Lecture 1: Introduction to Communication Protocols and "Non-Intrusive" Testing

Passive Testing Techniques for Communication Protocols

Dr. Jorge López, PhD.
jorgelopezcoronado[at]gmail.com

National Research Tomsk State University

February 9, 2016

OUTLINE

COURSE INFORMATION

Sessions:

COURSE INFORMATION

Sessions:

- 3 Weeks

COURSE INFORMATION

Sessions:

- 3 Weeks
- Tuesdays Lectures (1.5h) @ 12:25(?)
- Thursdays Lectures (1.5h) @ 12:25(?)
- Saturdays Laboratories (3h) @ 10:30(?) — yeeeiiii :)

COURSE INFORMATION

Sessions:

- 3 Weeks
- Tuesdays Lectures (1.5h) @ 12:25(?)
- Thursdays Lectures (1.5h) @ 12:25(?)
- Saturdays Laboratories (3h) @ 10:30(?) — yeeeiiii :)

Evaluation

Course Information

Sessions:

- 3 Weeks
- Tuesdays Lectures (1.5h) @ 12:25(?)
- Thursdays Lectures (1.5h) @ 12:25(?)
- Saturdays Laboratories (3h) @ 10:30(?) — yeeeiiii :)

Evaluation

- Saturdays of **death!**

COURSE INFORMATION

Sessions:

- 3 Weeks
- Tuesdays Lectures (1.5h) @ 12:25(?)
- Thursdays Lectures (1.5h) @ 12:25(?)
- Saturdays Laboratories (3h) @ 10:30(?) — yeeeiiii :)

Evaluation

- Saturdays of tests!

COURSE INFORMATION

Sessions:

- 3 Weeks
- Tuesdays Lectures (1.5h) @ 12:25(?)
- Thursdays Lectures (1.5h) @ 12:25(?)
- Saturdays Laboratories (3h) @ 10:30(?) — yeeeiiii :)

Evaluation

- Saturdays of tests!
  - Short(˜15mins) quizzes of the week's lectures

COURSE INFORMATION

Sessions:

- 3 Weeks
- Tuesdays Lectures (1.5h) @ 12:25(?)
- Thursdays Lectures (1.5h) @ 12:25(?)
- Saturdays Laboratories (3h) @ 10:30(?) — yeeeiiii :)

Evaluation

- Saturdays of tests!
  - Short(~15mins) quizzes of the week's lectures
- Laboratories are due before the beginning of the subsequent laboratory (@ 10:29:59 GMT + 6)

COURSE INFORMATION (CONT.)

Requisites

COURSE INFORMATION (CONT.)

Requisites

- General knowledge of state models (duh!)

COURSE INFORMATION (CONT.)

Requisites

- General knowledge of state models (duh!)
- General knowledge of C

COURSE INFORMATION (CONT.)

Requisites

- General knowledge of state models (duh!)
- General knowledge of C
- Do not be shy about your English!

COURSE INFORMATION (CONT.)

Requisites

- General knowledge of state models (duh!)
- General knowledge of C
- Do not be shy about your English!

Topics

COURSE INFORMATION (CONT.)

Requisites

- General knowledge of state models (duh!)
- General knowledge of C
- Do not be shy about your English!

Topics

- Communication protocols and non-intrusive testing methods

COURSE INFORMATION (CONT.)

Requisites

- General knowledge of state models (duh!)
- General knowledge of C
- Do not be shy about your English!

Topics

- Communication protocols and non-intrusive testing methods
- Static code analysis (for protocol implementations)

Course Information (cont.)

Requisites

- General knowledge of state models (duh!)
- General knowledge of C
- Do not be shy about your English!

Topics

- Communication protocols and non-intrusive testing methods
- Static code analysis (for protocol implementations)
- Passive network trace analysis

COURSE INFORMATION — LAB ENVIRONMENT

GNU/Linux environment!

You choose the distro, but, make sure your code compiles with a 'Makefile'

Course Information — Lab environment

GNU/Linux environment!

You choose the distro, but, make sure your code compiles with a 'Makefile'

My personal recommendation:

COURSE INFORMATION — LAB ENVIRONMENT

GNU/Linux environment!

You choose the distro, but, make sure your code compiles with a 'Makefile'

My personal recommendation:

- Download a pre-configured virtual machine image (OVFv2) created for our course , a CentOS 7 (and KDE for a cute look) **available** here

COURSE INFORMATION — LAB ENVIRONMENT

GNU/Linux environment!

You choose the distro, but, make sure your code compiles with a 'Makefile'

My personal recommendation:

- Download a pre-configured virtual machine image (OVFv2) created for our course , a CentOS 7 (and KDE for a cute look) **available** here
- Use any hypervisor engine — I use Oracle's VirtualBox, it's good, it's free

COURSE INFORMATION — LAB ENVIRONMENT

GNU/Linux environment!

You choose the distro, but, make sure your code compiles with a 'Makefile'

My personal recommendation:

- Download a pre-configured virtual machine image (OVFv2) created for our course , a CentOS 7 (and KDE for a cute look) **available** here
- Use any hypervisor engine — I use Oracle's VirtualBox, it's good, it's free
- default user: Eva-01(sudoer) password: **PTTCCLabVMP4ss!** //change it!

COURSE INFORMATION — LAB ENVIRONMENT

GNU/Linux environment!

You choose the distro, but, make sure your code compiles with a 'Makefile'

My personal recommendation:

- Download a pre-configured virtual machine image (OVFv2) created for our course , a CentOS 7 (and KDE for a cute look) **available** here
- Use any hypervisor engine — I use Oracle's VirtualBox, it's good, it's free
- default user: Eva-01(sudoer) password: **PTTCCLabVMP4ss!** //change it!

Others

Make a PDF. Use LaTeX, MS Word, OpenOffice writer, ...

WHY TO STUDY THIS?

- The widespread adoption of applications using networks (i.e., communication protocols)

WHY TO STUDY THIS?

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)

Why to study this?

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)
  - Critical Operations + Sensitive Information

WhatsApp

WHY TO STUDY THIS?

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)
  - Critical Operations + Sensitive Information
  - Tell me the last application you used which does not interact over the network?

WhatsApp

WHY TO STUDY THIS?

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)
  - Critical Operations + Sensitive Information
  - Tell me the last application you used which does not interact over the network?

WhatsApp

Important

Testing such applications / systems is crucial!

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing communication systems

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing communication systems

- **Active testing** feels like an intuitive method for testing software and systems

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not interfere the system?

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not interfere the system?

- The data on the system is susceptible to change with the tests

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not interfere the system?

- The data on the system is susceptible to change with the tests
- Certain functionality is not available if "real" data is not entered

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not interfere the system?

- The data on the system is susceptible to change with the tests
- Certain functionality is not available if "real" data is not entered
- Even if a system can be interrupted, it can be wanted to perform tests against the "real" service / application / data

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

Methods for Non-Intrusive Testing

We focus on two of them…

- Static (code) analysis

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A “white-box’ approach. we assume we do not disrupt the implementation, we have access to the source code of it

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them. . .

- Static (code) analysis
  - A “white-box’ approach. we assume we do not disrupt the implementation, we have access to the source code of it
  - From the code, we test the implementation

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A "white-box' approach. we assume we do not disrupt the implementation, we have access to the source code of it
  - From the code, we test the implementation
- Passive network trace analysis

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A “white-box’ approach. we assume we do not disrupt the implementation, we have access to the source code of it
  - From the code, we test the implementation
- Passive network trace analysis
  - A “black-box’ approach. we assume we do not disrupt the implementation, we have access to the “messages” being exchanged by the implementation

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A "white-box' approach. we assume we do not disrupt the implementation, we have access to the source code of it
  - From the code, we test the implementation
- Passive network trace analysis
  - A "black-box' approach. we assume we do not disrupt the implementation, we have access to the "messages" being exchanged by the implementation
  - The "origin" where the messages are taken is called a Point of Observation (P.O.)

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
    - A "white-box' approach. we assume we do not disrupt the implementation, we have access to the source code of it
    - From the code, we test the implementation
- Passive network trace analysis
    - A "black-box' approach. we assume we do not disrupt the implementation, we have access to the "messages" being exchanged by the implementation
    - The "origin" where the messages are taken is called a Point of Observation (P.O.)

Let's take a look a quick look at each of them

STATIC ANALYSIS

```
/*Test equal distribution of random number generation algorithm*/
#include <stdio.h>
#include <stdlib.h>
#define NUM 30
#define MEM_SIZE 512*1024 //512MB
int main()
{
        short i = 0 , j;
        long acc = 0;
        if(!numbers)
        {
                printf(``Can't allocate memory\n'');
                exit(-1);
        }
        while (1)
        {
                numbers[i] = rand() % NUM ;  //random numbers from 0 - NUM
                acc = 0;
                for (j = 0; j < i; j++)
                        acc += numbers[j];
                printf(``New average: %ld\n'',  acc/++i); //should converge to NUM/2
        }
}
```

Do you see any problems with the code?

STATIC ANALYSIS (CONT.)

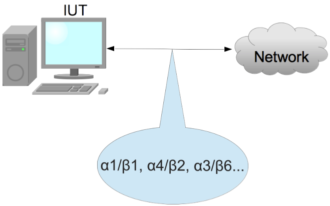
```
/*Test equal distribution of random number generation algorithm*/
#include <stdio.h>
#include <stdlib.h>
#define NUM 30
#define MEM_SIZE 512*1024 //512MB
int main()
{
        short i = 0 , j;
        long acc = 0;
        if(!numbers)
        {
                printf(``Can't allocate memory\n'');
                exit(-1);
        }
        while (1)
        {
                numbers[i] = rand() % NUM ;  //random numbers from 0 - NUM
                acc = 0;
                for (j = 0; j < i; j++)
                        acc += numbers[j];
                printf(``New average: %ld\n'',  acc/++i); //should converge to NUM/2
        }
}
```

Hard to see, hard to detect (iteration # 32,768)

Passive Network Trace Analysis

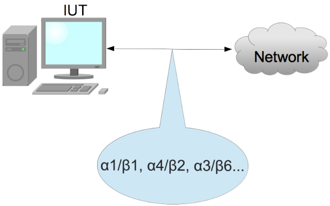
PASSIVE NETWORK TRACE ANALYSIS

Interaction



PASSIVE NETWORK TRACE ANALYSIS

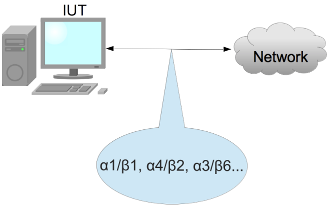
Interaction



Interpretation

PASSIVE NETWORK TRACE ANALYSIS

Interaction

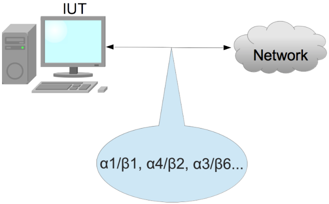


Interpretation

- The IUT interacts over the network.

PASSIVE NETWORK TRACE ANALYSIS

Interaction

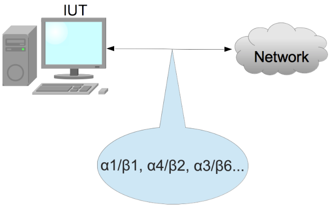


Interpretation

- The IUT interacts over the network.
- IUT inputs are 'αs', IUT outputs are 'βs'.

PASSIVE NETWORK TRACE ANALYSIS

Interaction

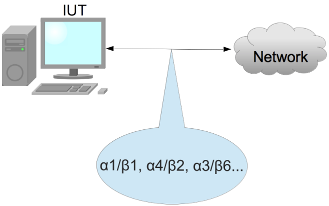


Interpretation

- The IUT interacts over the network.
- IUT inputs are 'αs', IUT outputs are 'βs'.
- αs are not produced by a tester!

PASSIVE NETWORK TRACE ANALYSIS

Interaction



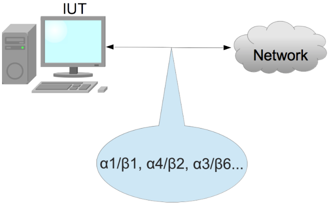
Interpretation

- The IUT interacts over the network.
- IUT inputs are 'αs', IUT outputs are 'βs'.
- αs are not produced by a tester!

How do we test this then?

PASSIVE NETWORK TRACE ANALYSIS (CONT.)

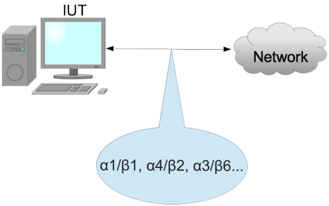
Interaction



Invariants — very popular

PASSIVE NETWORK TRACE ANALYSIS (CONT.)

Interaction

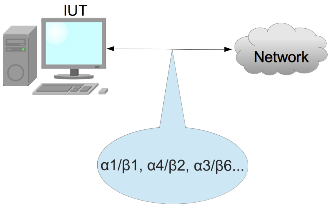


Invariants — very popular

- Briefly, try to guarantee that some *test purposes* hold over the network traces

PASSIVE NETWORK TRACE ANALYSIS (CONT.)

Interaction

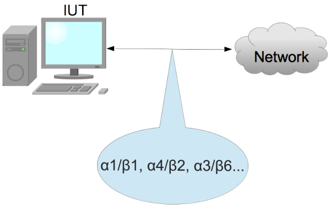


Invariants — very popular

- Briefly, try to guarantee that some *test purposes* hold over the network traces
- E.g., It is not allowed to observe $\beta 6$ before an occurrence of $\alpha 4$ (this holds for our presented trace)

PASSIVE NETWORK TRACE ANALYSIS (CONT.)

Interaction



Invariants — very popular

- Briefly, try to guarantee that some *test purposes* hold over the network traces
- E.g., It is not allowed to observe $\beta 6$ before an occurrence of $\alpha 4$ (this holds for our presented trace)
- **Many** languages proposed to express invariants

LET'S START GOING DEEPER

Preview is over...

LET'S START GOING DEEPER

Preview is over...

- I hope you are looking forward to this as much as you're looking forward for a fresh beer on a Friday night!

LET'S START GOING DEEPER

Preview is over...

- To sum up:

LET'S START GOING DEEPER

Preview is over...

- To sum up:
  - We want to test telecommunication systems without perturbing them

LET'S START GOING DEEPER

Preview is over...

- To sum up:
  - We want to test telecommunication systems without perturbing them
  - We propose to use two strategies: static (code) analysis & passive network trace analysis

LET'S START GOING DEEPER

Preview is over...

- To sum up:
  - We want to test telecommunication systems without perturbing them
  - We propose to use two strategies: static (code) analysis & passive network trace analysis

To test *<<something>>* you should understand *<<something>>*...

LET'S START GOING DEEPER

Preview is over...

- To sum up:
  - We want to test telecommunication systems without perturbing them
  - We propose to use two strategies: static (code) analysis & passive network trace analysis

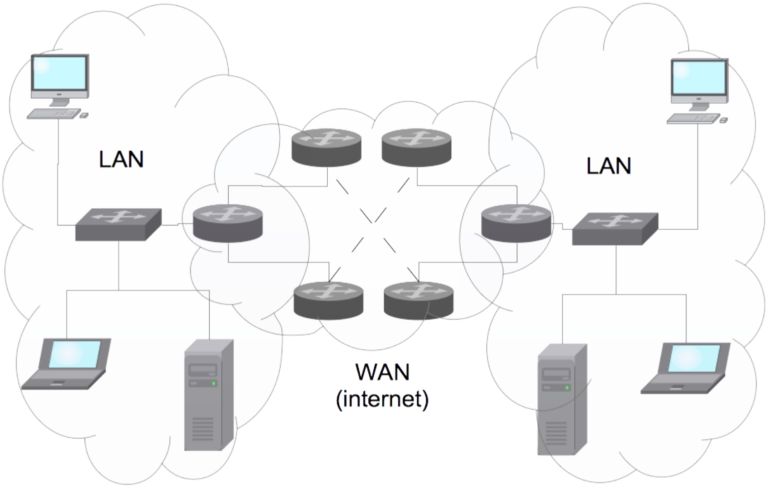
To test *<<something>>* you should understand *<<something>>*...

- Let's understand network protocols and how to easily interact with them

Communication Protocols
(for data/computer networks)

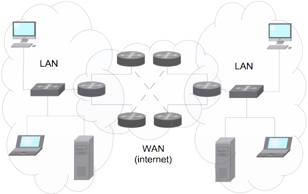
COMPUTER NETWORKS

Simplified Network Architecture



COMPUTER NETWORKS — TYPICAL NOTATIONS

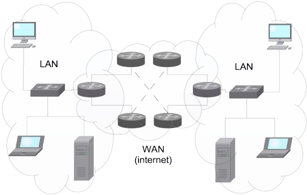
Simplified Network Architecture



- LAN = Local Area Network. WAN = Wide Area Network

COMPUTER NETWORKS — TYPICAL NOTATIONS

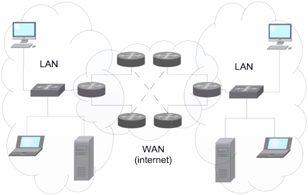
Simplified Network Architecture



- LAN = Local Area Network. WAN = Wide Area Network
- Router (intra-network communication): 

COMPUTER NETWORKS — TYPICAL NOTATIONS

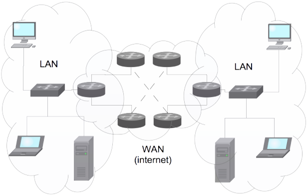
Simplified Network Architecture

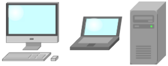


- LAN = Local Area Network. WAN = Wide Area Network
- Router (intra-network communication): 
- Switch (inter-network communication): 

COMPUTER NETWORKS — TYPICAL NOTATIONS

Simplified Network Architecture



- LAN = Local Area Network. WAN = Wide Area Network
- Router (intra-network communication): 
- Switch (inter-network communication): 
- Hosts (computers, data sources): 

COMPUTER NETWORKS — COMMUNICATION

Let's understand how <<*data*>> can reach one host to the other

Computer Networks — Communication

Let's understand how <<*data*>> can reach one host to the other

- It all starts by the local host sending <<*data*>> to the "local" LAN

COMPUTER NETWORKS — COMMUNICATION

Let's understand how <<*data*>> can reach one host to the other

- It all starts by the local host sending <<*data*>> to the "local" LAN
- From the "local" LAN the <<*data*>> is sent to the WAN

COMPUTER NETWORKS — COMMUNICATION

Let's understand how <<*data*>> can reach one host to the other

- It all starts by the local host sending <<*data*>> to the "local" LAN
- From the "local" LAN the <<*data*>> is sent to the WAN
- From the WAN the data gets forwarded iteratively until it reaches the "remote" LAN

Computer Networks — Communication

Let's understand how <<*data*>> can reach one host to the other

- It all starts by the local host sending <<*data*>> to the "local" LAN
- From the "local" LAN the <<*data*>> is sent to the WAN
- From the WAN the data gets forwarded iteratively until it reaches the "remote" LAN
- From the "remote" LAN the <<*data*>> is transfered to the remote host

COMPUTER NETWORKS — COMMUNICATION

Let's understand how <<*data*>> can reach one host to the other

- It all starts by the local host sending <<*data*>> to the "local" LAN
- From the "local" LAN the <<*data*>> is sent to the WAN
- From the WAN the data gets forwarded iteratively until it reaches the "remote" LAN
- From the "remote" LAN the <<*data*>> is transfered to the remote host

There are two layers in the communication process

COMPUTER NETWORKS — COMMUNICATION

Let's understand how <<*data*>> can reach one host to the other

- It all starts by the local host sending <<*data*>> to the "local" LAN
- From the "local" LAN the <<*data*>> is sent to the WAN
- From the WAN the data gets forwarded iteratively until it reaches the "remote" LAN
- From the "remote" LAN the <<*data*>> is transfered to the remote host

There are two layers in the communication process

- For the communication between hosts and LAN

COMPUTER NETWORKS — COMMUNICATION

Let's understand how <<*data*>> can reach one host to the other

- It all starts by the local host sending <<*data*>> to the "local" LAN
- From the "local" LAN the <<*data*>> is sent to the WAN
- From the WAN the data gets forwarded iteratively until it reaches the "remote" LAN
- From the "remote" LAN the <<*data*>> is transfered to the remote host

There are two layers in the communication process

- For the communication between hosts and LAN
- From the LAN to WAN, and WAN to WAN (this is grouped equally)

THE PHYSICAL LAYER (PHY)

To communicate we need a **physical** link, a connection

THE PHYSICAL LAYER (PHY)

To communicate we need a **physical** link, a connection

- A physical connection between the hosts and the switch (or host to host, etc.)

THE PHYSICAL LAYER (PHY)

To communicate we need a **physical** link, a connection

- A physical connection between the hosts and the switch (or host to host, etc.)
  - To transmit data from one host to another over different mediums. For instance, IEEE 802.3 wired Ethernet standard, IEEE 802.11 Wireless Ethernet standard, and IEEE 802.15 Bluetooth standard

THE PHYSICAL LAYER (PHY)

To communicate we need a **physical** link, a connection

- A physical connection between the hosts and the switch (or host to host, etc.)
  - To transmit data from one host to another over different mediums. For instance, IEEE 802.3 wired Ethernet standard, IEEE 802.11 Wireless Ethernet standard, and IEEE 802.15 Bluetooth standard
  - The voltage used to interpret 0s and 1s, which pin/cable has which data, the speed to transmit, the radio-frequency, the multiplexing, the time-slots to transmit, how to handle collisions between wireless-packets and many other parameters are part of the PHY later

THE PHYSICAL LAYER (PHY)

To communicate we need a **physical** link, a connection

- A physical connection between the hosts and the switch (or host to host, etc.)
  - To transmit data from one host to another over different mediums. For instance, IEEE 802.3 wired Ethernet standard, IEEE 802.11 Wireless Ethernet standard, and IEEE 802.15 Bluetooth standard
  - The voltage used to interpret 0s and 1s, which pin/cable has which data, the speed to transmit, the radio-frequency, the multiplexing, the time-slots to transmit, how to handle collisions between wireless-packets and many other parameters are part of the PHY later
  - Extensive! Out of the scope of our topics :)

THE PHYSICAL LAYER (PHY)

To communicate we need a **physical** link, a connection

- A physical connection between the hosts and the switch (or host to host, etc.)
  - To transmit data from one host to another over different mediums. For instance, IEEE 802.3 wired Ethernet standard, IEEE 802.11 Wireless Ethernet standard, and IEEE 802.15 Bluetooth standard
  - The voltage used to interpret 0s and 1s, which pin/cable has which data, the speed to transmit, the radio-frequency, the multiplexing, the time-slots to transmit, how to handle collisions between wireless-packets and many other parameters are part of the PHY later
  - Extensive! Out of the scope of our topics :)

message.zip: Used to transmit data from a device to other *physically* connected devices

The Data Link Layer

Data-Link (A.K.A Layer 2) Layer functions?

THE DATA LINK LAYER

Data-Link (A.K.A Layer 2) Layer functions?

- PHY is clearly not enough…

THE DATA LINK LAYER

Data-Link (A.K.A Layer 2) Layer functions?

- PHY is clearly not enough...
- Allows hosts to send data to several hosts (addressing) with a single physical connection

THE DATA LINK LAYER

Data-Link (A.K.A Layer 2) Layer functions?

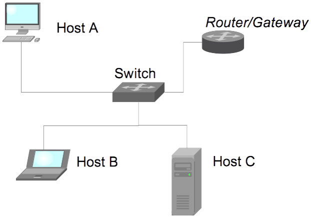
- PHY is clearly not enough. . .
- Allows hosts to send data to several hosts (addressing) with a single physical connection
- Also important, it detects potential physical transmission errors (CRC)

THE DATA LINK LAYER

Data-Link (A.K.A Layer 2) Layer functions?

- PHY is clearly not enough...
- Allows hosts to send data to several hosts (addressing) with a single physical connection
- Also important, it detects potential physical transmission errors (CRC)

Typical LAN architecture



The Data Link Layer (cont.)

Switching

THE DATA LINK LAYER (CONT.)

Switching

- Switched networks have stood the test of time, differently from other, e.g., Token Rings

THE DATA LINK LAYER (CONT.)

Switching

- Switched networks have stood the test of time, differently from other, e.g., Token Rings
- Easy to expand (perhaps that's why (:)

THE DATA LINK LAYER (CONT.)

Switching

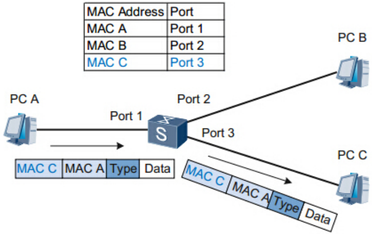
- Switched networks have stood the test of time, differently from other, e.g., Token Rings
- Easy to expand (perhaps that's why (:)
- All the magic of switching, a **switching table**. Device $\mapsto$ Port mapping (A non-injective non-surjective function)

THE DATA LINK LAYER (CONT.)

Switching

- Switched networks have stood the test of time, differently from other, e.g., Token Rings
- Easy to expand (perhaps that's why (:)
- All the magic of switching, a **switching table**. Device $\mapsto$ Port mapping (A non-injective non-surjective function)

Switching tables

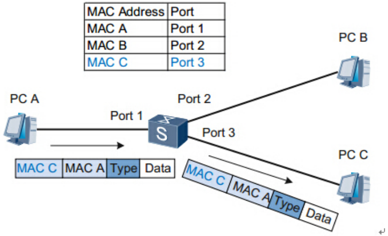


THE DATA LINK LAYER (CONT.)

Switching

- Switched networks have stood the test of time, differently from other, e.g., Token Rings
- Easy to expand (perhaps that's why (:)
- All the magic of switching, a **switching table**. Device $\mapsto$ Port mapping (A non-injective non-surjective function)

Switching tables



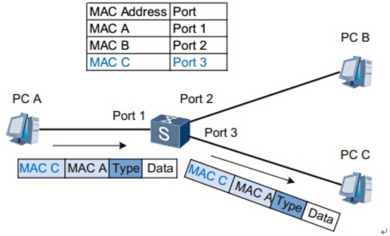
- Think about two switches connected together, what would happen?

THE DATA LINK LAYER (CONT.)

Switching

- Switched networks have stood the test of time, differently from other, e.g., Token Rings
- Easy to expand (perhaps that's why (:)
- All the magic of switching, a **switching table**. Device $\mapsto$ Port mapping (A non-injective non-surjective function)

Switching tables



- Think about two switches connected together, what would happen?
- Answer: In a single port many devices!

THE DATA LINK LAYER (CONT. 2)

The **Ethernet** Protocol Frame Structure

Preamble	Start Delim	Dest MAC	Src MAC	Ether Type	Payload	FCS
(7 bytes)	*(1 byte)*	*(6 bytes)*	*(6 bytes)*	*(2 bytes)*	*(1500-46 bytes)*	*(4 bytes)*

THE DATA LINK LAYER (CONT. 2)

The Ethernet Protocol Frame Structure

Preamble	Start Delim	Dest MAC	Src MAC	Ether Type	Payload	FCS
(7 bytes)	*(1 byte)*	*(6 bytes)*	*(6 bytes)*	*(2 bytes)*	*(1500-46 bytes)*	*(4 bytes)*

- Preamble + Start of Frame Delimiter: used by the operating system to determine / synchronize the packet data(sequential data)

THE DATA LINK LAYER (CONT. 2)

The **Ethernet** Protocol Frame Structure

Preamble	Start Delim	Dest MAC	Src MAC	Ether Type	Payload	FCS
(7 bytes)	*(1 byte)*	*(6 bytes)*	*(6 bytes)*	*(2 bytes)*	*(1500-46 bytes)*	*(4 bytes)*

- Preamble + Start of Frame Delimiter: used by the operating system to determine / synchronize the packet data(sequential data)
- Destination Media Access Control (MAC) address: destination *unique* (2^{48}) Identifier for network interfaces

THE DATA LINK LAYER (CONT. 2)

The **Ethernet** Protocol Frame Structure

Preamble	Start Delim	Dest MAC	Src MAC	Ether Type	Payload	FCS
(7 bytes)	*(1 byte)*	*(6 bytes)*	*(6 bytes)*	*(2 bytes)*	*(1500-46 bytes)*	*(4 bytes)*

- Preamble + Start of Frame Delimiter: used by the operating system to determine / synchronize the packet data(sequential data)
- Destination Media Access Control (MAC) address: destination *unique* (2^{48}) Identifier for network interfaces
- Source MAC: source device address, typically represented as 6 hex pairs separated by colons (b8:2a:14:36:dc:86)

THE DATA LINK LAYER (CONT. 2)

The Ethernet Protocol Frame Structure

Preamble (7 bytes)	Start Delim (1 byte)	Dest MAC (6 bytes)	Src MAC (6 bytes)	Ether Type (2 bytes)	Payload (1500-46 bytes)	FCS (4 bytes)

- Preamble + Start of Frame Delimiter: used by the operating system to determine / synchronize the packet data(sequential data)
- Destination Media Access Control (MAC) address: destination *unique* (2^{48}) Identifier for network interfaces
- Source MAC: source device address, typically represented as 6 hex pairs separated by colons (b8:2a:14:36:dc:86)
- Ether Type: what type of <<*data*>> it carries.

THE DATA LINK LAYER (CONT. 2)

The Ethernet Protocol Frame Structure

Preamble (7 bytes)	Start Delim (1 byte)	Dest MAC (6 bytes)	Src MAC (6 bytes)	Ether Type (2 bytes)	Payload (1500-46 bytes)	FCS (4 bytes)

- Preamble + Start of Frame Delimiter: used by the operating system to determine / synchronize the packet data(sequential data)
- Destination Media Access Control (MAC) address: destination *unique* (2^{48}) Identifier for network interfaces
- Source MAC: source device address, typically represented as 6 hex pairs separated by colons (b8:2a:14:36:dc:86)
- Ether Type: what type of <<*data*>> it carries.
- Payload: the <<*data*>>

The Data Link Layer (cont. 2)

The **Ethernet** Protocol Frame Structure

Preamble	Start Delim	Dest MAC	Src MAC	Ether Type	Payload	FCS
(7 bytes)	*(1 byte)*	*(6 bytes)*	*(6 bytes)*	*(2 bytes)*	*(1500-46 bytes)*	*(4 bytes)*

- Preamble + Start of Frame Delimiter: used by the operating system to determine / synchronize the packet data(sequential data)
- Destination Media Access Control (MAC) address: destination *unique* (2^{48}) Identifier for network interfaces
- Source MAC: source device address, typically represented as 6 hex pairs separated by colons (b8:2a:14:36:dc:86)
- Ether Type: what type of <<*data*>> it carries.
- Payload: the <<*data*>>
- Frame Checking Sequence: error detection and correction code (Prof. Yevtushenko's class (:)

MOVING FROM LAYER 2...

What we know now

MOVING FROM LAYER 2...

What we know now

- How to move <<*data*>> from one host to another in the same local area network

MOVING FROM LAYER 2...

What we know now

- How to move <<*data*>> from one host to another in the same local area network
- The typical architecture of a LAN

MOVING FROM LAYER 2...

What we know now

- How to move <<*data*>> from one host to another in the same local area network
- The typical architecture of a LAN
- The L2 frame structure, media access and addressing principles

MOVING FROM LAYER 2...

What we know now

- How to move <<*data*>> from one host to another in the same local area network
- The typical architecture of a LAN
- The L2 frame structure, media access and addressing principles

What we **don't** know

MOVING FROM LAYER 2. . .

What we know now

- How to move <<*data*>> from one host to another in the same local area network
- The typical architecture of a LAN
- The L2 frame structure, media access and addressing principles

What we **don't** know

- Many things — :) not relevant or too specific things to focus on, e.g., VLANs (IEEE 802.11Q)

MOVING FROM LAYER 2...

What we know now

- How to move <<*data*>> from one host to another in the same local area network
- The typical architecture of a LAN
- The L2 frame structure, media access and addressing principles

What we **don't** know

- Many things — :) not relevant or too specific things to focus on, e.g., VLANs (IEEE 802.11Q)
- How to move <<*data*>> from one host to another in **different** (inter) networks

MOVING FROM LAYER 2...

What we know now

- How to move <<*data*>> from one host to another in the same local area network
- The typical architecture of a LAN
- The L2 frame structure, media access and addressing principles

What we **don't** know

- Many things — :) not relevant or too specific things to focus on, e.g., VLANs (IEEE 802.11Q)
- How to move <<*data*>> from one host to another in **different** (inter) networks
- The holly grail of networking, **THE Internet**

THE NETWORK LAYER

Network (A.K.A Layer 3) Layer functions?

THE NETWORK LAYER

Network (A.K.A Layer 3) Layer functions?

- Routing / inter-network communication

THE NETWORK LAYER

Network (A.K.A Layer 3) Layer functions?

- Routing / inter-network communication
- Connectionless (best effort) communication

THE NETWORK LAYER

Network (A.K.A Layer 3) Layer functions?

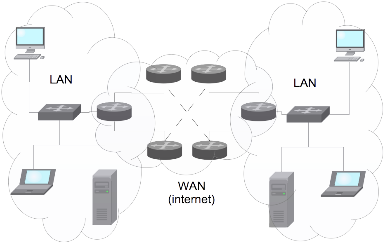
- Routing / inter-network communication
- Connectionless (best effort) communication
- Hierarchical host addressing addressing

THE NETWORK LAYER

Network (A.K.A Layer 3) Layer functions?

- Routing / inter-network communication
- Connectionless (best effort) communication
- Hierarchical host addressing addressing

Typical Inter-network architecture



THE NETWORK LAYER (CONT.)

Routing

The Network Layer (cont.)

Routing

- The core of L3, based on the routing (moving textit<<data>>) from one network to the other

THE NETWORK LAYER (CONT.)

Routing

- The core of L3, based on the routing (moving textit<<data>>) from one network to the other
- Based on Internet Protocol (IP) addresses, e.g.: 91.221.61.55 (typically represented with 4 decimal octets separated by a '.')

The Network Layer (cont.)

Routing

- The core of L3, based on the routing (moving textit<<data>>) from one network to the other
- Based on Internet Protocol (IP) addresses, e.g.: 91.221.61.55 (typically represented with 4 decimal octets separated by a '.')
- Local networks have IP addresses, which belong to the **same** network

The Network Layer (cont.)

Routing

- The core of L3, based on the routing (moving textit<<data>>) from one network to the other
- Based on Internet Protocol (IP) addresses, e.g.: 91.221.61.55 (typically represented with 4 decimal octets separated by a '.')
- Local networks have IP addresses, which belong to the **same** network
- Network masks determine the local network. A "netmask" is nothing more than a 4 byte *and* bitwise mask. A common notation is the Classless Inter-Domain Routing (CIDR), the most significant bits are the network address. For example, "/16" means the netmask is equal to: 11111111111111110000000000000000

THE NETWORK LAYER (CONT. 2)

Example

THE NETWORK LAYER (CONT. 2)

Example

- The CIDR (IP address/netmask) 192.168.1.124/24

The Network Layer (cont. 2)

Example

- The CIDR (IP address/netmask) 192.168.1.124/24

$$
\begin{aligned}
IPaddress\ :& 11000000101010000000000101111100 \\
& \&(bitwise) \\
Netmask\ :& 11111111111111111111111100000000 (255.255.255.0) \\
Network\ :& 11000000101010000000000100000000 = 192.168.1.0/24 \\
NetRange\ :& 192.168.1.1 - 192.168.1.255
\end{aligned}
$$

- First address is *associated* to the gateway, the device in charge of inter-routing (more about this later)

THE NETWORK LAYER (CONT. 2)

Example

- The CIDR (IP address/netmask) 192.168.1.124/24

$$
\begin{aligned}
IPaddress :& 11000000101010000000000101111100 \\
& \&(bitwise) \\
Netmask :& 11111111111111111111111100000000 (255.255.255.0) \\
Network :& 11000000101010000000000100000000 = 192.168.1.0/24 \\
NetRange :& 192.168.1.1 - 192.168.1.255
\end{aligned}
$$

- First address is *associated* to the gateway, the device in charge of inter-routing (more about this later)
- Last address is the broadcast address. When data is sent to this address, it is routed to **all** nodes in the network

THE NETWORK LAYER (CONT. 3)

For you

THE NETWORK LAYER (CONT. 3)

For you

- The CIDR 192.168.1.124/27 (answers in 4 octets decimal)

The Network Layer (cont. 3)

For you

- The CIDR 192.168.1.124/27 (answers in 4 octets decimal)

$$
\begin{aligned}
IPaddress &: \\
Netmask &: \\
Network &: \\
NetRange &: \\
BroadcastAddress &:
\end{aligned}
$$

THE NETWORK LAYER (CONT. 3)

For you

- The CIDR 192.168.1.124/27 (answers in 4 octets decimal)

$$\begin{aligned} IPaddress &: 192.168.1.124 \\ Netmask &: 255.255.255.224 \\ Network &: 192.168.1.96/27 \\ NetRange &: 192.168.1.97 - 192.168.1.127 \\ BroadcastAddress &: 192.168.1.127 \end{aligned}$$

The Network Layer (cont. 3)

For you

- The CIDR 192.168.1.124/27 (answers in 4 octets decimal)

$$IPaddress : 192.168.1.124$$
$$Netmask : 255.255.255.224$$
$$Network : 192.168.1.96/27$$
$$NetRange : 192.168.1.97 - 192.168.1.127$$
$$BroadcastAddress : 192.168.1.127$$

- What is the gateway address?

THE NETWORK LAYER (CONT. 3)

For you

- The CIDR 192.168.1.124/27 (answers in 4 octets decimal)

$$
\begin{aligned}
IPaddress &: 192.168.1.124 \\
Netmask &: 255.255.255.224 \\
Network &: 192.168.1.96/27 \\
NetRange &: 192.168.1.97 - 192.168.1.127 \\
BroadcastAddress &: 192.168.1.127
\end{aligned}
$$

- What is the gateway address?
  - Not enough information...

THE NETWORK LAYER (CONT. 3)

For you

- The CIDR 192.168.1.124/27 (answers in 4 octets decimal)

$$\begin{aligned} IPaddress &: 192.168.1.124 \\ Netmask &: 255.255.255.224 \\ Network &: 192.168.1.96/27 \\ NetRange &: 192.168.1.97 - 192.168.1.127 \\ BroadcastAddress &: 192.168.1.127 \end{aligned}$$

- What is the gateway address?
  - Not enough information...
- Speaking about gateways, how do they decide where to send the <<*data*>>?

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td colspan="2">IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="5" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="3">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="9">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="9">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="9">Options
(IHL – 20 bytes)</td></tr>
</table>

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Version: IP version, “current”, version 4. “Next” version 6

THE NETWORK LAYER (CONT. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Version: IP version, “current”, version 4. “Next” version 6
- IHL: Internet Header Length (length of the current header)

The Network Layer (cont. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td colspan="2">IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="5" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="3">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="9">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="9">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="9">Options
(IHL – 20 bytes)</td></tr>
</table>

- Version: IP version, “current”, version 4. “Next” version 6
- IHL: Internet Header Length (length of the current header)
- DSCP: Differentiated Services Code Point, classify traffic, e.g., voice applications vs. web pages

THE NETWORK LAYER (CONT. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

THE NETWORK LAYER (CONT. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- ECN: Explicit Congestion Notification, in short, it tries to avoid congestion by notifying the receiving end of its congestion

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- ECN: Explicit Congestion Notification, in short, it tries to avoid congestion by notifying the receiving end of its congestion
- Total Length: The length of the packet (header + data)

THE NETWORK LAYER (CONT. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- ECN: Explicit Congestion Notification, in short, it tries to avoid congestion by notifying the receiving end of its congestion
- Total Length: The length of the packet (header + data)
- Identification: Simple ID, normally set as an auto increment

THE NETWORK LAYER (CONT. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Flags: 0 = always 0 (reserved), DF = don't fragment (send this packet complete), MF = more fragments coming from this packet.

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Flags: 0 = always 0 (reserved), DF = don't fragment (send this packet complete), MF = more fragments coming from this packet.
- Fragment offset: if a packet is fragmented, this shows the position of this fragment relative to the beginning of the original IP packet

THE NETWORK LAYER (CONT. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

THE NETWORK LAYER (CONT. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Time To Live: To avoid infinite loops, typically set to the maximum number of hops

THE NETWORK LAYER (CONT. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Time To Live: To avoid infinite loops, typically set to the maximum number of hops
- Protocol: Which protocol (<<*data*>>) the IP datagram carries

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Time To Live: To avoid infinite loops, typically set to the maximum number of hops
- Protocol: Which protocol (<<*data*>>) the IP datagram carries
- Header checksum: error detection for the IP header

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Time To Live: To avoid infinite loops, typically set to the maximum number of hops
- Protocol: Which protocol (<<*data*>>) the IP datagram carries
- Header checksum: error detection for the IP header
- Source IP address: Sending host IP address

The Network Layer (cont. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

THE NETWORK LAYER (CONT. 00000100)

The **IPv4** Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Destination IP address: Receiving host IP address

The Network Layer (cont. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Destination IP address: Receiving host IP address
- Options: Not often used, e.g., RFC 4782, quick start (for sending rate)

THE NETWORK LAYER (CONT. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td>IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="4" rowspan="2">Identification

(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="2">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="8">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="8">Options
(IHL – 20 bytes)</td></tr>
</table>

- Destination IP address: Receiving host IP address
- Options: Not often used, e.g., RFC 4782, quick start (for sending rate)

With these fields, how does a router decide where to forward the IP datagram?

THE NETWORK LAYER (CONT. 00000100)

The IPv4 Datagram Structure

<table>
<tr><td>Version
(1 nibble)</td><td colspan="2">IHL
(1 nibble)</td><td>DSCP
(6 bits)</td><td>ECN
(2 bits)</td><td colspan="4">Total Length
(2 bytes)</td></tr>
<tr><td colspan="5" rowspan="2">Identification
(2 bytes)</td><td colspan="3">Flags</td><td rowspan="2">Fragment offset
(13 bits)</td></tr>
<tr><td>0
(1 bit)</td><td>DF
(1 bit)</td><td>MF
(1 bit)</td></tr>
<tr><td colspan="2">Time to Live
(1 byte)</td><td colspan="3">Protocol
(1 byte)</td><td colspan="4">Header checksum
(2 bytes)</td></tr>
<tr><td colspan="9">Source IP address
(4 bytes)</td></tr>
<tr><td colspan="9">Destination IP address
(4 bytes)</td></tr>
<tr><td colspan="9">Options
(IHL – 20 bytes)</td></tr>
</table>

- Destination IP address: Receiving host IP address
- Options: Not often used, e.g., RFC 4782, quick start (for sending rate)

With these fields, how does a router decide where to forward the IP datagram?

Routing protocols

ROUTING PROTOCOLS

Working principles

ROUTING PROTOCOLS

Working principles

- Basic and most important idea: find the gateway and deliver the data to it, gateway IP in the same network, MAC found via the Address Resolution Protocol (ARP), IP $\mapsto$ MAC translation

Routing protocols

Working principles

- Basic and most important idea: find the gateway and deliver the data to it, gateway IP in the same network, MAC found via the Address Resolution Protocol (ARP), IP $\mapsto$ MAC translation
- The gateway decides to which peer (router) to forward the traffic based on the destination IP address

ROUTING PROTOCOLS

Working principles

- Basic and most important idea: find the gateway and deliver the data to it, gateway IP in the same network, MAC found via the Address Resolution Protocol (ARP), IP $\mapsto$ MAC translation
- The gateway decides to which peer (router) to forward the traffic based on the destination IP address
- Classical shortest path-finding algorithm application, e.g., Bellman-Ford algorithm for classical Routing Information Protocol (RIP) protocol

ROUTING PROTOCOLS

Working principles

- Basic and most important idea: find the gateway and deliver the data to it, gateway IP in the same network, MAC found via the Address Resolution Protocol (ARP), IP $\mapsto$ MAC translation
- The gateway decides to which peer (router) to forward the traffic based on the destination IP address
- Classical shortest path-finding algorithm application, e.g., Bellman-Ford algorithm for classical Routing Information Protocol (RIP) protocol
- Shortest path is not not always the “best” path, specially for exterior gateway protocols, Internet service providers might prefer to pass traffic to a longer path which is less expensive (in terms of ₽, €, $), e.g., the Border Gateway Protocol — governs the Internet

BEYOND MOVING <<*data*>>

What we know

BEYOND MOVING <<*data*>>

What we know

- How to move <<*data*>> from one host into another.

BEYOND MOVING <<*data*>>

What we know

- How to move <<*data*>> from one host into another.
- (Without guaranteeing delivery, an IP packet can be lost, discarded and there is no mechanism to acknowledge delivery. IP is known to be “best effort” delivery)

BEYOND MOVING <<*data*>>

What we know

- How to move <<*data*>> from one host into another.
- (Without guaranteeing delivery, an IP packet can be lost, discarded and there is no mechanism to acknowledge delivery. IP is known to be "best effort" delivery)

What we do not know

BEYOND MOVING <<*data*>>

What we know

- How to move <<*data*>> from one host into another.
- (Without guaranteeing delivery, an IP packet can be lost, discarded and there is no mechanism to acknowledge delivery. IP is known to be "best effort" delivery)

What we do not know

- How to differentiate between data sent to different applications

BEYOND MOVING <<*data*>>

What we know

- How to move <<*data*>> from one host into another.
- (Without guaranteeing delivery, an IP packet can be lost, discarded and there is no mechanism to acknowledge delivery. IP is known to be "best effort" delivery)

What we do not know

- How to differentiate between data sent to different applications
- How to make a **reliable** delivery

THE TRANSPORT LAYER

Differentiating application data

THE TRANSPORT LAYER

Differentiating application data

- 1 city different ports $\mapsto$ 1 host different ports

THE TRANSPORT LAYER

Differentiating application data

- 1 city different ports $\mapsto$ 1 host different ports

The User Datagram Protocol (UDP)

Source Port *(2 bytes)*	**Destination Port** *(2 bytes)*
Length *(2 bytes)*	**Checksum** *(2 bytes)*

THE TRANSPORT LAYER

Differentiating application data

- 1 city different ports $\mapsto$ 1 host different ports

The User Datagram Protocol (UDP)

Source Port *(2 bytes)*	**Destination Port** *(2 bytes)*
Length *(2 bytes)*	**Checksum** *(2 bytes)*

- Source Port: The port of the originating host

THE TRANSPORT LAYER

Differentiating application data

- 1 city different ports $\mapsto$ 1 host different ports

The User Datagram Protocol (UDP)

Source Port *(2 bytes)*	**Destination Port** *(2 bytes)*
Length *(2 bytes)*	**Checksum** *(2 bytes)*

- Source Port: The port of the originating host
- Destination Port; The target port

THE TRANSPORT LAYER

Differentiating application data

- 1 city different ports $\mapsto$ 1 host different ports

The User Datagram Protocol (UDP)

Source Port *(2 bytes)*	**Destination Port** *(2 bytes)*
Length *(2 bytes)*	**Checksum** *(2 bytes)*

- Source Port: The port of the originating host
- Destination Port; The target port
- Length: The size of the UDP packet

THE TRANSPORT LAYER

Differentiating application data

- 1 city different ports $\mapsto$ 1 host different ports

The User Datagram Protocol (UDP)

Source Port *(2 bytes)*	**Destination Port** *(2 bytes)*
Length *(2 bytes)*	**Checksum** *(2 bytes)*

- Source Port: The port of the originating host
- Destination Port; The target port
- Length: The size of the UDP packet
- Checksum: error detection for the UDP header

THE TRANSPORT LAYER (CONT.)

UDP Characteristics

THE TRANSPORT LAYER (CONT.)

UDP Characteristics

- Very lightweight (L2 headers + L3 headers + UDP headers + data conform a packet, not much overhead)

THE TRANSPORT LAYER (CONT.)

UDP Characteristics

- Very lightweight (L2 headers + L3 headers + UDP headers + data conform a packet, not much overhead)
- Unordered, the order the sending host sent is not guaranteed at the receiving end (transport over IP and no more mechanisms)

THE TRANSPORT LAYER (CONT.)

UDP Characteristics

- Very lightweight (L2 headers + L3 headers + UDP headers + data conform a packet, not much overhead)
- Unordered, the order the sending host sent is not guaranteed at the receiving end (transport over IP and no more mechanisms)
- Unreliable, no facilities to guarantee data reaches the receiving end

THE TRANSPORT LAYER (CONT.)

UDP Characteristics

- Very lightweight (L2 headers + L3 headers + UDP headers + data conform a packet, not much overhead)
- Unordered, the order the sending host sent is not guaranteed at the receiving end (transport over IP and no more mechanisms)
- Unreliable, no facilities to guarantee data reaches the receiving end
- Very good for real-time protocols, e.g., media applications (1 packet of voice lost won't be perceived by the human ear, while delay would)

The Transport Layer (cont.)

UDP Characteristics

- Very lightweight (L2 headers + L3 headers + UDP headers + data conform a packet, not much overhead)
- Unordered, the order the sending host sent is not guaranteed at the receiving end (transport over IP and no more mechanisms)
- Unreliable, no facilities to guarantee data reaches the receiving end
- Very good for real-time protocols, e.g., media applications (1 packet of voice lost won't be perceived by the human ear, while delay would)

What if we want the "opposite"?

THE TRANSPORT LAYER (CONT.)

UDP Characteristics

- Very lightweight (L2 headers + L3 headers + UDP headers + data conform a packet, not much overhead)
- Unordered, the order the sending host sent is not guaranteed at the receiving end (transport over IP and no more mechanisms)
- Unreliable, no facilities to guarantee data reaches the receiving end
- Very good for real-time protocols, e.g., media applications (1 packet of voice lost won't be perceived by the human ear, while delay would)

What if we want the "opposite"?

Transmission Control Protocol (TCP)!

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>N
S</td><td>C
R
W</td><td>E
C
E</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Source Port, Destination Port = UDP

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Source Port, Destination Port = UDP
- Sequence Number: If SYN = 1, first data byte reference, corresponding Acknowledgment number is this number + 1. If SYN = 0, accumulated data byte from initial synchronization

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="7">**Source Port**
(2 bytes)</td><td colspan="5">**Destination Port**
(2 bytes)</td></tr>
<tr><td colspan="12">**Sequence Number**
(4 bytes)</td></tr>
<tr><td colspan="12">**Acknowledgement Number**
(4 bytes)</td></tr>
<tr><td>**Data Offset**
(1 nibble)</td><td>**Resrved**
(3 bits)</td><td>N S</td><td>C R W</td><td>E C E</td><td>U R G</td><td>A C K</td><td>P S H</td><td>R S T</td><td>S Y N</td><td>F I N</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="7">**Checksum**
(2 bytes)</td><td colspan="5">**Urgent Pointer**
(2 bytes)</td></tr>
<tr><td colspan="12">**Options**
(Data offset -5 bytes)</td></tr>
</table>

- Acknowledgment Number: if ACK =1, the expected byte to receive (by the receiver), acknowledges all previous data bytes, if this is in response to a SYN, it means just the initial sequance number setup is being acknowledged (no data)

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="6">Source Port
(2 bytes)</td><td colspan="6">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>N S</td><td>C R W</td><td>E C E</td><td>U R G</td><td>A C K</td><td>P S H</td><td>R S T</td><td>S Y N</td><td>F I N</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="6">Checksum
(2 bytes)</td><td colspan="6">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Acknowledgment Number: if ACK =1, the expected byte to receive (by the receiver), acknowledges all previous data bytes, if this is in response to a SYN, it means just the initial sequance number setup is being acknowledged (no data)
- Data Offset: measured in 4 bytes data size (5 = 20 bytes) where the actual data begins (header ends)

Transport Layer — TCP

<table>
<tr><td colspan="6">Source Port
(2 bytes)</td><td colspan="6">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="6">Checksum
(2 bytes)</td><td colspan="6">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Flags:

Transport Layer — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>N S</td><td>C R W</td><td>E C E</td><td>U R G</td><td>A C K</td><td>P S H</td><td>R S T</td><td>S Y N</td><td>F I N</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Flags:
  - NS, CWS, ECE: IP Explicit congestion Notification related

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Flags:
  - NS, CWS, ECE: IP Explicit congestion Notification related
  - SYN: initiates connection

Transport Layer — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Flags:
  - NS, CWS, ECE: IP Explicit congestion Notification related
  - SYN: initiates connection
  - ACK: acknowledges data

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Flags:
  - NS, CWS, ECE: IP Explicit congestion Notification related
  - SYN: initiates connection
  - ACK: acknowledges data
  - FIN: closes the connection (normally), RST: Aborts the connection (error)
  - PSH: move to the application the data, do not buffer it

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="8">Source Port
(2 bytes)</td><td colspan="4">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="8">Checksum
(2 bytes)</td><td colspan="4">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Flags:
  - NS, CWS, ECE: IP Explicit congestion Notification related
  - SYN: initiates connection
  - ACK: acknowledges data
  - FIN: closes the connection (normally), RST: Aborts the connection (error)
  - PSH: move to the application the data, do not buffer it
  - URG: Urgent field is “active”

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

Transport Layer — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Window Size: Flow control, how many bytes from the “ack” the host is capable of receiving

TRANSPORT LAYER — TCP

<table>
<tr><td colspan="9">Source Port
(2 bytes)</td><td colspan="3">Destination Port
(2 bytes)</td></tr>
<tr><td colspan="12">Sequence Number
(4 bytes)</td></tr>
<tr><td colspan="12">Acknowledgement Number
(4 bytes)</td></tr>
<tr><td>Data Offset
(1 nibble)</td><td>Resrved
(3 bits)</td><td>NS</td><td>CRW</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size
(2 bytes)</td></tr>
<tr><td colspan="9">Checksum
(2 bytes)</td><td colspan="3">Urgent Pointer
(2 bytes)</td></tr>
<tr><td colspan="12">Options
(Data offset -5 bytes)</td></tr>
</table>

- Window Size: Flow control, how many bytes from the “ack” the host is capable of receiving
- Urgent Pointer: **last** urgent byte in the data (i.e., urgent pointer and not urgent offset)

TCP — FINAL REMARKS

TCP — FINAL REMARKS

- Establishes connection with a “3-way handshake” SYN; SYN-ACK; ACK

TCP — FINAL REMARKS

- Establishes connection with a “3-way handshake” SYN; SYN-ACK; ACK
- After connection is established a FIN closes transmission of sending host (single FIN, “half-open“ connection)

TCP — FINAL REMARKS

- Establishes connection with a “3-way handshake” SYN; SYN-ACK; ACK
- After connection is established a FIN closes transmission of sending host (single FIN, “half-open“ connection)
- Guarantees, Ordered, Reliable, Congestion Avoidance <<*data*>> delivery

TCP — FINAL REMARKS

- Establishes connection with a “3-way handshake” SYN; SYN-ACK; ACK
- After connection is established a FIN closes transmission of sending host (single FIN, “half-open“ connection)
- Guarantees, Ordered, Reliable, Congestion Avoidance <<*data*>> delivery
- All this is controlled by the operating systems (I know you’re happy)

TCP — Final Remarks

- Establishes connection with a “3-way handshake” SYN; SYN-ACK; ACK
- After connection is established a FIN closes transmission of sending host (single FIN, “half-open“ connection)
- Guarantees, Ordered, Reliable, Congestion Avoidance <<*data*>> delivery
- All this is controlled by the operating systems (I know you’re happy)

What’s <<*data*>>?

TCP — FINAL REMARKS

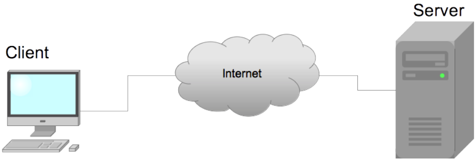
- Establishes connection with a “3-way handshake” SYN; SYN-ACK; ACK
- After connection is established a FIN closes transmission of sending host (single FIN, “half-open“ connection)
- Guarantees, Ordered, Reliable, Congestion Avoidance <<*data*>> delivery
- All this is controlled by the operating systems (I know you’re happy)

What’s <<*data*>>?

Data interchanged by applications! Finally we know what data is, right? ;)

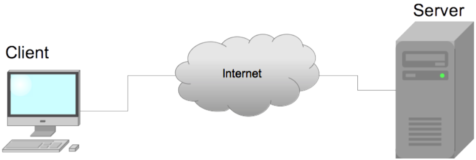
THE APPLICATION LAYER

Typical Architecture



THE APPLICATION LAYER

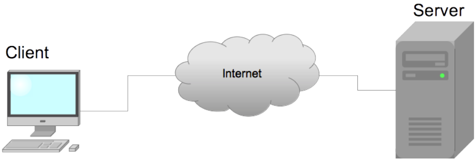
Typical Architecture



- Client-Server architecture, contrary to common belief, still prevails as the most important in communication protocols

THE APPLICATION LAYER

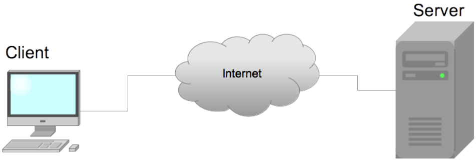
Typical Architecture



- Client-Server architecture, contrary to common belief, still prevails as the most important in communication protocols
- Each protocol chooses the message structure according to its necessities and design

The Application Layer

Typical Architecture



- Client-Server architecture, contrary to common belief, still prevails as the most important in communication protocols
- Each protocol chooses the message structure according to its necessities and design
- Some are like traditional message structure as the whole TCP/IP Stack, some others are “text-based”

The application layer – The Hyper Text Transfer Protocol

Text-based, request-response scheme

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Request elements:

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Request elements:
  - Request line: a request method (e.g., GET) a resource (e.g., /index.php), and a protocol version (e.g., HTTP/1.1)

The application layer – The Hyper Text Transfer Protocol

Text-based, request-response scheme

- Request elements:
  - Request line: a request method (e.g., GET) a resource (e.g., /index.php), and a protocol version (e.g., HTTP/1.1)
  - Header fields: A list of key-value elements separated by a colon with no particular order of fields (e.g., Connection: closed)

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Request elements:
  - Request line: a request method (e.g., GET) a resource (e.g., /index.php), and a protocol version (e.g., HTTP/1.1)
  - Header fields: A list of key-value elements separated by a colon with no particular order of fields (e.g., Connection: closed)
  - An empty line

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Request elements:
  - Request line: a request method (e.g., GET) a resource (e.g., /index.php), and a protocol version (e.g., HTTP/1.1)
  - Header fields: A list of key-value elements separated by a colon with no particular order of fields (e.g., Connection: closed)
  - An empty line
  - An optional body

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Request elements:
  - Request line: a request method (e.g., GET) a resource (e.g., /index.php), and a protocol version (e.g., HTTP/1.1)
  - Header fields: A list of key-value elements separated by a colon with no particular order of fields (e.g., Connection: closed)
  - An empty line
  - An optional body
- Note that all elements are separated by CR+LF

The application layer – The Hyper Text Transfer Protocol

Text-based, request-response scheme

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Response elements:

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Response elements:
  - Status Line: A response code + a reason message (e.g., 200 OK)

The application layer – The Hyper Text Transfer Protocol

Text-based, request-response scheme

- Response elements:
  - Status Line: A response code + a reason message (e.g., 200 OK)
  - Response Header fields: Similar to the request

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Response elements:
  - Status Line: A response code + a reason message (e.g., 200 OK)
  - Response Header fields: Similar to the request
  - An empty line

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Response elements:
  - Status Line: A response code + a reason message (e.g., 200 OK)
  - Response Header fields: Similar to the request
  - An empty line
  - An optional body

THE APPLICATION LAYER – THE HYPER TEXT TRANSFER PROTOCOL

Text-based, request-response scheme

- Response elements:
    - Status Line: A response code + a reason message (e.g., 200 OK)
    - Response Header fields: Similar to the request
    - An empty line
    - An optional body
- Note that all elements are separated by CR+LF

HTTP REQUEST-RESPONSE EXAMPLE

Request

```
GET / HTTP/1.1
Host: google.ru
Connection: Closed
```

HTTP REQUEST-RESPONSE EXAMPLE

Request

```
GET / HTTP/1.1
Host: google.ru
Connection: Closed
```

Response

```
HTTP/1.1 301 Moved Permanently
Location: http://www.google.ru/
Content-Type: text/html; charset=UTF-8
Date: Mon, 08 Feb 2016 19:09:10 GMT
Expires: Wed, 09 Mar 2016 19:09:10 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 218
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.ru/">here</A>.
</BODY></HTML>
```

TO WRAP UP...

The story of how the previous request reached the server

TO WRAP UP...

The story of how the previous request reached the server

- The protocol was HTTP

TO WRAP UP...

The story of how the previous request reached the server

- The protocol was HTTP
- The architecture is client-server

TO WRAP UP...

The story of how the previous request reached the server

- The protocol was HTTP
- The architecture is client-server
- The client request was created by an application, a web browser, for instance

TO WRAP UP...

The story of how the previous request reached the server

- The protocol was HTTP
- The architecture is client-server
- The client request was created by an application, a web browser, for instance
- Then, packet encapsulation...

Packet encapsulation

Packet encapsulation

PACKET ENCAPSULATION

TCP Headers *DST PORT 80*	HTTP Request

Packet encapsulation

IP Headers *DST IP* 79.136.239.38	TCP Headers *DST PORT 80*	HTTP Request

Packet encapsulation

Data-Link Headers	IP Headers *DST IP* `79.136.239.38`	TCP Headers *DST PORT 80*	HTTP Request	Data-Link Footer

Packet encapsulation

Data-Link Headers	IP Headers *DST IP 79.136.239.38*	TCP Headers *DST PORT 80*	HTTP Request	Data-Link Footer

After encapsulation?

PACKET ENCAPSULATION

Data-Link Headers	IP Headers *DST IP 79.136.239.38*	TCP Headers *DST PORT 80*	HTTP Request	Data-Link Footer

After encapsulation?

- First the packet is delivered to the default gateway, MAC address is obtained with ARP

PACKET ENCAPSULATION

Data-Link Headers	IP Headers *DST IP 79.136.239.38*	TCP Headers *DST PORT 80*	HTTP Request	Data-Link Footer

After encapsulation?

- First the packet is delivered to the default gateway, MAC address is obtained with ARP
- Default GW forwards the packet to the other routers until it reaches the destination host, the destination host IP address was obtained using the Domain Name System Protocol (maps from domain names, e.g., google.ru to IP addresses 79.136.239.38)

Packet encapsulation

Data-Link Headers	IP Headers *DST IP 79.136.239.38*	TCP Headers *DST PORT 80*	HTTP Request	Data-Link Footer

After encapsulation?

- First the packet is delivered to the default gateway, MAC address is obtained with ARP
- Default GW forwards the packet to the other routers until it reaches the destination host, the destination host IP address was obtained using the Domain Name System Protocol (maps from domain names, e.g., google.ru to IP addresses 79.136.239.38)
- Along the way, the packet was opened just in the corresponding layer

Packet encapsulation

Data-Link Headers	IP Headers *DST IP 79.136.239.38*	TCP Headers *DST PORT 80*	HTTP Request	Data-Link Footer

After encapsulation?

- First the packet is delivered to the default gateway, MAC address is obtained with ARP
- Default GW forwards the packet to the other routers until it reaches the destination host, the destination host IP address was obtained using the Domain Name System Protocol (maps from domain names, e.g., google.ru to IP addresses 79.136.239.38)
- Along the way, the packet was opened just in the corresponding layer
- The packet gets “d-encapsulated”, and delivered to the receiving application

Out of the scope, but important...

Encrypted protocols

OUT OF THE SCOPE, BUT IMPORTANT...

Encrypted protocols

- Secure Socket Layer (SSL), and Transport Layer Security (TLS) are **popular** methods to send encrypted data

OUT OF THE SCOPE, BUT IMPORTANT...

Encrypted protocols

- Secure Socket Layer (SSL), and Transport Layer Security (TLS) are **popular** methods to send encrypted data
- Generally speaking SSL/TLS encrypts just the application protocol layer, protocols as HTTPS, FTPS IMAPS use such approach

OUT OF THE SCOPE, BUT IMPORTANT...

Encrypted protocols

- Secure Socket Layer (SSL), and Transport Layer Security (TLS) are **popular** methods to send encrypted data
- Generally speaking SSL/TLS encrypts just the application protocol layer, protocols as HTTPS, FTPS IMAPS use such approach
- Over Encrypted data we cannot inspect, therefore Passive Testing of network traces is not (directly) applicable

That is all

OUT OF THE SCOPE, BUT IMPORTANT...

Encrypted protocols

- Secure Socket Layer (SSL), and Transport Layer Security (TLS) are **popular** methods to send encrypted data
- Generally speaking SSL/TLS encrypts just the application protocol layer, protocols as HTTPS, FTPS IMAPS use such approach
- Over Encrypted data we cannot inspect, therefore Passive Testing of network traces is not (directly) applicable

That is all

...for now