address to facilitate its identification in the different Wireshark tests we have done, but it is not mandatory).

## Starting DHCP server

Now we have everything configured properly, so we are ready to start the DHCP server. We can reboot the Network-Server to load the entire new DHCP configuration, but we can also do it restarting DHCP service. If we select this second option, we can simply type on a Terminal: "`sudo /etc/init.d/isc-dhcp-server restart`". After that, DHCP server is running with the new configuration.

# DNS – Domain Name Server

## *Introduction*

The next step, after setting up the DHCP server, is to install a DNS server. This service will be responsible for associating IP addresses and hosts' names. This will make easier for us to identify the hosts for tasks like network maintenance. Strictly speaking, DNS is not a mandatory service for a proper operation in a Local Area Network, but due to ease of memorize names instead of number sequences, it is almost impossible nowadays to find any network without a DNS server implemented.

## *Installation*

We have also chosen the host Network-Server to be our LAN's Domain Name Server, so we need a software tool which allows us to implement this server in our network. We can find many different tools to work as a DNS server, but in this case we have chosen "bind9" to do it. This is the most the most widely used DNS software on the Internet, providing a robust and stable platform on top of which organizations can build distributed computing systems with the knowledge that those systems are fully compliant with published DNS standards. BIND is open source software that implements the Domain Name System (DNS) protocols for the Internet. [22]

Network-Server is running Ubuntu 14.04.1 LTS x64 and we are going to use "bind9" as DNS server, which is available into the Ubuntu repositories. So to install "bind9" we only have to type "sudo apt-get install bind9" in a terminal. After that, all needed software will be installed in Network-Server and ready to be configured.

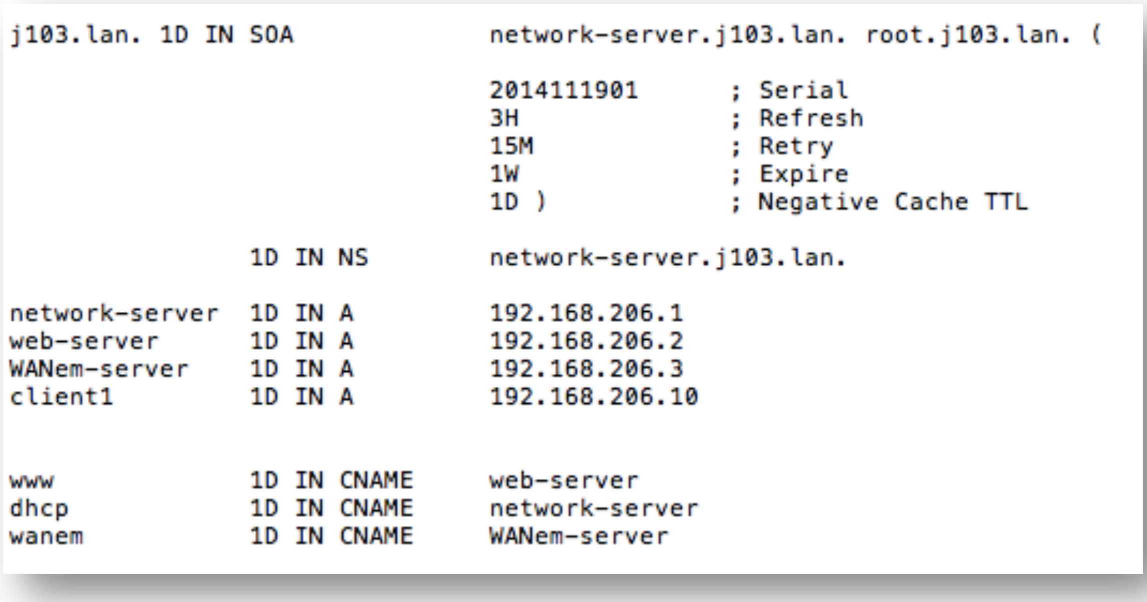
## *Configuration*

After the installation, we have to set up some DNS files to get our Domain Name Server working properly. Most of them have been created in *"/etc/bind/"* and we will focus on "*named.conf.local*" and "*db.local*".

First of all we create a copy of "*db.local*" but changing its name to "*db.j103*" and we can do it typing "`cp db.local db.j103`". We have chosen "*db.j103*" but any other name could be valid, provided that you change it in "*named.conf.local*", as we will see later.

### Zone Files

Once we have our own database file ("*db.j103* ") we have to set it up properly. In the following capture we see how it has been configured for our environment. Keep in mind that DNS is an essential protocol for Internet operation and it has a huge amount of options available. Here we will set up DNS for a local environment, so we are going to use the basic ones.



```
j103.lan. 1D IN SOA          network-server.j103.lan. root.j103.lan. (

                             2014111901       ; Serial
                             3H               ; Refresh
                             15M              ; Retry
                             1W               ; Expire
                             1D )             ; Negative Cache TTL

                1D IN NS     network-server.j103.lan.

network-server  1D IN A      192.168.206.1
web-server      1D IN A      192.168.206.2
WANem-server    1D IN A      192.168.206.3
client1         1D IN A      192.168.206.10


www             1D IN CNAME  web-server
dhcp            1D IN CNAME  network-server
wanem           1D IN CNAME  WANem-server
```

Figure 29. DNS zone File

At the beginning of this file (Figure 29) we find the domain name, in this case "j103.lan." Then SOA (Start of the Authority) parameter specifies authoritative information about a DNS zone. After that we can find DNS server's name followed by the domain name (nertwork-server.j103.lan.) and DNS administrator's name. Note final dot in the name server declaration is essential to point the end of this domain.

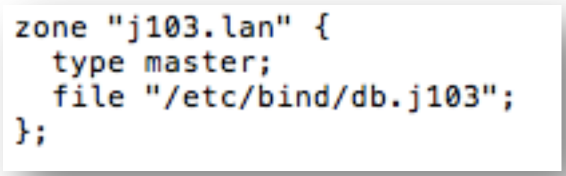
On the following lines we will find a "*serial*" number, which indicates the last time this file was modified. Then we have "*refresh*", "*retry*", "*expire*" and "*TTL*" options and their functions could be checked in the DNS documentation files.

After these lines, we find this file's core: which IP address corresponds with which name server. Here we have to specify again that the NS (Name Server) of the DNS host is "*network-server.j103.lan.*". Then we see four IP addresses (A) and their four name servers.

Finally we have another block with the Canonical Names (CNAME), which are alias of one name to another: the DNS lookup will continue by retrying the lookup with the new name. This option is used to point which host is working as web-server, so when someone types on a web browser "*www.j103.lan*" he will be redirected to "*web-server.j103.lan*", which corresponds to "192.168.206.2" IP address.

## Named.conf.local File

Once we have our zone file completely configured and ready to work, it's time to set up "/etc/bind/named.conf.local". Here we have to indicate where the zone file is which we have created before. In the following capture we see this file configured for our environment.



```
zone "j103.lan" {
  type master;
  file "/etc/bind/db.j103";
};
```

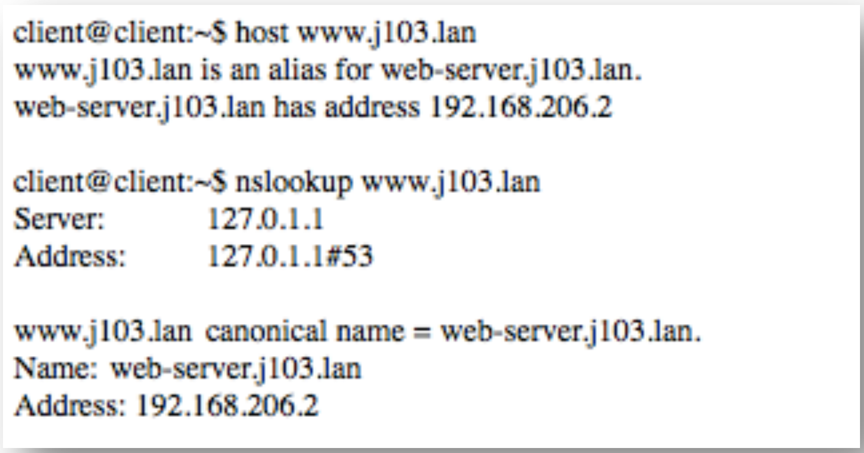
Figure 30. DNS file – named.conf.local

As we observe in Figure 30, we need to specify where the proper zone file is located. In our local environment we will translate name servers into IP addresses, so we have only configured DNS direct zone file. DNS have also reverse translation (IP address to domain name mapping) but this configuration is not going to be detailed here.

## Starting DNS server

Once we have our zone file created and properly imported into "*named.conf.local"* we are able to restart the DNS service, in order to get the new configuration loaded. To restart this service we only need to type "`sudo /etc/init.d/bind9 restart`".

Now we have our DNS server working and ready to translate name servers into IP addresses. If we want to check if DNS server is working properly, we only have to type in a Terminal "`host www.j103.lan`". We then see which is the IP address of this server and also that "*www*" is a Canonical Name from "*web-server*" (Figure 31).



```
client@client:~$ host www.j103.lan
www.j103.lan is an alias for web-server.j103.lan.
web-server.j103.lan has address 192.168.206.2

client@client:~$ nslookup www.j103.lan
Server:         127.0.1.1
Address:        127.0.1.1#53

www.j103.lan  canonical name = web-server.j103.lan.
Name:  web-server.j103.lan
Address: 192.168.206.2
```

Figure 31. Terminal "`host www.j103.lan`"

## Web Server – Apache, PHP and MySQL

The web server will be the main target in all of our tests. In the *Wireshark Part* we will use some tools, such as *Traceroute* or *ping* to test the WANem server's behaviour with traffic going from clients towards the web server. We will use a complete LAMP server (Linux, Apache, MySQL and PHP), although in this scenario MySQL will not be used. It has been installed for a future use.

### *Apache Web Server:*

First of all we will install the web server's base. To install the Apache server we only have to type "`sudo apt-get install apache2 -y`" in a Terminal and all the needed files will be downloaded in our machine. After that, we can check if the installation was successful, typing in a web browser the loopback IP address (http://127.0.0.1). If everything went fine, we should see a web page from Apache confirming us that "*it works*".

As we can read on its official web page: "The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems. The goal of this project is to provide a **secure, efficient and extensible server** that provides HTTP services in sync with the current HTTP standards." [23] We could find many modules to extend this server and a lot of sophisticated security options, but in this project we are going to use only the default options. That will be enough for our goals.

### *PHP 5 module:*

Now we will install the PHP 5 module for the web server. It could be done simple typing "`sudo apt-get install php5 libapache2-mod-php5 -y`" in a Terminal. After that we need to restart the whole web server ("`sudo /etc/init.d/apache2 restart`"). We could check everything is working properly writing on a web browser: "http://127.0.0.1/php.info". Then we should see all the PHP modules installed.

### *MySQL (Client and Server):*

Finally we need to install MySQL client and server in the web server machine. We only have to write in a terminal "`sudo apt-get install mysql-server mysql-client -y`". It will ask us for a root password and then everything will be downloaded and installed.

As we saw with Apache, MySQL also has many different security options and modules. If we want to install a complete MySQL environment, we should also download some extra modules. Typing in a terminal the following command line, all of them would be installed "`sudo apt-get install php5-mysql php5-curl php5-gd php5-idn php-pear php5-imagick php5-imap php5-mcrypt php5-memcache php5-ming php5-ps php5-pspell php5-recode php5-snmp php5-sqlite php5-tidy php5-xmlrpc php5-xsl -y.`"

### *PHP myAdmin:*

Another interesting module to be installed is the MySQL and PHP administration's interface. It allows us to manage different options and user accounts related to php and

MySQL administration. We have to write the following command line "`sudo apt-get install phpmyadmin -y`" and the administration's interface will be downloaded. It will ask which server we have so we specify "*apache2*". Now we can manage php and MySQL options writing on a web browser: *http://127.0.0.1/phpmyadmin*.

## IP – Internet Protocol

The last protocol we need to set up will be the Internet Protocol but here, strictly speaking, we don't have to "install" any software to manage it. All commands and tools needed are installed by default, because this is an essential protocol. Unlike "ISC DHCP" or "BIND 9", which are only installed in servers, IP manage tools (i.e. "*route*" or "netstat" commands) are already installed when you boot the Operative System for the first time.

### *IP Routing Table*

In the *Wireshark labs* section we will need, for the majority of the tests, traffic going from client to server and from server to client through the WANem host. These rules are stored in the IP routing table and each machine always looks into this table before sending any traffic. Therefore, it is essential to have this tables properly configured if we want to be sure where we are sending the traffic through.

The IP routing tables could be checked simply typing in a terminal "*`netstat -r`*" (Figure 32). In the following capture we have a client table of our environment:

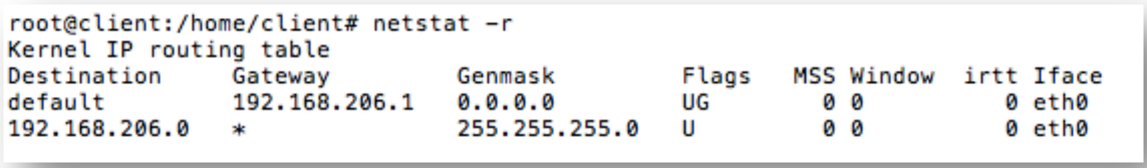
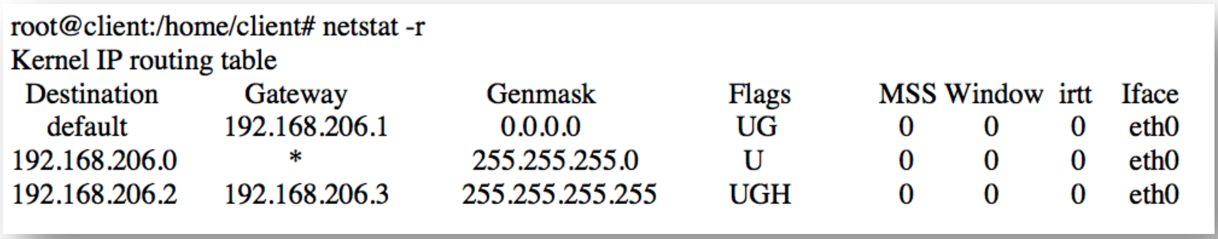

```
root@client:/home/client# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         192.168.206.1   0.0.0.0         UG        0 0          0 eth0
192.168.206.0   *               255.255.255.0   U         0 0          0 eth0
```

Figure 32. Client – "*`netstat -r`*"

As we can observe in the capture above, in this table here are two inputs. They are the system default entries when we have our host connected to a Local Area Network:

- *"192.168.206.0"* : this rule indicates the client where it has to send the traffic, which goes to a machine on the same Local Area Network. It means such traffic goes directly to the destination host.
- *"default"* : with this rule, the table tells the host what is the default gateway where it has to send the traffic to, when this traffic goes outside the Local Area Network. In this case, the default gateway is the Network-Server's IP address because this machine is the one who is working as Router for this LAN.

The IP routing table could be easily modified with the command "*route*". It allows us to add or remove table's entries simply writing "add" or "del", as appropriate. Below we see how to add a rule into the IP routing table to send traffic through a specific host.



```
root@client:/home/client# netstat -r
Kernel IP routing table
 Destination      Gateway          Genmask          Flags    MSS Window  irtt  Iface
   default      192.168.206.1       0.0.0.0          UG        0      0     0   eth0
192.168.206.0        *           255.255.255.0       U         0      0     0   eth0
192.168.206.2   192.168.206.3   255.255.255.255      UGH       0      0     0   eth0
```

Figure 33. Client – Modified IP routing table

As we can observe in this capture (Figure 33), we have modified the IP routing table by adding a new rule. When the client is going to send traffic to the local web server (192.168.206.2), this rule indicates where it has to be sent through. To add this rule into the table we only have to type in a Terminal the following command line:

```
“sudo route add -host 192.168.206.2 gw 192.168.206.3”
```

Here we have the command “route” followed by “add” option. Then we have to specify if we are adding a host or a network rule with its IP address and after that we will specify the gateway. This gateway should be the WANem server (192.168.206.3), which is going to forward the traffic to the final destination, the local web server (192.168.206.2).

If we want to add this rule into the web-server’s IP routing table, we only have to write there the properly rule. We would change the –host IP address with the client1 IP address, so we would have:

```
“sudo route add -host 192.168.206.10 gw 192.168.206.3”
```

## 3.3. – Conclusions

In this second section, we have built a complete environment with the available hardware. We have also set up all the needed software, such as protocols, servers or network tools, so now we are ready to begin with the network tests. It is important to highlight that the servers have been configured with the basic options. If we need to modify an option or add any other module, it will be explained when required.

# 4. – Wireshark Labs

*This section will be focused on network tests. First of all, we will remind briefly the laboratory's environment and then we will perform some tests making use of networking tools, such as Traceroute, Ping or Wireshark. After each test, we will analyze the results, trying to explain why we have obtained it.*

*We will begin with the environment, where we could see again the laboratory's structure, with the IP addresses of each client or server.*

*After that, we will see the main functions in the Wide Area Network emulator. This kind of software will give us the possibility of emulating a real WAN with its delays, jitter or packet loss.*

*Then we could start with the network tests. For these tests we will use command lines, written in a Terminal, such as Traceroute or Ping. Another useful tool will be Wireshark, packet sniffer software, which will allows us to capture and analyze every packet in our LAN.*

## 4.1. – THE ENVIRONMENT

We begin this part remembering the laboratory's environment and its most important hosts for our Wireshark tests and analysis. Here we have the Local Area Network's diagram (Figure 34). The network's core it's the Dell Vostro machine, which is running two virtual machines at the same time: Network-Server and Web-Server. Each virtual machine has its own IP address.

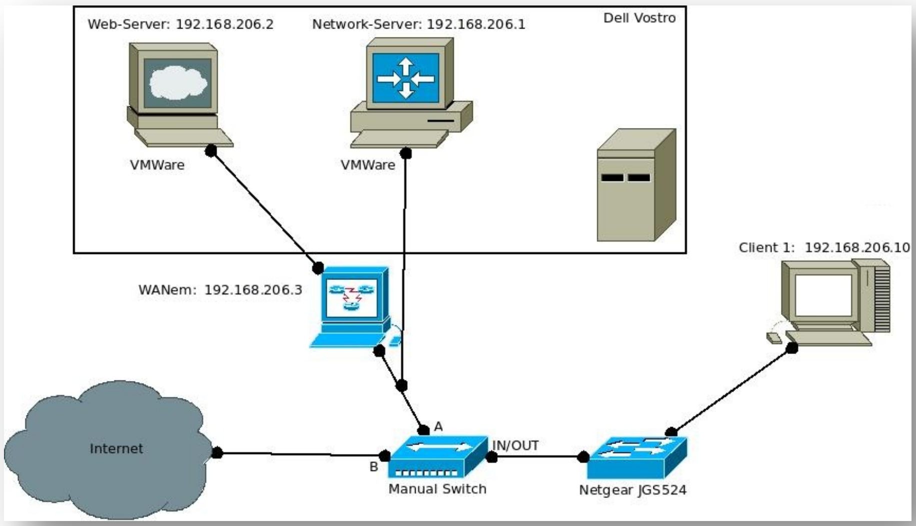


Figure 34. LAN's diagram

In our environment clients are connected to the web-server through the WANem host, which will help us to emulate a real environment. With it we would be able to modify some characteristics such as delays, jitter or packet loss in the connections between clients and web-server.

The other devices to be found in this environment are a manual switch and a non-configurable switch. With the manual switch we can choose if we want to work with the local area network or with Internet. It's the easiest way to do this, but it can also be done by editing the IP routing table in Network-Server host. The plug & play switch is connecting the clients to one network or another (it depends on the first switch choice).

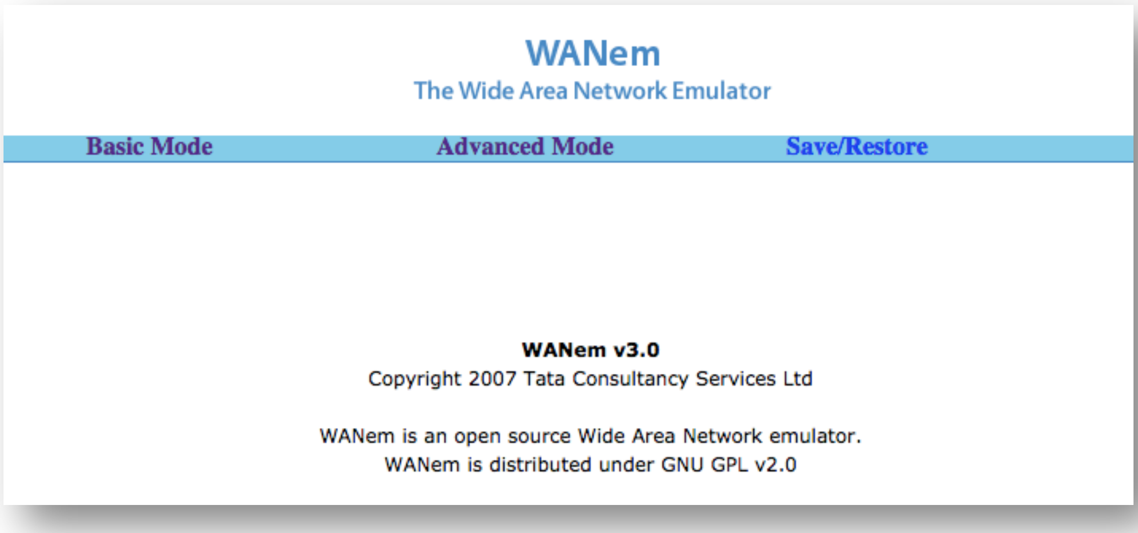
Notice that network-server is the only host that is not connected through the WANem server. That's because this machine provides IP addresses and resolve DNS queries. It would be many problems if DHCP messages exchanged were delayed or even lost.

## 4.2. – WANem Interface

Before beginning with Wireshark tests we need to know, how the WANem interface is and what options do we have. We will see the Home Screen and then two different modes: basic and advanced.

### 4.2.1. – Home Screen

To access into the WANem configuration, we need to type the WANem IP address in a web browser followed by *"/WANem"* (in this case *"192.168.206.3/WANem"*). Once we have typed it, the WANem's home screen will load and now we are connected to the WANem's configuration interface.


WANem
The Wide Area Network Emulator

WANem v3.0
Copyright 2007 Tata Consultancy Services Ltd

WANem is an open source Wide Area Network emulator.
WANem is distributed under GNU GPL v2.0

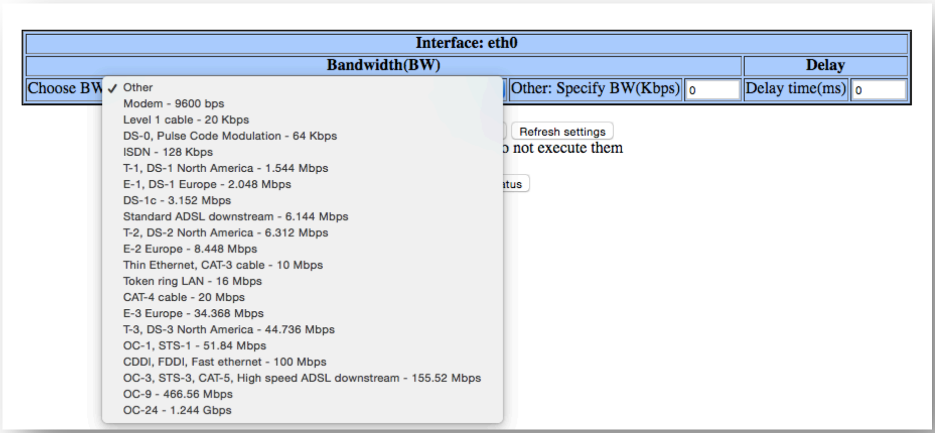
Figure 35. WANem Home Screen

In the main screen (Figure 35) we can find the title, which identifies this interface with the *"Wide Area Network Emulator"*, three different menus (Basic Mode, Advanced Mode and Save/Restore). We can also find the WANem version (in this case WANem v3.0), the

Copyright license and finally an Open Source description telling us that WANem is distributed under GNU GPL v2.0.

### 4.2.2. – Basic Mode

If we go into the basic mode (Figure 36), we can change the bandwidth of our communication or add some delay. If we open the list, we can choose among many different bandwidths, but we can also specify it manually into the right text box. The other option we are able to find in this basic mode is to add delay to our communication. In this case we don't have any list with delays, we have to type it into the delays text box.



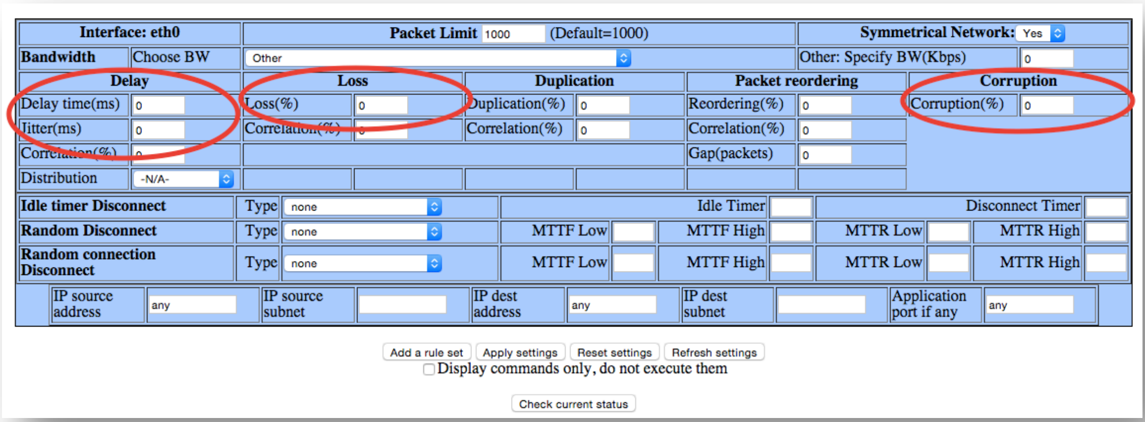
| Interface: eth0 | | | | |
|---|---|---|---|---|
| Bandwidth(BW) | | | Delay | |
| Choose BW | ✓ Other | Other: Specify BW(Kbps) 0 | Delay time(ms) | 0 |

- ✓ Other
- Modem - 9600 bps
- Level 1 cable - 20 Kbps
- DS-0, Pulse Code Modulation - 64 Kbps
- ISDN - 128 Kbps
- T-1, DS-1 North America - 1.544 Mbps
- E-1, DS-1 Europe - 2.048 Mbps
- DS-1c - 3.152 Mbps
- Standard ADSL downstream - 6.144 Mbps
- T-2, DS-2 North America - 6.312 Mbps
- E-2 Europe - 8.448 Mbps
- Thin Ethernet, CAT-3 cable - 10 Mbps
- Token ring LAN - 16 Mbps
- CAT-4 cable - 20 Mbps
- E-3 Europe - 34.368 Mbps
- T-3, DS-3 North America - 44.736 Mbps
- OC-1, STS-1 - 51.84 Mbps
- CDDI, FDDI, Fast ethernet - 100 Mbps
- OC-3, STS-3, CAT-5, High speed ADSL downstream - 155.52 Mbps
- OC-9 - 466.56 Mbps
- OC-24 - 1.244 Gbps

Figure 36. WANem Basic Mode

### 4.2.3. – Advanced Mode

On the other hand, if we choose the Advanced Mode (Figure 37), we will find many more options, such as random disconnection, IP addresses restrictions, symmetrical or asymmetrical network, packet reordering and many others. We can also specify the bandwidth in our communication, as in the basic mode but for our tests we will focus on delay, jitter, loss and corruption.



| Interface: eth0 | | Packet Limit 1000 (Default=1000) | | | | | | Symmetrical Network: Yes | |
|---|---|---|---|---|---|---|---|---|---|
| Bandwidth | Choose BW | Other | | | | | | Other: Specify BW(Kbps) | 0 |
| Delay | | Loss | | Duplication | | Packet reordering | | Corruption | |
| Delay time(ms) | 0 | Loss(%) | 0 | Duplication(%) | 0 | Reordering(%) | 0 | Corruption(%) | 0 |
| Jitter(ms) | 0 | Correlation(%) | 0 | Correlation(%) | 0 | Correlation(%) | 0 | | |
| Correlation(%) | 0 | | | | | Gap(packets) | 0 | | |
| Distribution | -N/A- | | | | | | | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Idle timer Disconnect | Type | none | Idle Timer | | Disconnect Timer | | | |
| Random Disconnect | Type | none | MTTF Low | | MTTF High | | MTTR Low | MTTR High |
| Random connection Disconnect | Type | none | MTTF Low | | MTTF High | | MTTR Low | MTTR High |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| IP source address | any | IP source subnet | | IP dest address | any | IP dest subnet | | Application port if any | any |

Figure 37. WANem Advanced Mode

Once we have chosen basic or advanced mode and we have configured each option, we are ready to apply settings and start with the WANem tests.

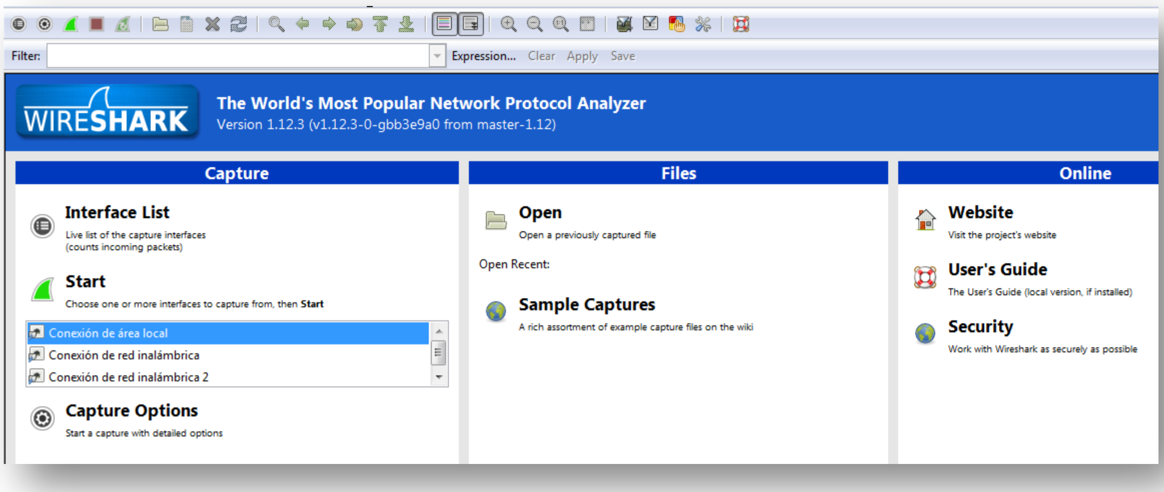
## 4.3. – Wireshark Tests

After this review of laboratory's environment and the WANem interface, we are ready to begin with the Wireshark tests. First of all I am going to explain Wireshark tool (interface and main configuration) and then some tests will be done, focusing on different protocols each time. The first one will be IP (Internet Protocol), then TCP (Transmission Control Protocol), after that we will see some DNS captures (although DNS is not passing through the WANem host) and finally we can test HTTP behaviour under different conditions.

### 4.3.1. – GETTING STARTED

Wireshark will be our basic work tool in this third section. *There are many other packet sniffer software, but we've chosen Wireshark because it's Open Source, cross-platform and it has an intuitive and powerful interface with many filter options (by protocol, by port, by IP address source or destination, by MAC address…)*

First we need to know Wireshark, its interface and its options. In the next capture (Figure 38) we can see the main screen, which appears when we just open the application.


Filter: Expression... Clear Apply Save

The World's Most Popular Network Protocol Analyzer
Version 1.12.3 (v1.12.3-0-gbb3e9a0 from master-1.12)

Capture

Interface List
Live list of the capture interfaces (counts incoming packets)

Start
Choose one or more interfaces to capture from, then Start

Conexión de área local
Conexión de red inalámbrica
Conexión de red inalámbrica 2

Capture Options
Start a capture with detailed options

Files

Open
Open a previously captured file

Open Recent:

Sample Captures
A rich assortment of example capture files on the wiki

Online

Website
Visit the project's website

User's Guide
The User's Guide (local version, if installed)

Security
Work with Wireshark as securely as possible

Figure 38. Wireshark main screen

Wireshark is a packet sniffer, which allows us to capture messages from the network and display them. It consists of two parts: the **packet capture library** and the **packet analyzer**. The next image (Figure 39) shows a diagram of a packet sniffer and how they work.

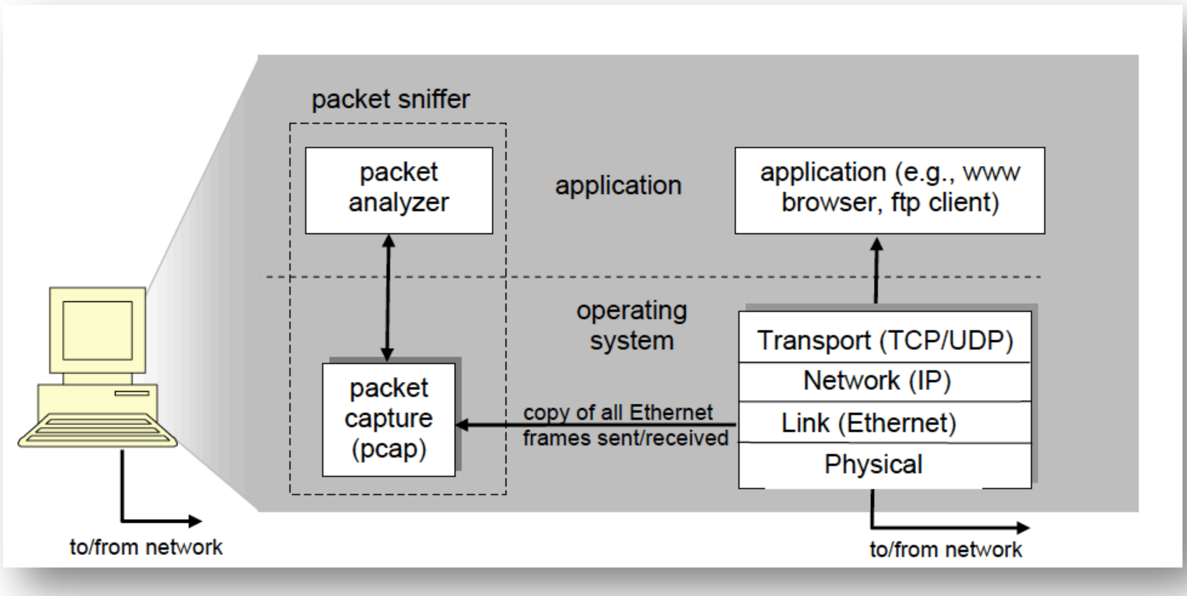


Figure 39. Packet Sniffer diagram [1]

The first part, the packet capture library (pcap), receives a copy of every frame transmitted over a physical media such as an Ethernet cable. These frames are transmitted to the packet analyzer, which understands this frames and displays to the user.

Now, we're going do some tests with Wireshark and then we will see in detail how do these protocols behave in our environment. The tests will be focused on following protocols:

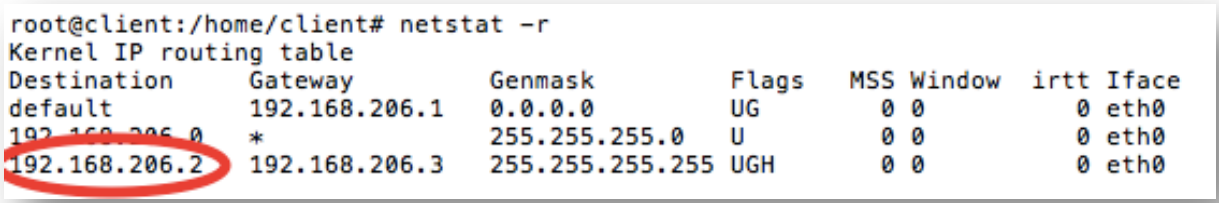
- IP: Network layer
- TCP: Transport layer
- DNS and HTTP: Application layer

### 4.3.2. – IP (Internet Protocol)

IP, Internet Protocol, takes care of the communication between computers. It is responsible for addressing, sending and receiving the data packets over the Internet. When an IP packet is sent from a computer, it arrives at an IP router. The IP router is responsible for "routing" the packet to the correct destination, directly or via another router.

#### Connection delays

We use *Traceroute* diagnostic tool to check how many hops we have between the web server and our host. First, we need to check if the IP routing tables in Client1 and in the Web-Server have the proper rules configured. Keep in mind that if we want to see the IP routing table, we could obtain it just by typing the command "`netstat -r`" in a Terminal and then following capture will appear:

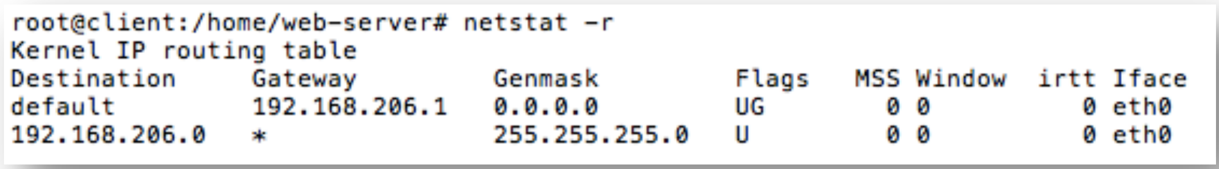


```
root@client:/home/client# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         192.168.206.1   0.0.0.0         UG        0 0          0 eth0
192.168.206.0   *               255.255.255.0   U         0 0          0 eth0
192.168.206.2   192.168.206.3   255.255.255.255 UGH       0 0          0 eth0
```

Figure 40. IP routing table with WANem

In this capture (Figure 40) we can see three rules. One for the traffic, which is going to the same LAN (192.168.206.0); another one for the traffic, which goes outside of this LAN (default) and last but not least the most important rule for us is the third rule; configured on Client1 to send the traffic, which is going to the Web-Server (192.168.206.2) **through** the WANem host (192.168.206.3).

Once we have checked Client1 has its IP routing table properly configured, we need to check the same table in the Web-Server. We can do it typing the same command as in Client1 ("`netstat -r`"). We will then see the following capture:

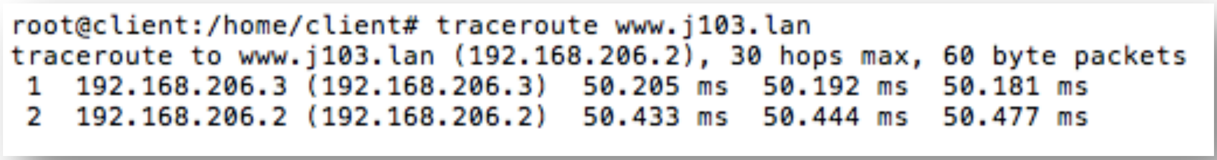


```
root@client:/home/web-server# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         192.168.206.1   0.0.0.0         UG        0 0          0 eth0
192.168.206.0   *               255.255.255.0   U         0 0          0 eth0
```

Figure 41. IP routing table in Web-Server without WANem rule

As we can observe in Figure 41, in the Web-Server host we don't have any WANem rule yet. We only have two rules: one for the traffic outgoing of the LAN and one for the internal LAN traffic. That traffic, which this host has to send to the same LAN, is going directly to its destination.

To sum up, Client1 sends its traffic to Web-Server through the WANem host, but the Web-Server sends its traffic to Client1 directly. With this scenario, if we set up a 50ms delay on WANem host, we will have the next Traceroute capture:



```
root@client:/home/client# traceroute www.j103.lan
traceroute to www.j103.lan (192.168.206.2), 30 hops max, 60 byte packets
 1  192.168.206.3 (192.168.206.3)  50.205 ms  50.192 ms  50.181 ms
 2  192.168.206.2 (192.168.206.2)  50.433 ms  50.444 ms  50.477 ms
```

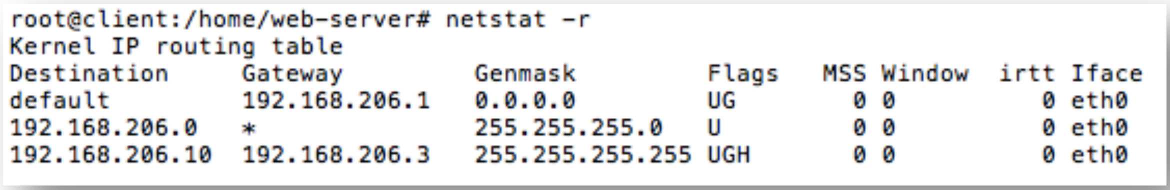
Figure 42. Traceroute: Client1 ➔ WANem ➔ Web-Server ➔ Client1

Here (Figure 42) we have the *Traceroute* working properly. Client1 is sending all the traffic to Web-Server through WANem, and this one is delaying it by 50ms. The second packet has the same delay as the first one because the Web-Server host is sending its answer directly to us.

Now we set up a new rule in Web-Server's IP routing table which will send all the traffic to Client1 through WANem host, just as Client1 does it. Keep in mind that to add this rule we only have to type:

```
"root@client:/home/web-server# route add -host 192.168.206.10 netmask 0.0.0.0 gw 192.168.206.3"
```

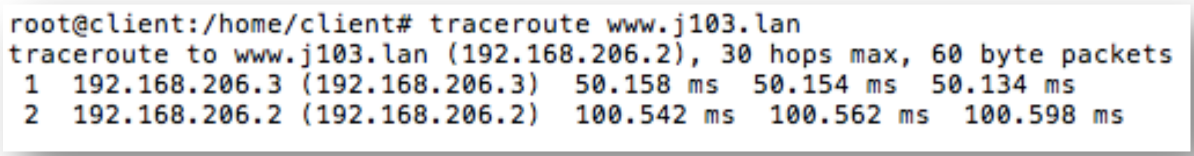
Once we have introduced the previous command, we can check the rule has been successfully added by simply typing again "`netstat -r`":



```
root@client:/home/web-server# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         192.168.206.1   0.0.0.0         UG        0 0          0 eth0
192.168.206.0   *               255.255.255.0   U         0 0          0 eth0
192.168.206.10  192.168.206.3   255.255.255.255 UGH       0 0          0 eth0
```

Figure 43. IP routing table in Web-Server with WANem rule

We already have set up both Client1 and Web-Server with the proper rules to send the traffic to each other through the WANem host (Figure 43). Now we are going to test this scenario and we will see how this new path affects Traceroute's delay.



```
root@client:/home/client# traceroute www.j103.lan
traceroute to www.j103.lan (192.168.206.2), 30 hops max, 60 byte packets
 1  192.168.206.3 (192.168.206.3)  50.158 ms  50.154 ms  50.134 ms
 2  192.168.206.2 (192.168.206.2)  100.542 ms  100.562 ms  100.598 ms
```

Figure 44. Traceroute: Client1 ➔ WANem ➔ Web-Server ➔ WANem ➔ Client1

As we can observe in this capture (Figure 44), in contrast to the previous scenario, the second packet has duplicated its delay. That is because of WANem host is delaying the traffic in both directions. If we have a look at Traceroute's operating mode, we are able to understand this duplicated delay.

Traceroute, by default, sends a sequence of User Datagram Protocol (UDP) packets addressed to a destination host. The time-to-live (TTL) value is used to know how many routers are between the host that sends the Traceroute and the destination. Routers decrement packets' TTL value by 1 when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded.

Traceroute works by sending packets with gradually increasing TTL value, starting with TTL value of 1. The first router receives the packet, decrements the TTL value and drops the packet because it then has TTL value zero. The router sends an ICMP Time Exceeded message back to the source. The next set of packets are given a TTL value of 2, so the first router forwards the packets, but the second router drops them and replies with ICMP Time Exceeded.

So when the WANem host sends the second Traceroute packet from Client1 to Web-Server, this packet has a delay of 50ms. Then Web-Server host receives this second Traceroute packet, decrements the TTL value and drops it. And here we have the "problem" with the duplicated delay. When Web-Server sends back the ICMP Time Exceeded message, this message is sent also through the WANem host (unlike in the first scenario, where it was sent directly to Client1). WANem host receives the ICMP Time Exceeded message, it delays this message by 50ms (added to 50ms this packet already had) and then it is sent to Client 1.

That is the reason of this duplicated delay in the second packet of the Traceroute: 50ms when Traceroute going to Web Server + 50ms when ICMP message going back to Client1. It could seem confusing, but if you know how Traceroute works, it is completely understandable.

We can check, as expected, the behaviour is the same even if we change the delay in the WANem host:

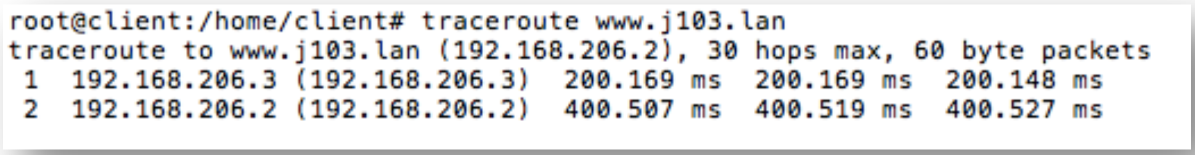

```
root@client:/home/client# traceroute www.j103.lan
traceroute to www.j103.lan (192.168.206.2), 30 hops max, 60 byte packets
 1  192.168.206.3 (192.168.206.3)  200.169 ms  200.169 ms  200.148 ms
 2  192.168.206.2 (192.168.206.2)  400.507 ms  400.519 ms  400.527 ms
```

Figure 45. Traceroute: Client1 ➔ WANem ➔ Web-Server ➔ WANem ➔ Client1

In this case (Figure 45) we have configured a 200ms delay in the WANem host, and as we can observe in this capture, Traceroute tells us the second packet has also double delay compared to the first one.

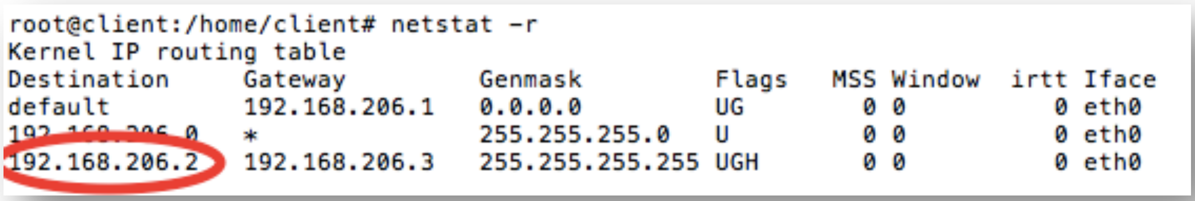
To sum up we have our WANem host delaying the traffic between Client1 and Web-Server in both directions.

## Packet Loss

Now we are going to see how WANem works with packet loss. For this task we will use Ping (instead of Traceroute), which will show us the average rate of packets lost.

Ping is a computer network administration utility used to test the reachability of a host on an Internet Protocol (IP) network. With it we can also measure the round-trip time for messages sent from the source to a destination computer and estimate the average rate of messages, which arrive to the destination.

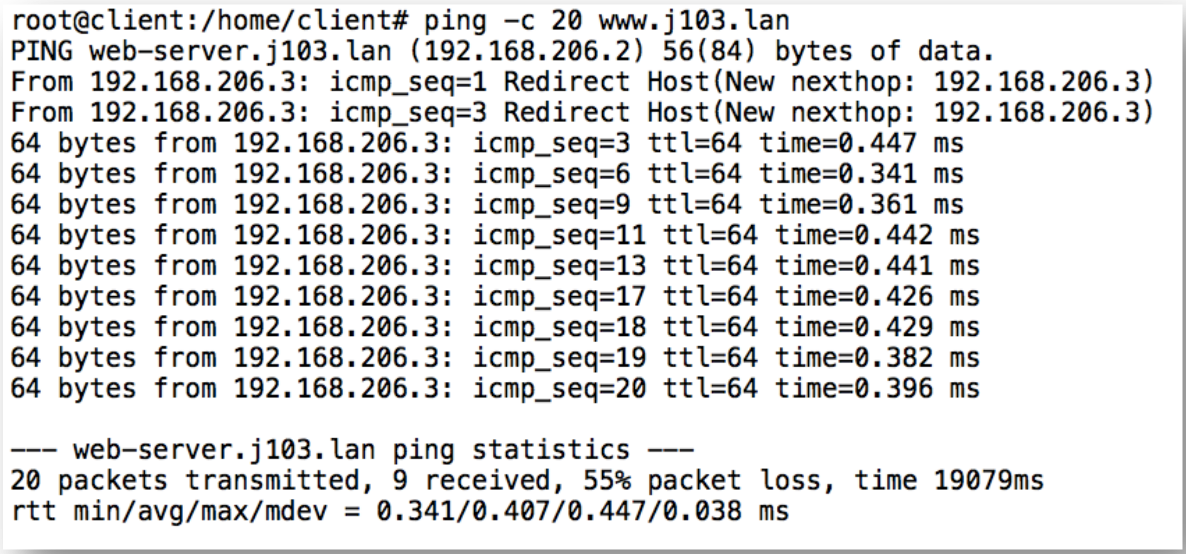
For this test we will also have to set up the IP routing table to send the traffic from Client1 to Web-Server through the WANem host. We can verify the IP routing table just by typing “`netstat -r`”. If we have a capture as the following, then we could start with the PING tests.



```
root@client:/home/client# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         192.168.206.1   0.0.0.0         UG        0 0          0 eth0
192.168.206.0   *               255.255.255.0   U         0 0          0 eth0
192.168.206.2   192.168.206.3   255.255.255.255 UGH       0 0          0 eth0
```

Figure 46. IP routing table Client1

A rule as shown in this capture (Figure 46) should appear in Client1’s IP routing table. Once we have our IP routing table in Client1 properly configured, we can begin with the PING tests to check the WANem’s behaviour with the packet loss option.



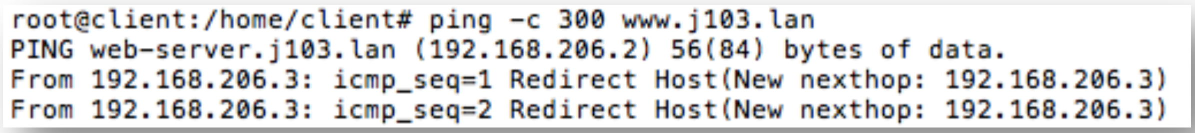
```
root@client:/home/client# ping -c 20 www.j103.lan
PING web-server.j103.lan (192.168.206.2) 56(84) bytes of data.
From 192.168.206.3: icmp_seq=1 Redirect Host(New nexthop: 192.168.206.3)
From 192.168.206.3: icmp_seq=3 Redirect Host(New nexthop: 192.168.206.3)
64 bytes from 192.168.206.3: icmp_seq=3 ttl=64 time=0.447 ms
64 bytes from 192.168.206.3: icmp_seq=6 ttl=64 time=0.341 ms
64 bytes from 192.168.206.3: icmp_seq=9 ttl=64 time=0.361 ms
64 bytes from 192.168.206.3: icmp_seq=11 ttl=64 time=0.442 ms
64 bytes from 192.168.206.3: icmp_seq=13 ttl=64 time=0.441 ms
64 bytes from 192.168.206.3: icmp_seq=17 ttl=64 time=0.426 ms
64 bytes from 192.168.206.3: icmp_seq=18 ttl=64 time=0.429 ms
64 bytes from 192.168.206.3: icmp_seq=19 ttl=64 time=0.382 ms
64 bytes from 192.168.206.3: icmp_seq=20 ttl=64 time=0.396 ms

--- web-server.j103.lan ping statistics ---
20 packets transmitted, 9 received, 55% packet loss, time 19079ms
rtt min/avg/max/mdev = 0.341/0.407/0.447/0.038 ms
```

Figure 47. Ping packet loss test

We have set up in WANem host a 50% packet loss, and as we can see here (Figure 47), PING tool indicates us that 55% of the packets were lost. If you set up a probability of 50%

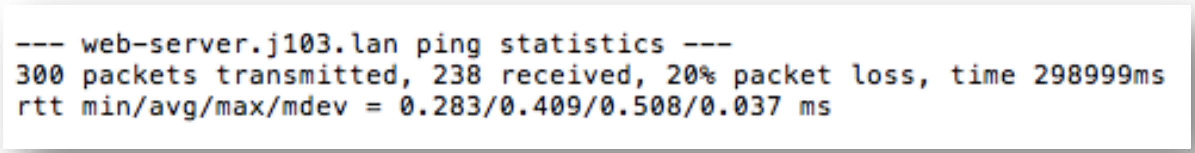
packet loss, around 50% of the packets will be lost. This is an average rate of packet loss, because PING cannot measure exactly how many packets have been lost.

If we set up a 20% packet loss in WANem host and we do a 300 packets PING, we could see how this average is more accurate.



```
root@client:/home/client# ping -c 300 www.j103.lan
PING web-server.j103.lan (192.168.206.2) 56(84) bytes of data.
From 192.168.206.3: icmp_seq=1 Redirect Host(New nexthop: 192.168.206.3)
From 192.168.206.3: icmp_seq=2 Redirect Host(New nexthop: 192.168.206.3)
```

Figure 48a. 300 packets PING (beginning)



```
--- web-server.j103.lan ping statistics ---
300 packets transmitted, 238 received, 20% packet loss, time 298999ms
rtt min/avg/max/mdev = 0.283/0.409/0.508/0.037 ms
```

Figure 48b. 300 packets PING (end)

This two captures (Figures 48a and 48b) show beginning and end of a 300 packets PING, and as we can see in the second line of the last one, 20% of packets were lost. Unlike the last test, here we have exactly the same packet loss as we have configured in WANem. So it means with a high number of packets sent with a PING, it approaches the configured packet loss rate. Although here we have exactly the same rate, it doesn't always happen always. We can also obtain a packet loss rate around the theoretical, but not the same one.

### 4.3.3. – TCP - Transmission Control Protocol

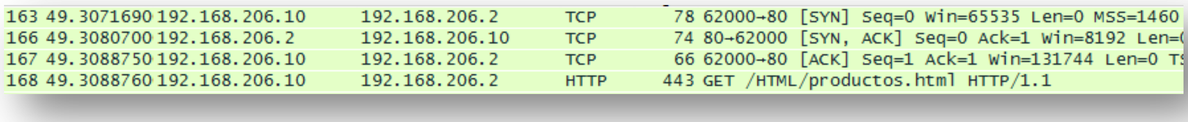
TCP means “*Transmission Control Protocol*” and it provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). Due to network congestion, traffic load balancing, or others unpredictable network behaviours, IP packets can be lost, duplicated, or out of order delivered. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps to minimize network congestion.

In this case we won’t use Traceroute or Ping tools as we used for IP tests. We will use Wireshark, a packet sniffer software and with it, we will be able to capture and analyze every frame on our network. It shows us far more information than Traceroute or Ping. We can filter by TCP protocol and follow the complete sequence of segments. We can also check the exact time where those segments have arrived, its sequence’s number, length, IP address source or destination and many more details.

To do some of these tests, we should have our IP routing table configured as we had for the IP tests (Connection Delays and Packet Loss). We will need to check if Client1 has a rule in its IP routing table to send the traffic to Web-Server through the WANem host.

At the beginning of each test, it will be specified if the traffic is going through the WANem host or not.

Now, we are going to test TCP behaviour on two different scenarios. First, we connect Client1 and Web-Server with a direct connection. So in this first test all the traffic is going directly from Client1 to Web-Server, without going through WANem.

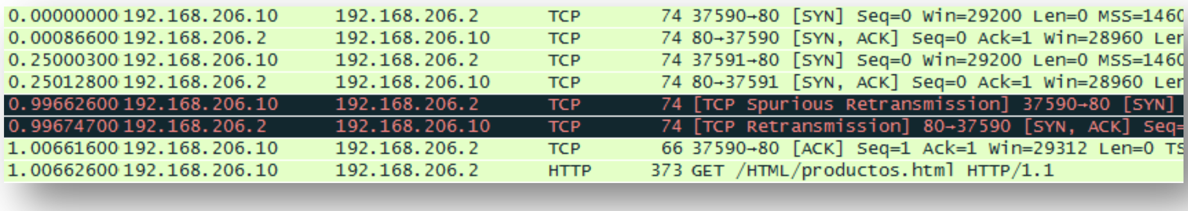


```
163 49.3071690 192.168.206.10     192.168.206.2      TCP      78 62000→80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
166 49.3080700 192.168.206.2      192.168.206.10     TCP      74 80→62000 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=(
167 49.3088750 192.168.206.10     192.168.206.2      TCP      66 62000→80 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TS
168 49.3088760 192.168.206.10     192.168.206.2      HTTP    443 GET /HTML/productos.html HTTP/1.1
```

Figure 49. TCP segments exchanged without delays (opening)

As we can see in this Wireshark capture (Figure 49), TCP uses a three-way handshake to establish the connection: SYN, SYN-ACK, ACK. There are no delays, so there are no retransmissions. This is a direct connection between the client and the web-server, so WANem has nothing to do here.

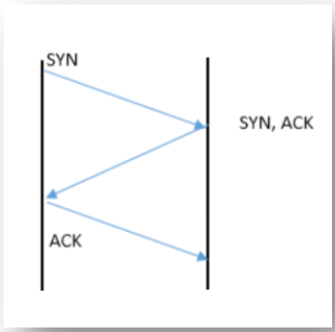
Now, we establish a connection between Client1 and Web-Server through the WANem host, which is going to add a 500 ms delay.



```
0.00000000 192.168.206.10     192.168.206.2      TCP      74 37590→80 [SYN] Seq=0 Win=29200 Len=0 MSS=146(
0.00086600 192.168.206.2      192.168.206.10     TCP      74 80→37590 [SYN, ACK] Seq=0 Ack=1 Win=28960 Ler
0.25000300 192.168.206.10     192.168.206.2      TCP      74 37591→80 [SYN] Seq=0 Win=29200 Len=0 MSS=146(
0.25012800 192.168.206.2      192.168.206.10     TCP      74 80→37591 [SYN, ACK] Seq=0 Ack=1 Win=28960 Ler
0.99662600 192.168.206.10     192.168.206.2      TCP      74 [TCP Spurious Retransmission] 37590→80 [SYN]
0.99674700 192.168.206.2      192.168.206.10     TCP      74 [TCP Retransmission] 80→37590 [SYN, ACK] Seq=
1.00661600 192.168.206.10     192.168.206.2      TCP      66 37590→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TS
1.00662600 192.168.206.10     192.168.206.2      HTTP    373 GET /HTML/productos.html HTTP/1.1
```

Figure 50. TCP segments exchanged with a 500ms delay (opening)

In this second capture (Figure 50), we can see how TCP tries to establish the connection but, due to 500 ms delay, the establishment segments are retransmitted. Now we are going to analyze this second situation.



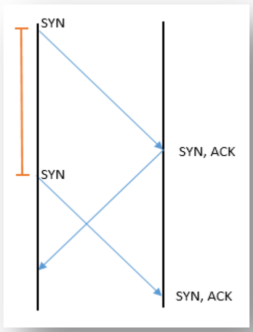
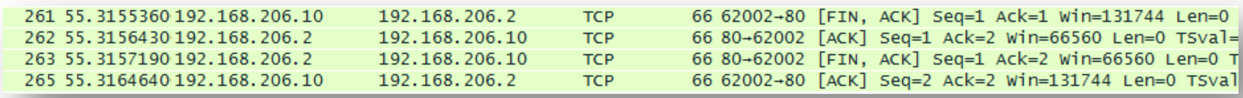


Figure 51a. Diagram without delay

Figure 51b. Diagram with delay

Here we have two diagrams. The first one (Figure 51a) represents the exchange of segments between client and server needed to establish a TCP connection when there are no delays. So here everything is fine and we don't have any retransmission. WANem is not working on this connection. On the second one (Figure 51b), due to WANem host delay, the timer – set when the first segment was sent – ends and the SYN segment is retransmitted.

If we observe how does TCP close a connection, we could find the same behaviour as before. On the direct connection there will be no problems, but on the other one, the delay introduced by WANem will cause several retransmissions. As we have seen before, TCP uses a four-way handshake to close the connection. In the next Wireshark capture (Figure 52), we could see TCP does it.

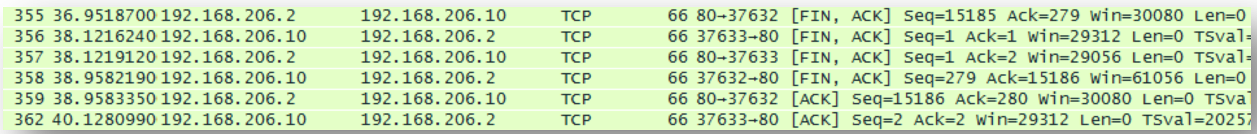


```
261 55.3155360 192.168.206.10    192.168.206.2     TCP        66 62002→80 [FIN, ACK] Seq=1 Ack=1 Win=131744 Len=0
262 55.3156430 192.168.206.2     192.168.206.10    TCP        66 80→62002 [ACK] Seq=1 Ack=2 Win=66560 Len=0 TSval=
263 55.3157190 192.168.206.2     192.168.206.10    TCP        66 80→62002 [FIN, ACK] Seq=1 Ack=2 Win=66560 Len=0 T
265 55.3164640 192.168.206.10    192.168.206.2     TCP        66 62002→80 [ACK] Seq=2 Ack=2 Win=131744 Len=0 TSval
```

Figure 52. TCP segments exchanged without delays (closing)

We can see the four-way handshake working properly. There are no retransmissions because there are no delays or packet loss in this communication, so everything goes fine.

Now we are going to see what happens when TCP wants to close a connection, which is going through the WANem host.



```
355 36.9518700 192.168.206.2     192.168.206.10    TCP       66 80→37632 [FIN, ACK] Seq=15185 Ack=279 Win=30080 Len=0
356 38.1216240 192.168.206.10    192.168.206.2     TCP       66 37633→80 [FIN, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=
357 38.1219120 192.168.206.2     192.168.206.10    TCP       66 80→37633 [FIN, ACK] Seq=1 Ack=2 Win=29056 Len=0 TSval=
358 38.9582190 192.168.206.10    192.168.206.2     TCP       66 37632→80 [FIN, ACK] Seq=279 Ack=15186 Win=61056 Len=0
359 38.9583350 192.168.206.2     192.168.206.10    TCP       66 80→37632 [ACK] Seq=15186 Ack=280 Win=30080 Len=0 TSval
362 40.1280990 192.168.206.10    192.168.206.2     TCP       66 37633→80 [ACK] Seq=2 Ack=2 Win=29312 Len=0 TSval=20257
```

Figure 53. TCP segments exchanged with a delay (closing)

In this capture (Figure 53) we can see how, due to WANem host delay, the [FIN,ACK] segments are sent twice. Each machine is retransmitting its own FIN segment because of the timer ends due to delay. Finally the TCP connection can be properly closed because each host receives the correct ACK segment.

### 4.3.4. – DNS - Domain Name Server

DNS, Domain Name Server, is an Internet service that translates domain names into IP addresses. As mentioned before, in our environment DNS server is not behind the WANem host, so our DNS queries go directly from the client to the DNS server (Figure 14).

Due to these problems with delayed DNS queries, we are not going to test this protocol when sending its traffic through WANem. We will just briefly see what is the path of these DNS queries in our environment and then analyze how it works by using a Wireshark capture.

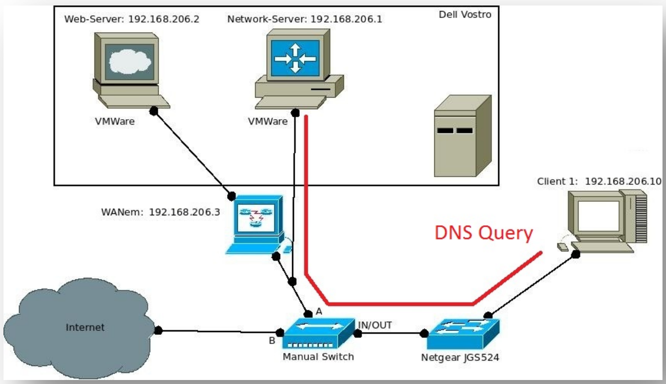
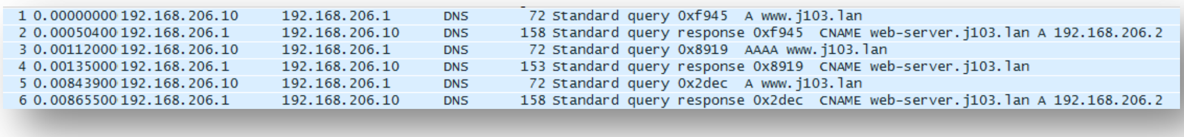


Figure 54. DNS query's path

In this capture (Figure 54) we observe the path of the DNS queries between the clients (Client1 in this case) and the DNS server. Keep in mind that in our environment, the Network-Server machine is working at the same time as DNS server and as DHCP server. So it's the only host, which never sends its traffic through the WANem host.

Now, with this Wireshark capture, we are going to check if the DNS server is working properly.



```
1 0.00000000 192.168.206.10    192.168.206.1     DNS     72 Standard query 0xf945  A www.j103.lan
2 0.00050400 192.168.206.1     192.168.206.10    DNS    158 Standard query response 0xf945  CNAME web-server.j103.lan A 192.168.206.2
3 0.00112000 192.168.206.10    192.168.206.1     DNS     72 Standard query 0x8919  AAAA www.j103.lan
4 0.00135000 192.168.206.1     192.168.206.10    DNS    153 Standard query response 0x8919  CNAME web-server.j103.lan
5 0.00843900 192.168.206.10    192.168.206.1     DNS     72 Standard query 0x2dec  A www.j103.lan
6 0.00865500 192.168.206.1     192.168.206.10    DNS    158 Standard query response 0x2dec  CNAME web-server.j103.lan A 192.168.206.2
```

Figure 55. DNS query – Wireshark capture

As we can see on this capture (Figure 55), the client requests to the DNS server the IP address of "www.j103.lan". DNS server answer contains information about the real name of the server and its IP address. If we observe the capture, we can see that "www.j103.lan" is the canonical name of "web-server.j103.lan" and it IP address is 192.168.206.2.

If we compare the DNS server's response (Figure 15) with the information given on the network's diagram (Figure 14), we can check that DNS server is working fine.

### 4.3.5. – HTTP - Hypertext Transfer Protocol

As we can read on the World Wide Web Consortium (W3C) website: "*The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.*"

HTTP is a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it.

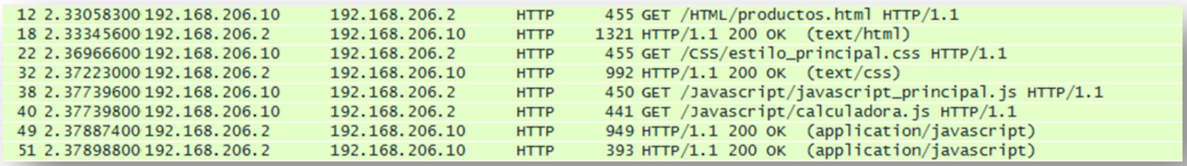
In this section we are going to explore some aspects of HTTP protocol such as:

- GET/response interaction
- HTTP message formats

To perform these tests with HTTP we will use again the Wireshark tool and for some of them, we need to send the traffic from Client1 to Web-Server through the WANem host. We should have our IP routing table configured as we had for the IP and TCP tests, to verify it we only have to type "`netstat -r`" in a Terminal. Client1 should have a rule in its IP routing table to send the traffic to Web-Server through the WANem host.

At the beginning of each test, it will be specified if the traffic is going through the WANem host or not.

In this first test WANem host is not involved. We have a direct connection between Client1 and Web-Server and we just only want to see the HTTP request method, the HTTP version used and which status code we can find.

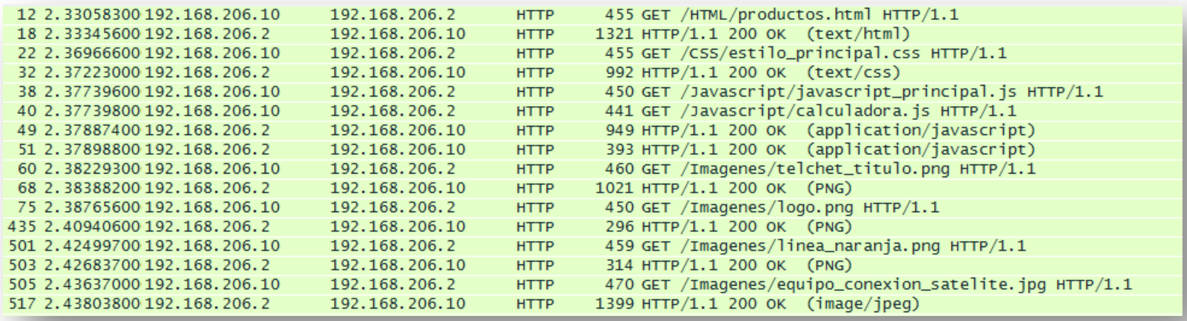


```
12 2.33058300 192.168.206.10     192.168.206.2      HTTP      455 GET /HTML/productos.html HTTP/1.1
18 2.33345600 192.168.206.2      192.168.206.10     HTTP     1321 HTTP/1.1 200 OK  (text/html)
22 2.36966600 192.168.206.10     192.168.206.2      HTTP      455 GET /CSS/estilo_principal.css HTTP/1.1
32 2.37223000 192.168.206.2      192.168.206.10     HTTP      992 HTTP/1.1 200 OK  (text/css)
38 2.37739600 192.168.206.10     192.168.206.2      HTTP      450 GET /Javascript/javascript_principal.js HTTP/1.1
40 2.37739800 192.168.206.10     192.168.206.2      HTTP      441 GET /Javascript/calculadora.js HTTP/1.1
49 2.37887400 192.168.206.2      192.168.206.10     HTTP      949 HTTP/1.1 200 OK  (application/javascript)
51 2.37898800 192.168.206.2      192.168.206.10     HTTP      393 HTTP/1.1 200 OK  (application/javascript)
```

Figure 56. HTTP GET/response interaction

As we can see on this capture (Figure 56), Client1 requests each element (html, images, css files…) needed to load the webpage. GET is the method used to obtain these elements. We can also see the HTTP version used is "1.1" and the status code received is "200 OK". In this case we have a success status code, so it means that the request was fulfilled.

Now we are going to test HTTP's behaviour on two different scenarios: with and without delays. Let's begin by loading a web page with a direct connection between the client and the web server.

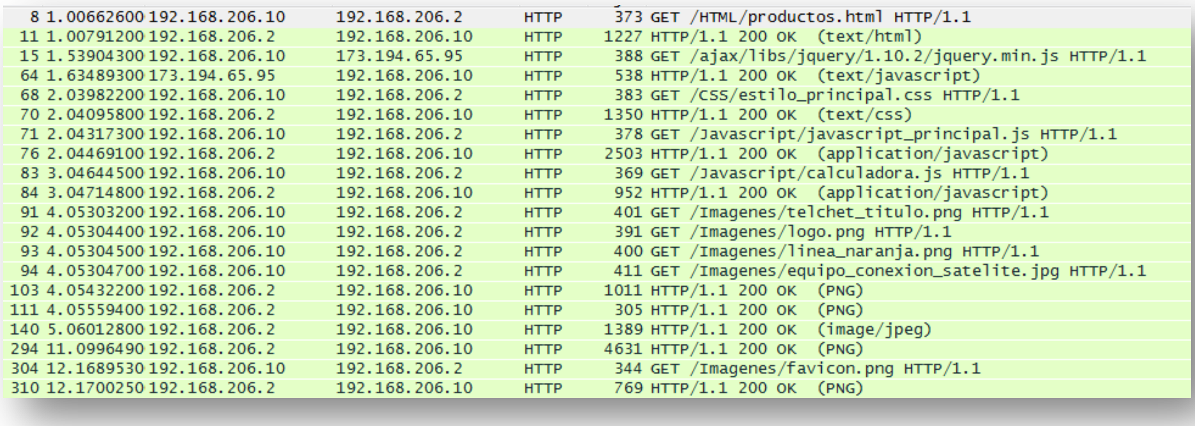


```
 12 2.33058300 192.168.206.10    192.168.206.2     HTTP    455 GET /HTML/productos.html HTTP/1.1
 18 2.33345600 192.168.206.2     192.168.206.10    HTTP   1321 HTTP/1.1 200 OK  (text/html)
 22 2.36966600 192.168.206.10    192.168.206.2     HTTP    455 GET /CSS/estilo_principal.css HTTP/1.1
 32 2.37223000 192.168.206.2     192.168.206.10    HTTP    992 HTTP/1.1 200 OK  (text/css)
 38 2.37739600 192.168.206.10    192.168.206.2     HTTP    450 GET /Javascript/javascript_principal.js HTTP/1.1
 40 2.37739800 192.168.206.10    192.168.206.2     HTTP    441 GET /Javascript/calculadora.js HTTP/1.1
 49 2.37887400 192.168.206.2     192.168.206.10    HTTP    949 HTTP/1.1 200 OK  (application/javascript)
 51 2.37898800 192.168.206.2     192.168.206.10    HTTP    393 HTTP/1.1 200 OK  (application/javascript)
 60 2.38229300 192.168.206.10    192.168.206.2     HTTP    460 GET /Imagenes/telchet_titulo.png HTTP/1.1
 68 2.38388200 192.168.206.2     192.168.206.10    HTTP   1021 HTTP/1.1 200 OK  (PNG)
 75 2.38765600 192.168.206.10    192.168.206.2     HTTP    450 GET /Imagenes/logo.png HTTP/1.1
435 2.40940600 192.168.206.2     192.168.206.10    HTTP    296 HTTP/1.1 200 OK  (PNG)
501 2.42499700 192.168.206.10    192.168.206.2     HTTP    459 GET /Imagenes/linea_naranja.png HTTP/1.1
503 2.42683700 192.168.206.2     192.168.206.10    HTTP    314 HTTP/1.1 200 OK  (PNG)
505 2.43637000 192.168.206.10    192.168.206.2     HTTP    470 GET /Imagenes/equipo_conexion_satelite.jpg HTTP/1.1
517 2.43803800 192.168.206.2     192.168.206.10    HTTP   1399 HTTP/1.1 200 OK  (image/jpeg)
```

Figure 57. HTTP loading webpage – No delays

In this first test (Figure 57), as we have direct connection between our computer and the web server, everything is going to be fine. The WANem host is not working here, so there are no delays on our communication. We can observe the whole web page is loaded in less than 0.2 seconds.

Now we are going to load the same web page again. This time we have modified the path and the connection between the client and the web-server is going through the WANem host. This one has been configured to add a 500ms delay on every communication going through it.
Remind that before loading the web page, we have to clean the web browser's cache.



```
  8 1.00662600 192.168.206.10    192.168.206.2     HTTP    373 GET /HTML/productos.html HTTP/1.1
 11 1.00791200 192.168.206.2     192.168.206.10    HTTP   1227 HTTP/1.1 200 OK  (text/html)
 15 1.53904300 192.168.206.10    173.194.65.95     HTTP    388 GET /ajax/libs/jquery/1.10.2/jquery.min.js HTTP/1.1
 64 1.63489300 173.194.65.95     192.168.206.10    HTTP    538 HTTP/1.1 200 OK  (text/javascript)
 68 2.03982200 192.168.206.10    192.168.206.2     HTTP    383 GET /CSS/estilo_principal.css HTTP/1.1
 70 2.04095800 192.168.206.2     192.168.206.10    HTTP   1350 HTTP/1.1 200 OK  (text/css)
 71 2.04317300 192.168.206.10    192.168.206.2     HTTP    378 GET /Javascript/javascript_principal.js HTTP/1.1
 76 2.04469100 192.168.206.2     192.168.206.10    HTTP   2503 HTTP/1.1 200 OK  (application/javascript)
 83 3.04644500 192.168.206.10    192.168.206.2     HTTP    369 GET /Javascript/calculadora.js HTTP/1.1
 84 3.04714800 192.168.206.2     192.168.206.10    HTTP    952 HTTP/1.1 200 OK  (application/javascript)
 91 4.05303200 192.168.206.10    192.168.206.2     HTTP    401 GET /Imagenes/telchet_titulo.png HTTP/1.1
 92 4.05304400 192.168.206.10    192.168.206.2     HTTP    391 GET /Imagenes/logo.png HTTP/1.1
 93 4.05304500 192.168.206.10    192.168.206.2     HTTP    400 GET /Imagenes/linea_naranja.png HTTP/1.1
 94 4.05304700 192.168.206.10    192.168.206.2     HTTP    411 GET /Imagenes/equipo_conexion_satelite.jpg HTTP/1.1
103 4.05432200 192.168.206.2     192.168.206.10    HTTP   1011 HTTP/1.1 200 OK  (PNG)
111 4.05559400 192.168.206.2     192.168.206.10    HTTP    305 HTTP/1.1 200 OK  (PNG)
140 5.06012800 192.168.206.2     192.168.206.10    HTTP   1389 HTTP/1.1 200 OK  (image/jpeg)
294 11.0996490 192.168.206.2     192.168.206.10    HTTP   4631 HTTP/1.1 200 OK  (PNG)
304 12.1689530 192.168.206.10    192.168.206.2     HTTP    344 GET /Imagenes/favicon.png HTTP/1.1
310 12.1700250 192.168.206.2     192.168.206.10    HTTP    769 HTTP/1.1 200 OK  (PNG)
```

Figure 58. HTTP loading webpage – Delays

As we can observe on this second capture (Figure 58), from the time the first element was requested until the last one was delivered, there is a gap of more than 11 seconds. It means WANem host is doing its job well and it delays all the traffic going through it. These objects are finally delivered because there is only delay in the communication and WANem is not losing packets.

# 5. – Conclusions and Future Lines

## 5.1. – Conclusions

Having arrived at this point, we affirm that the main goal has been completely achieved. We have built a complete environment where students could learn about networking protocols through practical training. They can also emulate a real Wide Area Network and modify the connection delay or the average rate of packets lost, among many other characteristics.

In the *Wireshark Labs* section, in addition to network tests, we have analyzed the results of these tests, explaining the reason of each result. We have some results, which are not exact, because they are based on a network traffic estimation (e.g. Ping packet lost).

It is worth noting that in the *Laboratory Installation* section we have found some difficulties, most of them due to Network-Server and Web-Server being virtual machines. It has involved some complications when trying to get the laboratory operating. Finally these difficulties have been successfully solved and now everything is working properly and ready to be used to teach networking.

## 5.2. – Future Lines

The main software tool used to emulate our real environment has been WANem (Wide Area Network emulator) and, as we have seen in the section where we explained this tool, WANem software has many different options. We have used some of them (like connection delays or data packets loss) but there are many more.

This project could be extended simply performing new network tests with these unused WANem features. Some of them may be really interesting to prove in this environment, such as jitter, randomly disconnections or packet corruption. Another option is to join two or three different WANem characteristics working at the same time and test how our environment responds.

If some new tests are done, the results could be also analyzed with the current analysing network tools, like *Traceroute*, *Ping* or *Wireshark.* They are powerful enough to work in an upcoming development.

# Bibliography

1. [Figure 1] Cosby, Scott. “BACnet Architecture”. *OSI Model.* http://www.chipkin.com/bacnet-architecture/ (retrieved January 23, 2015)

2. [Figure 2] “TCP/IP Model Figure 2”. Kevin R. Fall, W. Richard Stevens. (2011) *TCP/IP Illustrated, Volume 1*, Addison-Wesley

3. [Figure 3] “Multicast”. *IPv4 addresses.* http://www.net130.com/CMS/Pub/network/network_protocal/2005_08_21_36977_3.htm (retrieved January 25, 2015)

4. [Figure 4] “Analyzing Classful Ipv4 Networks”. *Classful network.* http://resources.intenseschool.com/ccna-prep-analyzing-classful-ipv4-networks/ (retrieved January 20, 2015)

5. [Figure 5] “IPv4 Subnetting”. IPv4*Subnets* http://sunix.hk/study/cisco/ipv4-subnetting.html (retrieved January 24, 2015)

6. [Figure 6 – 7]. “Address Resolution Protocol”. *ARP request and reply.* http://networklessons.com/cisco/arp-address-resolution-protocol-explained/ (retrieved January 24, 2015)

7. [Figure 8]. “Route My World!”. *IPv4 Header.* http://routemyworld.com/2009/02/01/bsci-ip-version-6/ (retrieved January 24, 2015)

8. “Wireshark Wiki” *Dynamic Host Configuration Protocol (DHCP)*, Year 2010. URL: http://wiki.wireshark.org/DHCP (retrieved January 24, 2015)

9. [RFC 2131] “Network Working Group” *Dynamic Host Configuration Protocol (DHCP)*, March 1997. URL: https://www.ietf.org/rfc/rfc2131.txt (retrieved January 29, 2015)

10. [Figure 10] “DHCP Snooping and IP Source Guard”. *DHCP.* https://www.howtonetwork.net/members/login.cfm?hpage=DHCP_Snooping_and_IP_Source_Guard.cfm (retrieved January 29, 2015)

11. [Figure 11] “An Intro to DNS”. *Domain Name System.* http://sophiedogg.com/intro-to-dns/ (retrieved January 29, 2015)

12. [Figure 12] “Microsoft TechNet” Domain Name Service. https://technet.microsoft.com/en-us/library/bb962025.aspx (retrieved January 29, 2015)

13. [Figure 13] “DNS Header”. *OpCode.* http://www.inacon.de/ph/data/DNS/Header_fields/Header_section_format/DNS-Header-Field-Flag-OPCODE_OS_RFC-1035.htm (retrieved January 31, 2015)

14. [Figure 14] “The TCP/IP Guide”. *TCP Segment Format.* http://www.tcpipguide.com/free/t_TCPMessageSegmentFormat-3.htm (retrieved January 31, 2015)

15. [Figure 15] “Connection oriented communication (TCP/IP)”. *TCP session establishment and termination*. http://wiki.mikrotik.com/wiki/Manual:Connection_oriented_communication_(TCP/IP) (retrieved January 31, 2015)

16. [Figure 16] “HTTP persistent connection”. *HTTP keep-alive*. https://en.wiki2.org/wiki/HTTP_persistent_connection (retrieved January 31, 2015)

17. [Figure 18] “Application Layer”. *HTTP Message Format*. http://www.studycampus.com/PgD/cnm/lesson2.htm (retrieved January 31, 2015)

18. [RFC 2616] “Hypertext Transfer Protocol -- HTTP/1.1”. *Status Codes Definitions*. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html (retrieved January 31, 2015)

19. “Iptables Firewall Rules”. *Save IpTables Permanently*. http://www.thomas-krenn.com/en/wiki/Saving_Iptables_Firewall_Rules_Permanently (retrieved January 31, 2015)

20. “Ubuntu Official Website”. *Ubuntu and Debian.* http://www.ubuntu.com/about/about-ubuntu/ubuntu-and-debian (retrieved January 31, 2015)

21. “Internet System Consortium”. *ISC DHCP*. https://www.isc.org/downloads/dhcp/ (retrieved January 31, 2015)

22. “Internet System Consortium”. *BIND and DNS*. https://www.isc.org/downloads/bind/ (retrieved January 31, 2015)

23. “Apache Official Website”. *HTTP Server Project*. http://httpd.apache.org/ (retrieved January 31, 2015)

24. James F. Kurose, Keith W. Ross. (2013) *Computer Networking (A Top Down Approach)*, Pearson

25. Kevin R. Fall, W. Richard Stevens. (2011) *TCP/IP Illustrated, Volume 1*, Addison-Wesley

26. Nemeth E., Snyder G., Hein Trent R., Whaley B., (2010) *UNIX® AND LINUX® SYSTEM ADMINISTRATION HANDBOOK*, Prentice Hall

27. Maxwell S., (2002) *UNIX System Administration: A Beginner’s Guide*, McGraw-Hill/Osborne

# Appendix A – Questions & Tasks

## Laboratory Installation

### The Environment

There is more than one way to implement the environment, however we need to select the one which closely fits our network. We are going to study two options, first a simple one and then another one more complicated but also more useful. **Try to design several possibilities and choose the best one for this network.**

### Setting up Network-Server as a Router

One layout needs to configure Network-Server as a router. It will distinguish between internal and outgoing traffic and it will send this second one outside the laboratory. **Try to discover if some new hardware is needed and then set up the router to forward outgoing traffic properly.**

### Wide Area Network emulator (WANem)

This environment requires a software tool, which allows us to modify the network conditions (like delays, jitter, packet loss, packet corruption…), and this tool will be WANem. **Try to install this Linux-based distribution on a host and introduce it into the network. The WANem host should be reachable for everyone in the LAN, even for new clients.**

### Servers

#### DHCP – Dynamic Host Configuration Protocol

To implement a new network, one of the first tasks should be how to allocate and manage the IP addresses. To tackle this, we have to choose which host will handle this IP addresses and what service we need to do it. We have to install a DHCP server and configure it to work in our network. Keep in mind that DHCP (Dynamic Host Configuration Protocol) is the protocol which takes care of IP addresses management in every network.

The chosen server for IP addresses management in our Local Area Network is “ISC DHCP”. **Try to install DHCP Server into Network-Server host and configure it to allocate and manage IP addresses in the whole Local Area Network.**

## DNS – Domain Name Server

The next step, after setting up the DHCP server, is to install a DNS server. This service will be responsible for associating IP addresses and hosts' names. This will make easier for us to identify the hosts for tasks like network maintenance.

We have also chosen the host Network-Server to be our LAN's Domain Name Server, so we need a software tool which allows us to implement this server in our network. We have chosen "bind9" to do it. **Try to install the DNS server and configure it to associate IP addresses and hosts' names in the Local Area Network.**

## Web Server – Apache, PHP and MySQL

The web server will be the main target in all of our tests. In the *Wireshark Part* we will use some tools, such as *traceroute* or *ping* to test the WANem server's behaviour with traffic going from clients towards the web server. We will use a complete LAMP server (Linux, Apache, MySQL and PHP). **Try to install these servers and configure them properly to host a webpage.**

## IP – Internet Protocol: *IP Routing Table*

In the *Wireshark labs* section we will need, for the majority of the tests, traffic going from client to server and from server to client through the WANem host. These rules are stored in the IP routing table and each machine always looks into this table before sending any traffic. Therefore, it is essential to have this tables properly configured if we want to be sure where we are sending the traffic through. **Try to configure the IP routing table on each machine you want to send traffic through the WANem host.**

# Wireshark Labs

## WANem Interface

Before beginning with Wireshark tests we need to know, how the WANem interface is and what options do we have. **Connect to the WANem server and try to configure the different modes (basic and advanced). Then try to test the applied configuration doing a Ping from a client to the Web-Server (sending the traffic through the WANem host).**

## Traceroute – IP Connection Delays

Use *Traceroute* diagnostic tool to check how many hops we have between the web server and our host. First, check if the IP routing tables in Client1 and in the Web-Server have the proper rules configured. We are going to test *Traceroute* in two different scenarios: first set up IP routing table to send traffic through the WANem host only in the client. Then do it in the Web-Server too. **Try to discover why the second packet duplicates its delay in the second scenario.**

## Ping – IP Packet Loss

Set up a loss percentage in WANem host and then use Ping tool to estimate the average rate of packets lost. Is it accurate? Try to test the same scenario with many more PING packets (e.g. 300 packets)

## Wireshark Tests

To perform the following tests we will establish a connection between the client and a webpage hosted in the local web-server. Remind that before loading the web page for a new test, we have to clean the web browser’s cache.

## TCP - Transmission Control Protocol

Now, we are going to test TCP behaviour on two different scenarios. First, we will connect Client and Web-Server with a direct connection. Then, we establish a connection between Client and Web-Server through the WANem host, which is going to add a 500 ms delay. **What happens with the opening three-way handshake and with the closing four-way handshake?** Use Wireshark to show the TCP exchange and explain the differences between in both cases. Keep in mind that we can filter by TCP protocol and follow the complete sequence of segments.

## HTTP - Hypertext Transfer Protocol

Use Wireshark to filter the traffic and show only HTTP requests and responses. **What HTTP version are we using? What are the status codes received from the server? Has the entire webpage loaded without problems? How long does it take to load the whole webpage?**

Now we are going to load the same web page again. This time we need to modify the path and the connection between the client and the web-server will go through the WANem host, which has been configured to add a 500ms delay on every communication going through it. **Now, how long does it take to load the whole webpage? Could you find any data retransmissions?**