Redes de Computadores

Lab 2 - Computer Networks

Manuel P. Ricardo

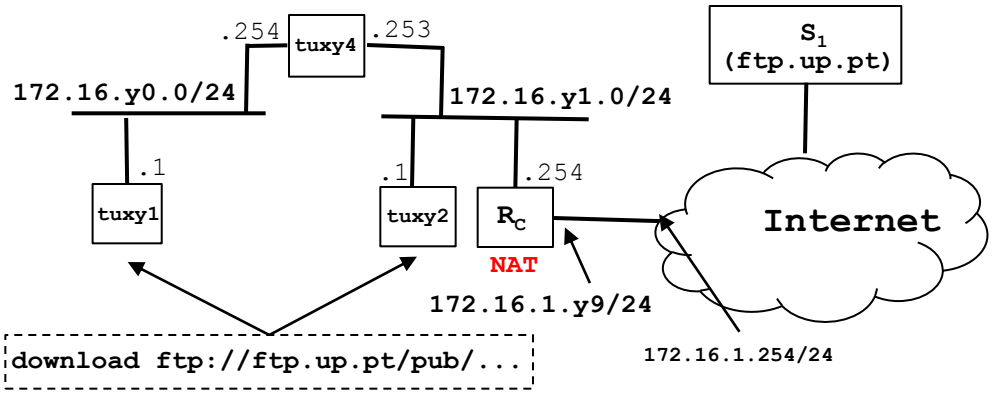
Faculdade de Engenharia da Universidade do Porto

Lab Work – Two parts

- Part 1 – Development of one application

  download ftp://ftp.up.pt/pub/...

- Part 2 – Configuration and study of a computer network



Part 1 – Development of download application

Development of an Application

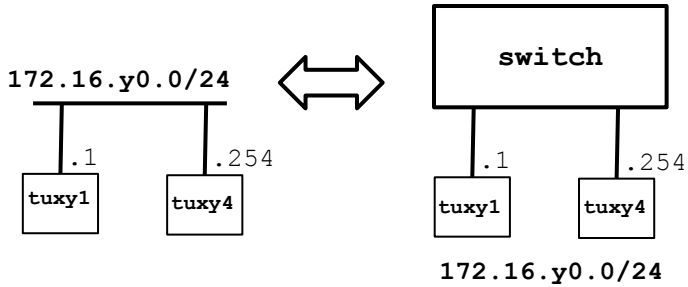
- Develop application `download ftp://ftp.up.pt/pub/...`
  - » Application downloads a single file
  - » Implements FTP application protocol, as described in RFC959
  - » Adopts URL syntax, as described in RFC1738

    `ftp://[<user>:<password>@]<host>/<url-path>`

- Steps
  - » Experiments using Telnet application (Telnet, SMTP, POP, HTTP and FTP); focus on FTP
  - » Specification/design of download application
    - unique use case: connect, login host, passive, get path, success (file saved in CWD) or un-success (indicating failing phase)
    - challenging programming aspects: gethostbyname, sockets, control connection, passive, data connection
  - » Implement a very simple FTP client at home
    - reuse existing programs: **`clientTCP.c, getIP.C`**
- Learning objectives
  - » Describe client – server concept and its peculiarities in TCP/IP
  - » Characterize application protocols in general, characterize URL, describe in detail the behaviour of FTP
  - » Locate and read RFCs
  - » Implement a simple FTP client in C language
  - » Use sockets and TCP in C language
  - » Understand service provided DNS and use it within a client

Part 2 – Configuration and Study of a Network

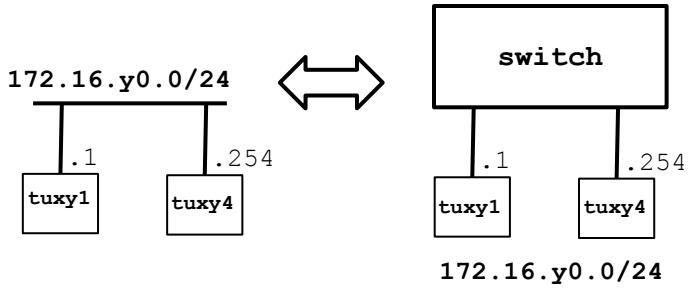
Part 2 / Exp 1- Configure an IP Network



Steps

1. Disconnect switch from netlab and connect tux computers
2. Configure tuxy1 and tuxy4 using **ifconfig** and **route** commands
3. Register the IP and MAC addresses of network interfaces
4. Use ping command to verify connectivity between these computers
5. Inspect forwarding (**route –n**) and ARP (**arp –a**) tables
6. Delete ARP table entries in tuxy1 (**arp –d ipaddress**)
7. Start Wireshark in tuxy1.eth0 and start capturing packets
8. In tuxy1, ping tuxy4 for a few seconds
9. Stop capturing packets
10. Save log study it at home

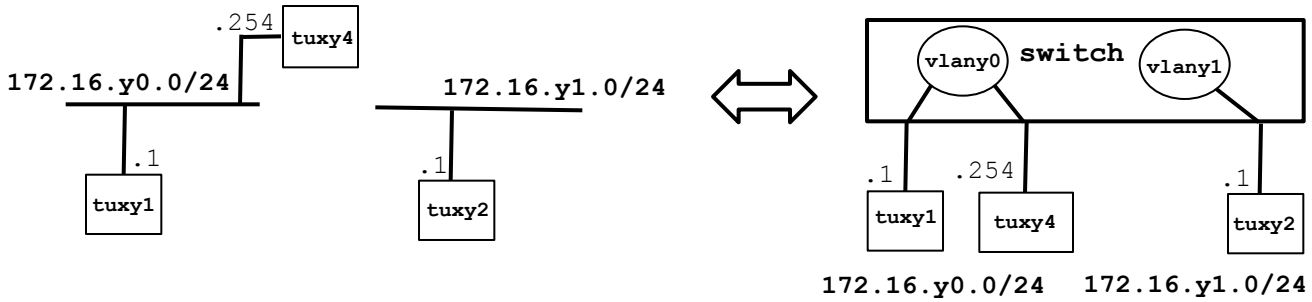
Part 2 / Exp 1- Configure an IP Network



Questions

- What are the ARP packets and what are they used for?
- What are the MAC and IP addresses of ARP packets and why?
- What packets does the ping command generate?
- What are the MAC and IP addresses of the ping packets?
- How to determine if a receiving Ethernet frame is ARP, IP, ICMP?
- How to determine the length of a receiving frame?
- What is the loopback interface and why is it important?

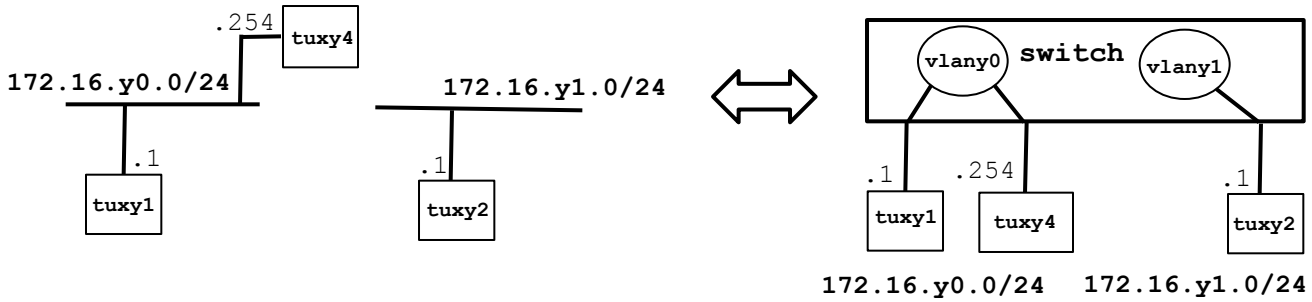
Part 2 / Exp 2 – Implement two virtual LANs in a switch



Steps

1. Configure tuxy2 and annotate its IP and MAC addresses
2. Create vlany0 in the switch and add corresponding ports
3. Create vlany1 and add corresponding port
4. Start capture at tuxy1.eth0
5. In tuxy1, ping tuxy4 and then ping tuxy2
6. Stop capture and save log
7. Start new captures in tuxy1.eth0, tuxy4.eth0, and tuxy2.eth0
8. In tuxy1, do ping broadcast (ping –b 172.16.y0.255) for a few seconds
9. Observe results, stop captures and save logs
10. Repeat steps 7, 8 and 9, but now do
    ping broadcast in tuxy2 (ping –b 172.16.y1.255)

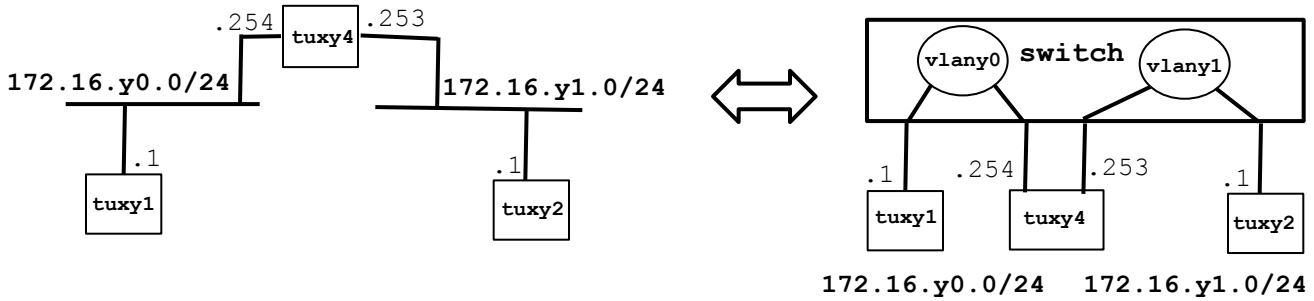
Part 2 / Exp 2 – Implement two virtual LANs in a switch



Questions

- How to configure vlany0?
- How many broadcast domains are there? How can you conclude it from the logs?

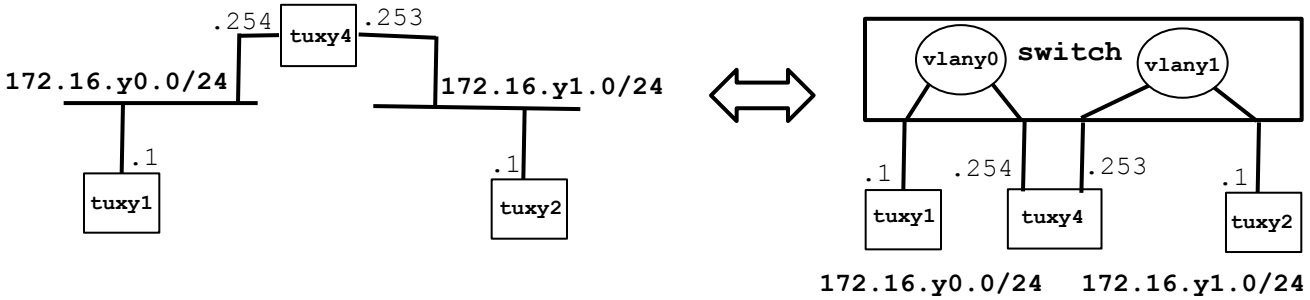
Part 2 / Exp 3 – Configure a Router in Linux



Steps

1. Transform tuxy4 (linux machine) into a router
   - Configure also tuxy4.eth1
   - Enable IP forwarding
   - Disable ICMP echo-ignore-broadcast
2. Observe MAC addresses and IP addresses in tuxy4.eth0 and tuxy4.eth1
3. Reconfigure tuxy1 and tuxy2 so that each of them can reach the other
4. Observe the routes available at the 3 tuxes (route –n)
5. Start capture at tuxy1
6. From tuxy1, ping the other network interfaces (171.16.y0.254, 171.16.y1.253, 171.16.y1.1) and verify if there is connectivity.
7. Stop capture and save logs
8. Start capture in tuxy4; use 2 instances of Wireshark, one per network interface
9. Clean the ARP tables in the 3 tuxes
10. In tuxy1, ping tuxy2 for a few seconds.
11. Stop captures in tuxy4 and save logs

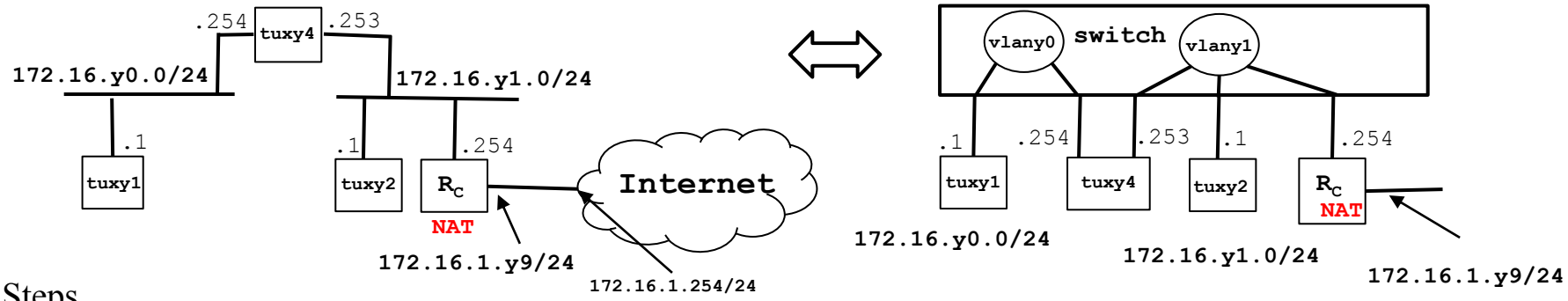
Part 2 / Exp 3 – Configure a Router in Linux



Questions

- What routes are there in the tuxes? What are their meaning?
- What information does an entry of the forwarding table contain?
- What ARP messages, and associated MAC addresses, are observed and why?
- What ICMP packets are observed and why?
- What are the IP and MAC addresses associated to ICMP packets and why?

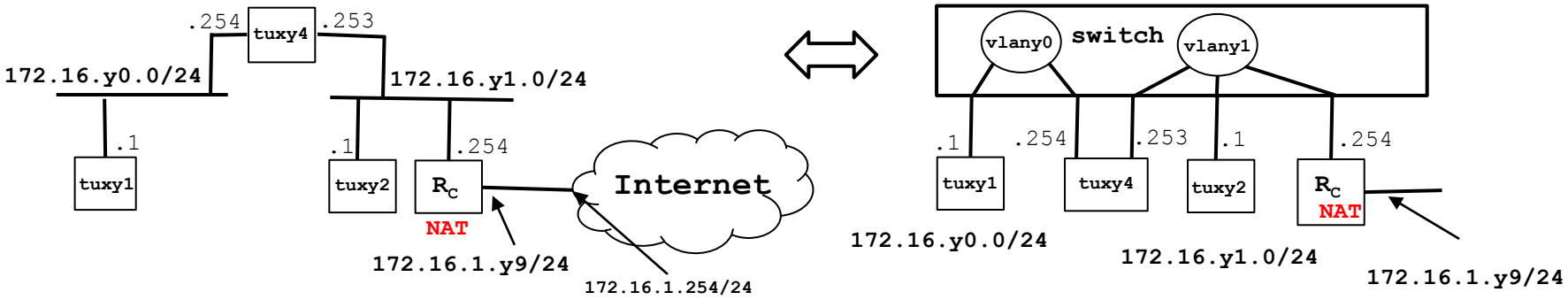
Part 2 / Exp 4 –
Configure a Commercial Router and Implement NAT



Steps

1. Configure commercial router R_C and connect it (no NAT) to the lab network (172.16.1.0/24)
2. Verify routes
   - tuxy4 as default router of tuxy1; Rc as default router for tuxy2 and tuxy4
   - Routes for 172.16.y0.0/24 in tuxy2 and Rc
3. Using ping commands and wireshark, verify if tuxy1 can ping all the network interfaces of tuxy4, tuxy2 and Rc
4. In tuxy2
   - remove the route to 172.16.y0.0/24 via tuxy4
   - In tuxy1, ping tuxy2
   - Using capture, try to understand the path followed by ICMP ECHO and ECHO-REPLY packets
   - In tuxy1, do traceroute tuxy2
   - In tuxy2, add again the route to 172.16.y0.0/24 via tuxy4
   - In tuxy1, do traceroute tuxy2
5. In tuxy1, ping the router of the lab I.321 (172.16.1.254) and try to understand what happens
6. Add NAT functionality to router Rc
7. In tuxy1 ping 172.16.1.254, verify if there is connectivity , and try to understand what happened

Part 2 / Exp 4 –
Configure a Commercial Router and Implement NAT

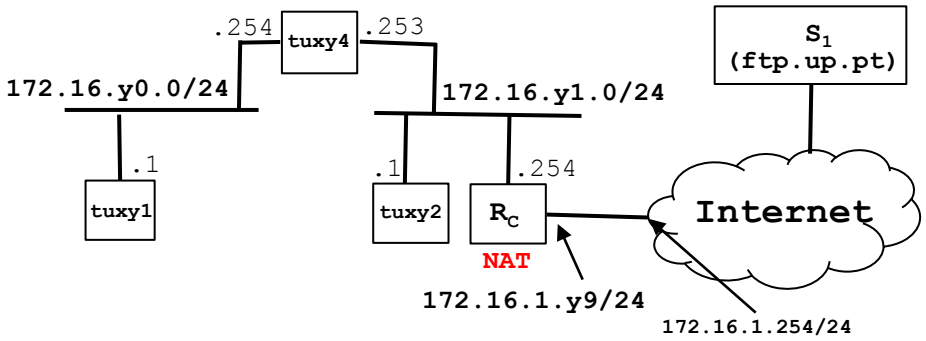


Questions

- How to configure a static route in a commercial router?
- What are the paths followed by the packets in the experiences performed and why?
- How to configure NAT in a commercial router?
- What does NAT do?

Part 2 / Exp 5 – DNS

```
tuxy1$ vi /etc/resolv.conf
        search netlab.fe.up.pt
        nameserver  172.16.1.2
```



Steps

1. Configure DNS at tuxy1, tuxy4, tuxy2; use DNS server lixa.netlab.fe.up.pt (172.16.1.2)
2. Verify if names can be used in these hosts (e.g ping hostname, use browser)
3. Execute ping new-hostname-in-the-Intenet; observe DNS related packets in Wireshark

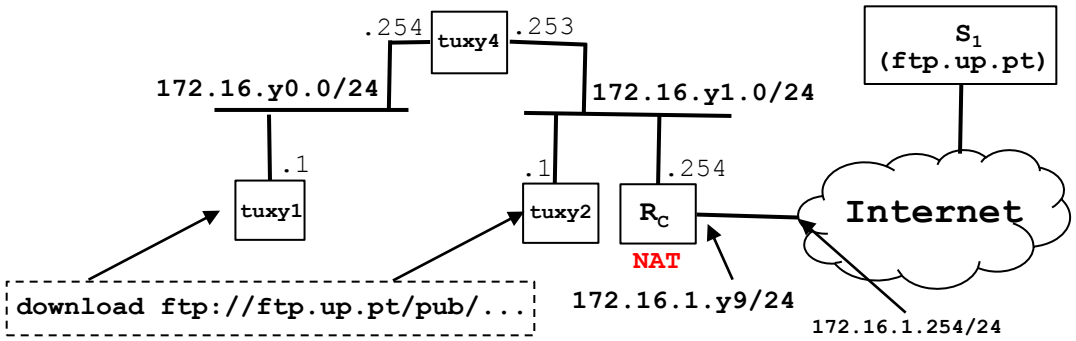
Questions

» How to configure the DNS service at an host?

» What packets are exchanged by DNS and what information is transported

Part 2 / Exp 6 – TCP connections

```
tuxy1$ vi /etc/resolv.conf
        search netlab.fe.up.pt
        nameserver  172.16.1.2
```



Steps

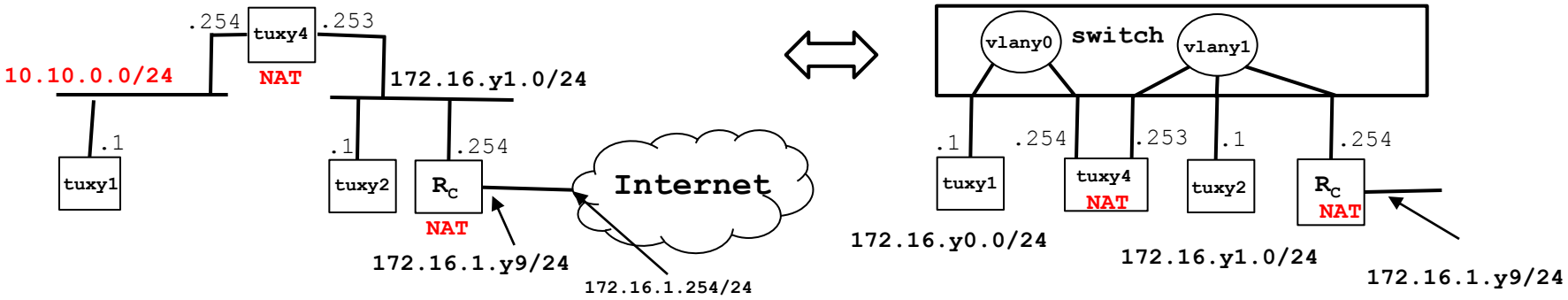
1. Compile your dowonload application in tuxy1
2. In tuxy1, restart capturing with Wireshark and run your application **download ftp://ftp.up.pt/....**
3. Verify if file has arrived correctly, stop capturing and save the log
4. Using Wireshark observe packets exchanged including:
   - TCP control and data connections, and its phases (establishment, data, termination)
   - Data transferred through the FTP control connection
   - TCP ARQ mechanism
   - TCP congestion control mechanism in action
   - Note: use also Wireshark Statistics tools (menu) to study TCP phases, ARQ and congestion control mechanism
5. Repeat download in tuxy1 but now, in the middle of the transference, start a new download in tuxy2
   - Use the Wireshark statistics tools to understand how the throughput of a TCP connection varies along the time

Part 2 / Exp 6 – TCP connections

Questions

- How many TCP connections are opened by your ftp application?
- In what connection is transported the FTP control information?
- What are the phases of a TCP connection?
- How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?
- How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according the TCP congestion control mechanism?
- Is the throughput of a TCP data connections disturbed by the appearance of a second TCP connection? How?

Optional: Part 2 / Exp 7 – Implement NAT in Linux



Optional (only if you have time; objective is to observe NAT)

- » Implement the NAT functionality in tuxy4, using iptables.
- » Use Wireshark on both interfaces of tuxy4
- » In tuxy1, generate traffic for the Internet. Use different types of traffic

  TCP (e.g. wget), UDP (e.g. traceroute), and ICMP traffic (e.g. ping)
- » Observe the translation of IP and port addresses

```
tuxy3 iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
tuxy3$ iptables -A FORWARD -i eth0 -m state --state NEW,INVALID -j DROP
tuxy3$ iptables -L
tuxy3$ iptables -t nat -L
```

Worplan

- 1ª aula - Part 1
  - » experiments using Telnet, focus on FTP
  - » usage of gethostbyname and socket functions
  - » client architecture; main use case
- 2ª aula - Part 2: Steps 1 e 2
  - » linux, lab, cables, ipconfig, route, arp, wireshark, capture and log analysis
- Aulas seguintes - Part 2: Steps 2, 3, 4, 5 e 6
- Last week
  - » Demonstration of download application
  - » Delivery of report (use moodle for upload)

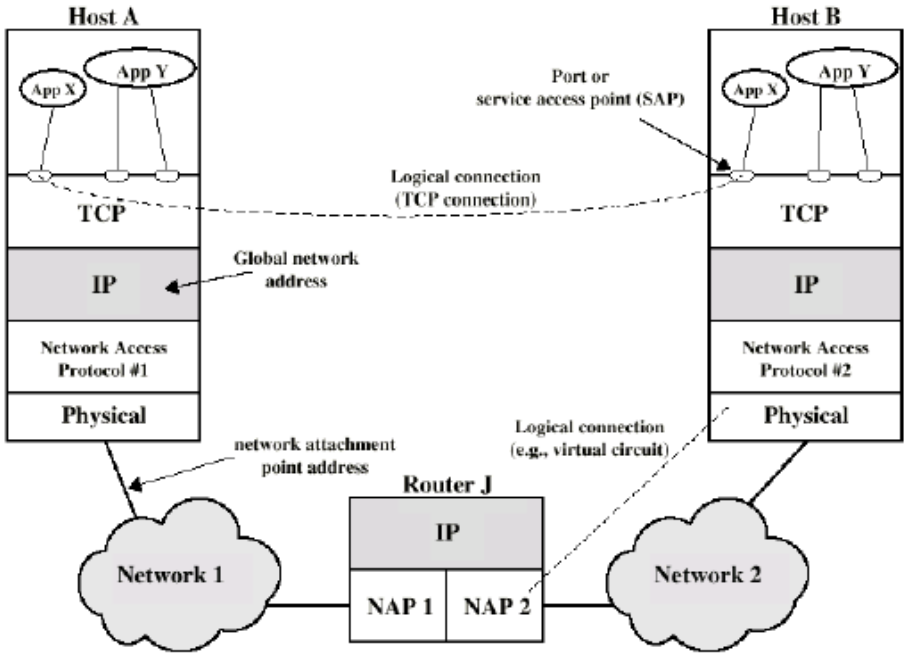
Final Report

- Report must contain
  - » Title, Authors, Summary
  - » Introduction
  - » Part 1 - Download application
    - – Architecture of the download application
    - – Report of a successful download
  - » Part 2 – Network configuration and analysis
    - – For each experiment (1 to 7)
      - ◆ Network architecture, experiment objectives, main configuration commands
      - ◆ **Analysis of the logs captured that are relevant for the learning objectives**
  - » Conclusions
  - » References
  - » Annexes: code of the download application, configuration commands, logs captured
- Maximum length of 8 pages A4, font 11pt
- Upload through moodle

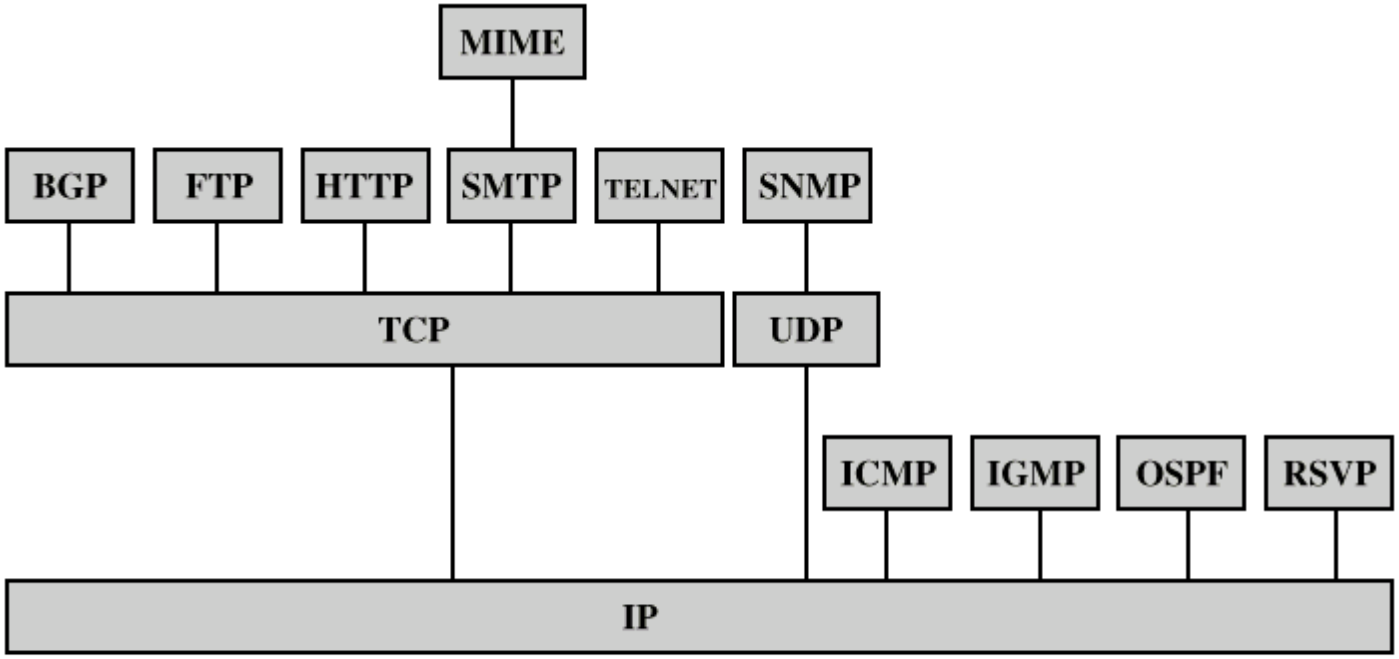
TCP/IP and Application Protocols

Algumas Características do TCP/IP

- O IP (*Internet Protocol*) é implementado em todos os computadores (*hosts*) e *routers*
- Cada computador tem um endereço IP único em cada subrede a que pertence
- Cada processo num computador tem um endereço único (porta)



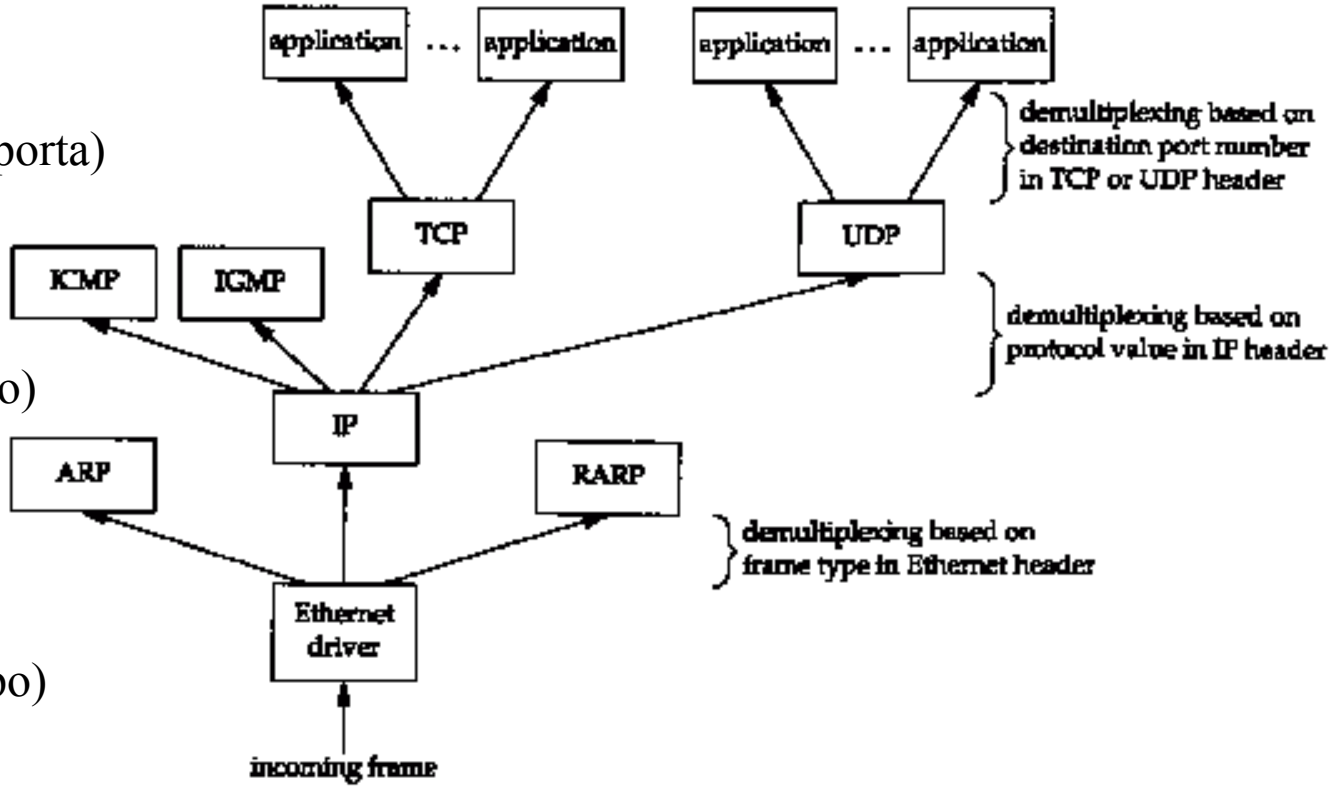
Protocolos TCP/IP



BGP = Border Gateway Protocol
FTP = File Transfer Protocol
HTTP = Hypertext Transfer Protocol
ICMP = Internet Control Message Protocol
IGMP = Internet Group Management Protocol
IP = Internet Protocol
MIME = Multi-Purpose Internet Mail Extension
OSPF = Open Shortest Path First
RSVP = Resource ReSerVation Protocol
SMTP = Simple Mail Transfer Protocol
SNMP = Simple Network Management Protocol
TCP = Transmission Control Protocol
UDP = User Datagram Protocol

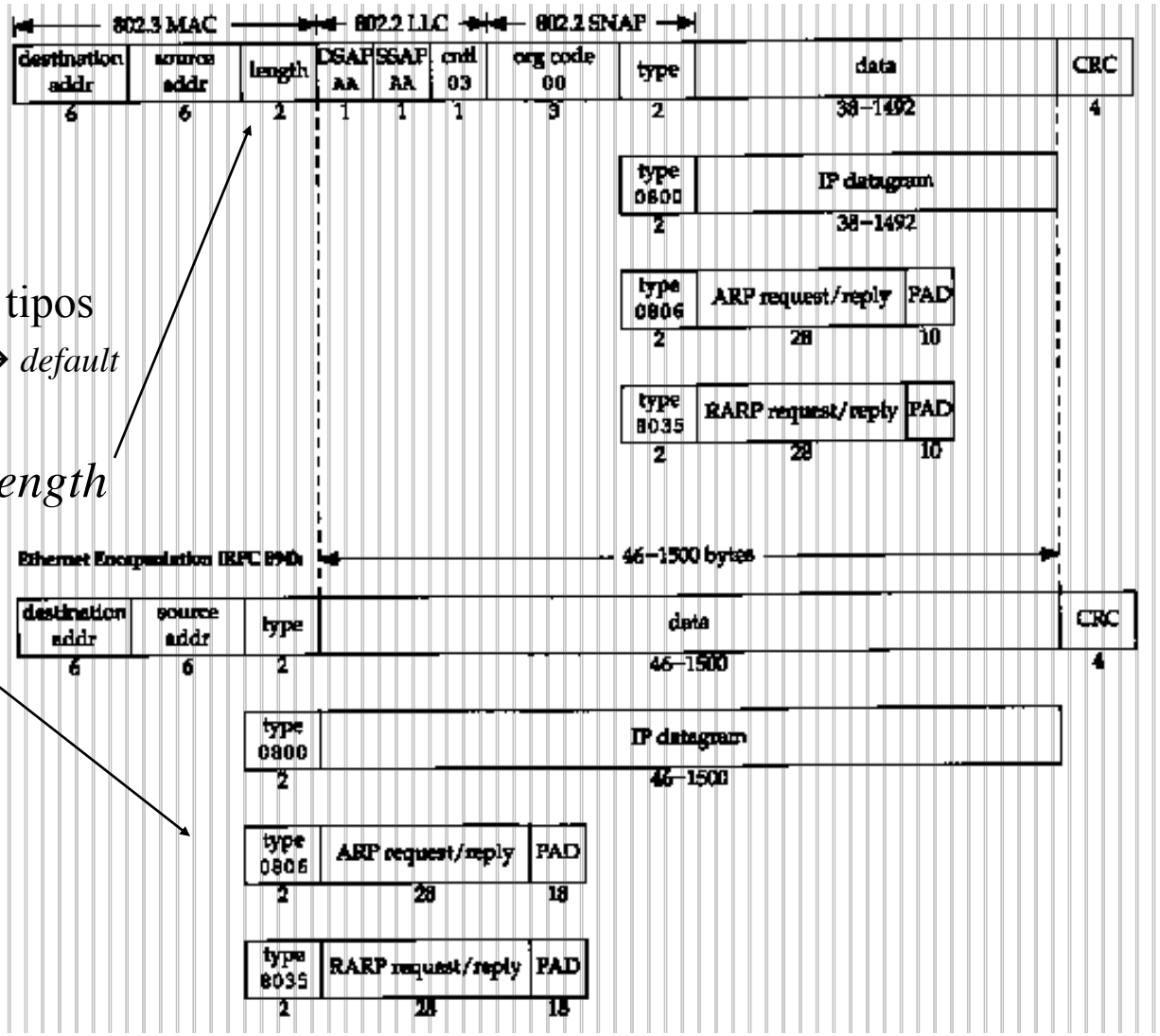
Desmultiplexagem

- Cabeçalho TCP/UDP (porta)
  - FTP → 21
  - Telnet → 23
  - ...
- Cabeçalho IP (protocolo)
  - ICMP → 1
  - IGMP → 2
  - TCP → 6
  - UDP → 17
- Cabeçalho Ethernet (tipo)
  - IP → 0x0800
  - ARP → 0x0806
  - RARP → 0x8053

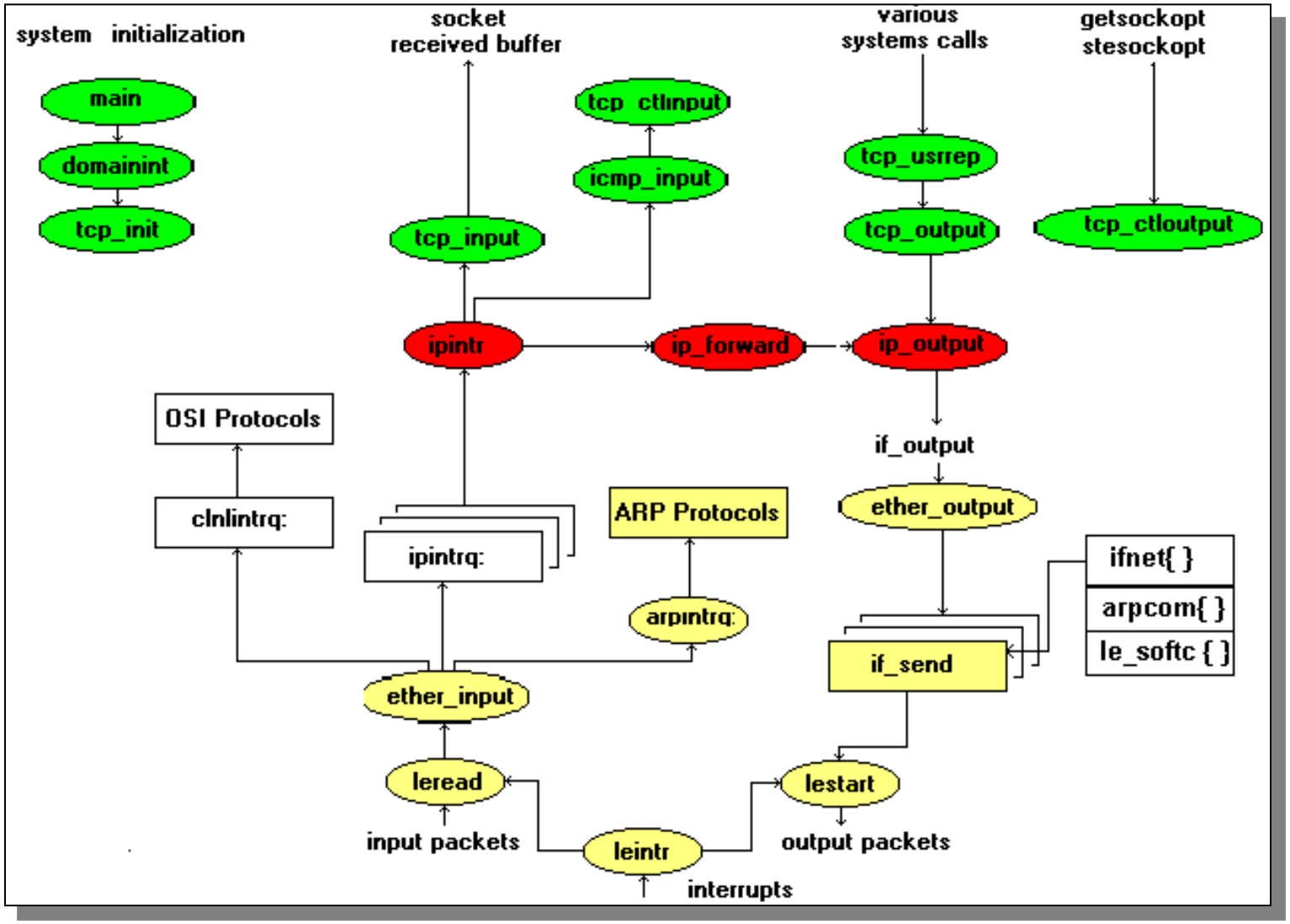


Encapsulamento Ethernet

- Cartas Ethernet
  - » Devem receber
    - encapsulamento IEEE 802
    - encapsulamento Ethernet
  - » Se conseguem enviar os 2 tipos
    - encapsulamento Ethernet → *default*
- Valores validos IEEE 802 *length*
  - » Diferentes de *type* válidos
    - Ex. 0800 = 2048

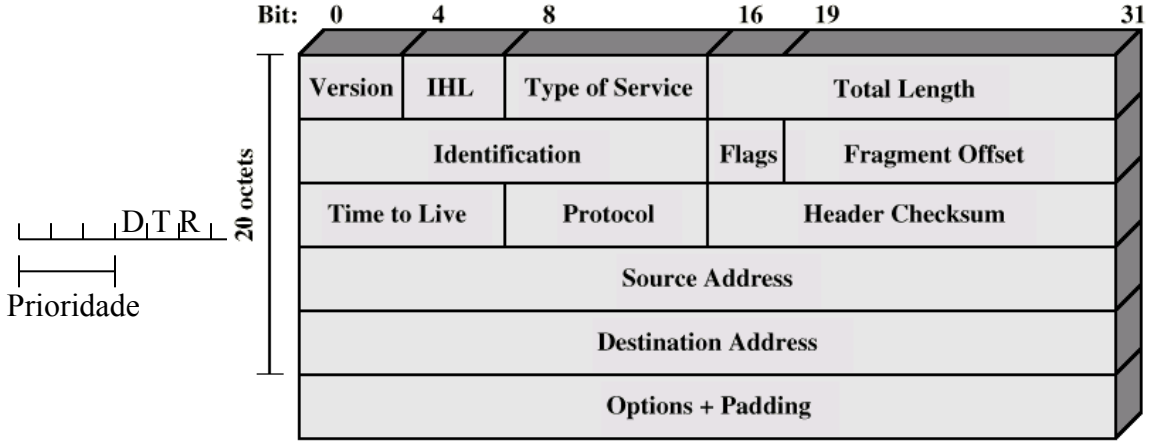


Funcões da Pulha TCP/IP



Protocolo IP

- **Version** - versão do protocolo (v4)
- **IHL** - comprimento do cabeçalho (em palavras de 32 bits); 20..60 octetos
- **Type of Service** - tipo de serviço a fornecer pela rede
- **Total Length** - comprimento total do datagrama (máx. 65535 octetos)
- **Identification** - identificador comum a todos os fragmentos de um datagrama
- **DF** - *Don't Fragment*
- **MF** - *More Fragments*
- **Fragment Offset**
- **Time To Live (TTL)** - limita a vida de um pacote; decrementado de cada vez que passa por um *router*; quando chega a 0 o pacote é eliminado

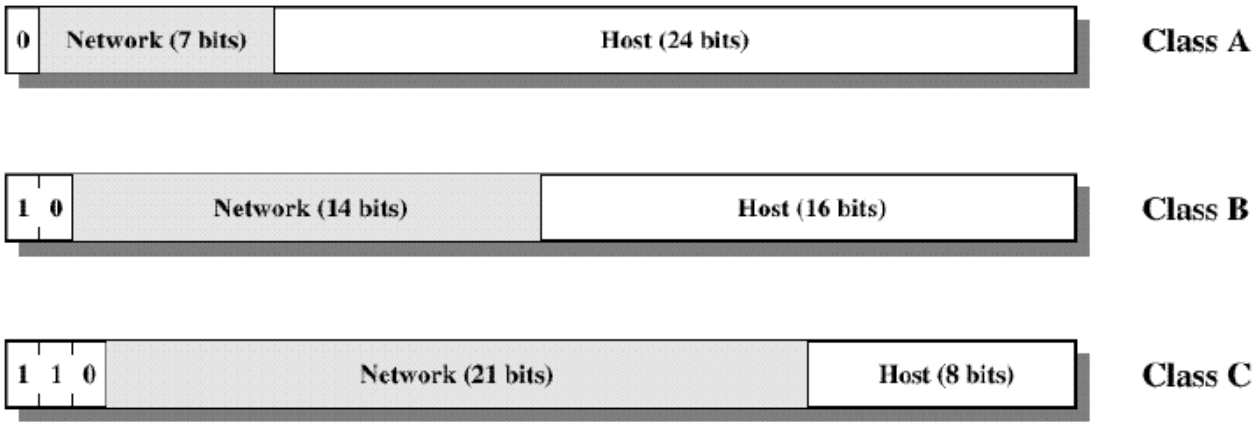


- Protocol - protocolo da camada de transporte encapsulado (ex. TCP, UDP)
- Source Address - endereço do emissor
- Destination address - endreço do destinatário
- Options - 1 octeto identifica a opção; 1 octeto contém o comprimento (opcional); ex.: Record Route

IP - Endereços

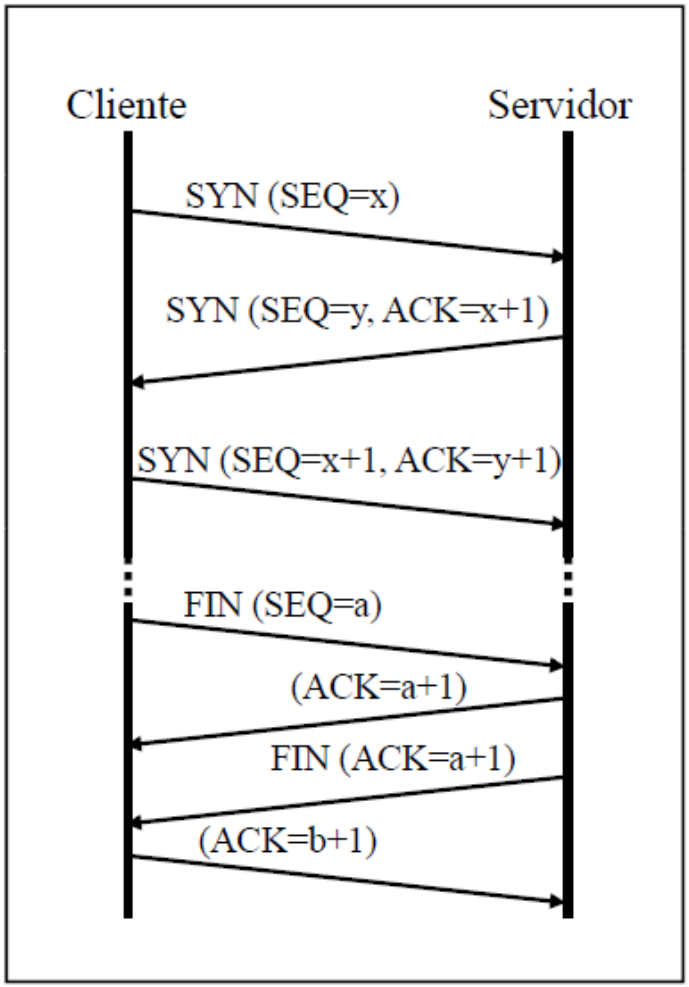
- Endereços globais de 32 bits, estruturados em duas partes: rede (*netid*) e *host* (*hostid*)
  - » Originalmente os endereços eram baseados em classes (A, B, C, D, E)
    - – Prefixo de rede de comprimento fixo
  - » Endereços sem classes (CIDR)
    - – Prefixo de rede de comprimento variável

Classe	*Valores*
A	0.0.0.0 → 127.255.255.255
B	128.0.0.0 → 191.255.255.255
C	192.0.0.0 → 223.255.255.255
D	224.0.0.0 → 239.255.255.255
E	240.0.0.0 → 247.255.255.255



TCP – Transmission Control Protocol

- RFC 793
- Características
  - » Assegura um fluxo de octetos extremo a extremo, fiável, sobre um suporte não fiável
  - » Protocolo orientado às conexões
  - » Conexões *full-duplex*
  - » Confirmação positiva (ACK)
  - » Recupera de perdas e erros (retransmissões) após *time-out*
  - » Entrega ordenada dos dados à aplicação
  - » Controlo de fluxo e de congestionamento
  - » Multiplexagem de várias conexões TCP sobre o mesmo endereço IP
- Estabelecimento de conexão TCP
  - » *3 way handshake*
  - » Modelo cliente-servidor



TCP

Source Port - porto de origem

Destination Port - porto do destino

Sequence Number - identifica, no fluxo do emissor, a sequência de octetos enviada

Acknowledgement Number - corresponde ao número do octeto que se espera de receber

HLEN - o comprimento do cabeçalho TCP (em palavras de 32 bits)

URG - informa se o campo Urgent Pointer deve ser interpretado

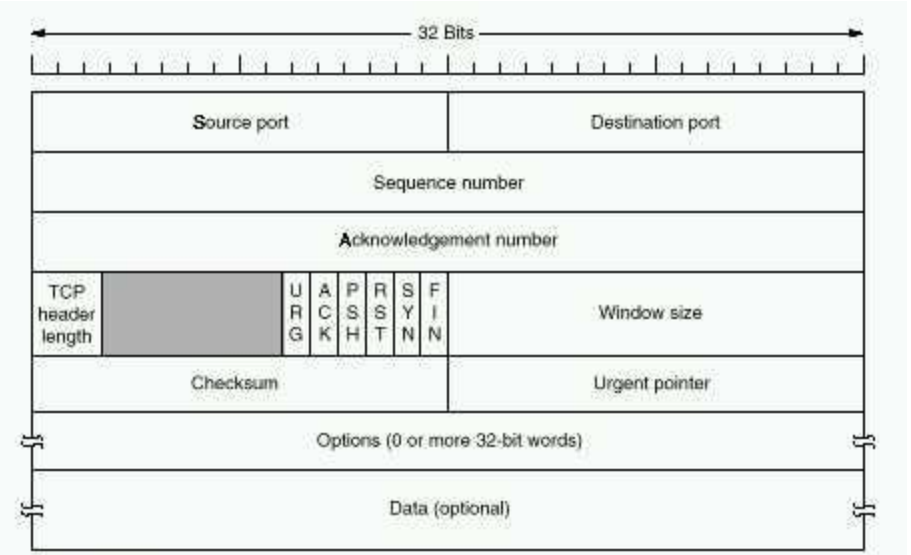
ACK - informa se o campo Ack. Nr é válido

PSH - permite inactivar a *bufferização*

RST - usado para a reinicializar uma ligação

SYN – permite estabeler uma ligação

FIN - permite a terminar uma ligação



The TCP header.

Window Size - número de bytes que o par da comunicação pode enviar sem confirmação (controlo de fluxo)

Checksum - abrange o cabeçalho, os dados e o pseudo-cabeçalho

Berkeley Sockets

- API - Application Programming Interface
  - » sistema operativo: UNIX
  - » linguagem de programação: C
  - » protocolos de comunicação
    - TCP/IP
    - UNIX
    - XNS
  - » Estruturas de dados de endereços
  - » Primitivas: socket(), bind(), connect(), listen(), accept(), recvfrom(), sendto(), close()
  - » Associação - par de *sockets*

Berkeley Sockets

- Estruturas de dados de endereços
  - » BSD

```
<sys/socket.h>
struct sockaddr {
    u_short   sa_family;   /*Address family - ex: AF_INET*/
    char      sa_data[14]; /*Protocol address*/
};
```

  - » Internet

```
<netinet/in.h>
struct in_addr {
    u_long        s_addr;
};
struct sockaddr_in {
    short     sin_family;        /*AF_INET*/
    u_short   sin_port;          /*Port number*/
    struct    in_addr sin_addr;  /*32 bit netid/hosdtid*/
    char      sin_zero[8];       /*unused*/
};
```

Berkeley Sockets

➔ `int socket(int family, int type, int protocol)`

family: AF_INET, AF_UNIX
type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
protocol: protocolo a usar (com o valor 0 é determinado pelo sistema)

» Retorno
  – descritor de *socket*
  – -1, em caso de erro

➔ `int bind(int sockfd, struct sockaddr* myaddr, int addrlen)`

sockfd: descritor do *socket*
myaddr: endereço local (IP + porto)
addrlen: comprimento da estrutura myaddr

» Retorno
  – 0 em caso de sucesso
  – -1 em caso de erro

» Esta primitiva associa o *socket* ao endereço local myaddr

Berkeley Sockets

➔ `int connect(int sockfd, struct sockaddr* serveraddr, int addrlen)`

serveraddr: endereço do servidor remoto (IP + porto)

» Retorno
  – 0 em caso de sucesso
  – -1 em caso de erro

» TCP: estabelecimento de ligação com servidor remoto

» UDP: armazenamento do endereço *serveraddr*

➔ `int listen(int sockfd, int backlog)`

backlog: número de pedidos de ligação em fila de espera

» Retorno
  – 0 em caso de sucesso
  –-1 em caso de erro

» Primitiva especifica o número máximo de ligações em fila de espera

Berkeley Sockets

```
➔ int accept(int sockfd, struct sockaddr* peeraddr, int* addrlen)
```

peeraddr: estrutura usada para armazenar o endereço do cliente (IP + porto)

addrlen: apontador para o comprimento da estrutura peeraddr

» Retorno
  – descritor do *socket* aceite, endereço do cliente e respectivo comprimento
  – -1 em caso de erro

» Primitiva atende pedido de ligação e cria outro *socket* com as mesmas propriedades que o sockfd

```
➔ int send(int sockfd, const void* buf, int len, unsigned int flags)
➔ int recv(int sockfd, void* buf, int len, unsigned int flags)
```

buf: apontador para a posição de memória que contém/vai conter os dados

flags: MSG_OOB, MSG_PEEK, MSG_DONTROUTE

»Retorno
  – número de octetos escritos/lidos
  – 0 em caso de a ligação ter sido fechada
  –-1 em caso de erro

»Estas primitivas permitem o envio e a recepção de dados da rede

Berkeley Sockets

```
➔ int sendto(int sockfd, const void* buf, int len,
         unsigned int flags,
         struct sockaddr* to, int tolen)

➔ int recvfrom(int sockfd, void* buf, int len,
           unsigned int flags,
           struct sockaddr* from, int* fromlen)
```

» to: endereço do destinatário do pacote

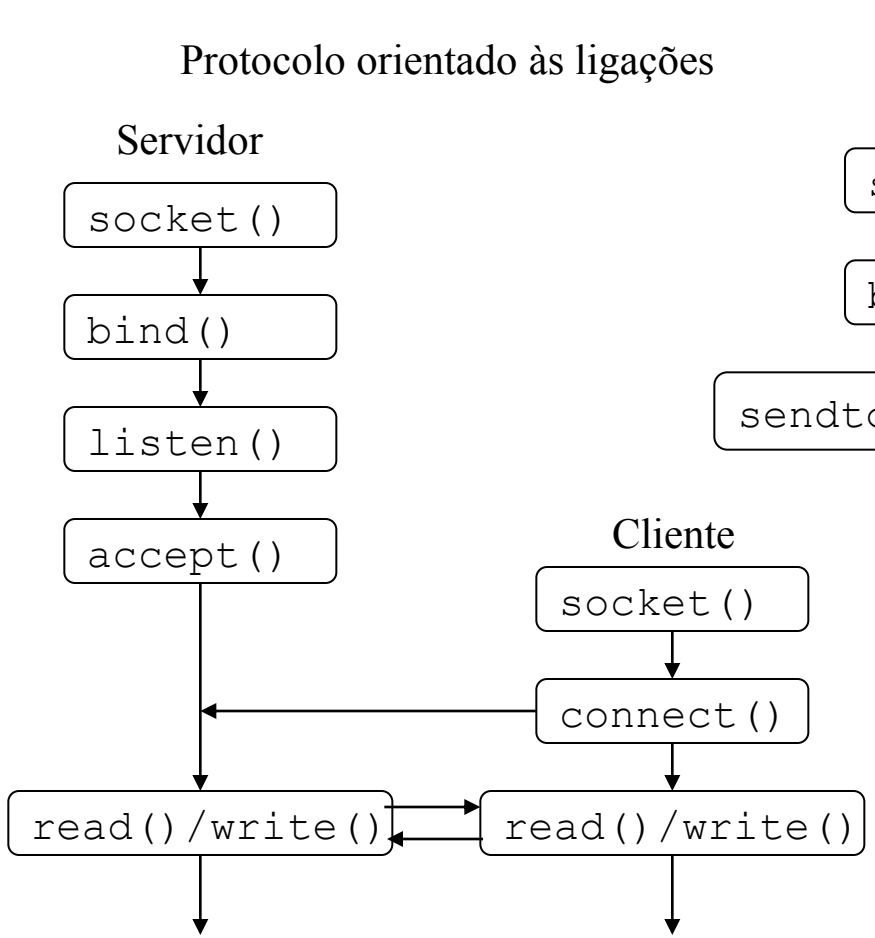
» from: endereço do emissor presente no pacote recebido

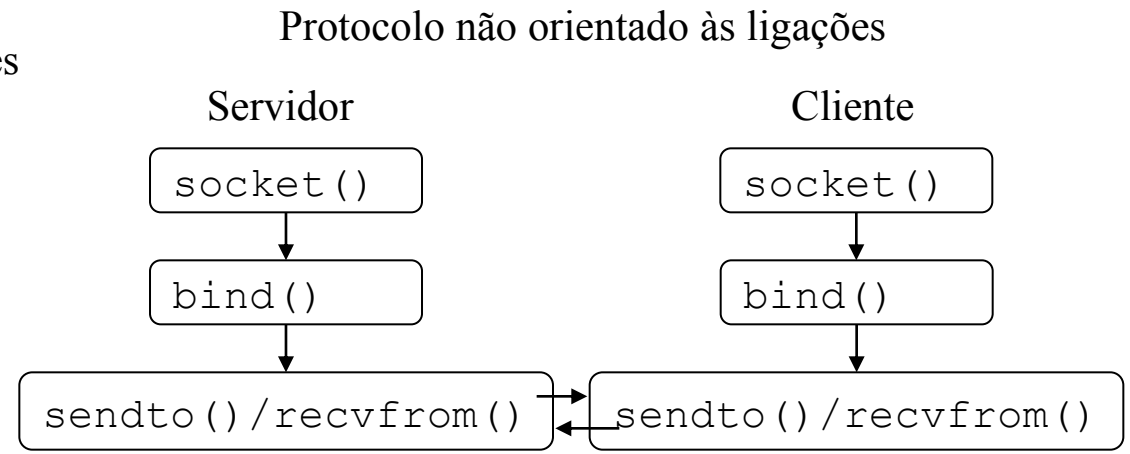
» estas primitivas são semelhantes ao send()/recv() mas permitem adicionalmente o envio de mensagens em cenários *connectionless* (UDP), sem haver portanto estabelecimento de ligação

```
➔ int close(int sockfd)
```

» esta primitiva é usada para fechar o socket

Berkeley Sockets





Nota: o cliente de uma ligação TCP pode chamar a primitiva *bind()* antes de estabelecer a ligação

Berkeley Sockets

- Ordenamento dos octetos
  - » varia com a arquitectura (ex: Intel é *little endian*, Motorola é *big endian*)
    - – Little endian → little end firt; Big endian → big end first
  - » *network byte order* → Big endian
  - » primitivas de conversão (long - 32 bits, short - 16 bits):

→ **`u_long htonl(u_long hostlong)`**

→ **`u_short htons(u_short hostshort)`**

→ **`u_long ntohl(u_long netlong)`**

→ **`u_short ntohs(u_short netshort)`**

- Conversão entre formatos de endereços
  - » *dotted decimal notation* para endereço Internet de 32 bits com ordenamento de rede

→ **`unsigned long inet_addr(char * cp)`**

  - » endereço Internet de 32 bits com ordenamento de rede para *dotted decimal notation*

→ **`char* inet_ntoa(struct in_addr in)`**

Berkeley Sockets

- Opções dos *sockets*
  - **`setsockopt()`**
  - **`getsockopt()`**
  - **`fcntl()`**
  - **`ioctl()`**
- Entrada/Saída assíncronas
  - » utilização de sinais
- Multiplexagem de Entradas/Saídas
  - » rotina **`select()`**

- *Domain Name Service*
  - » permite a obtenção do endereço de uma máquina a partir do nome

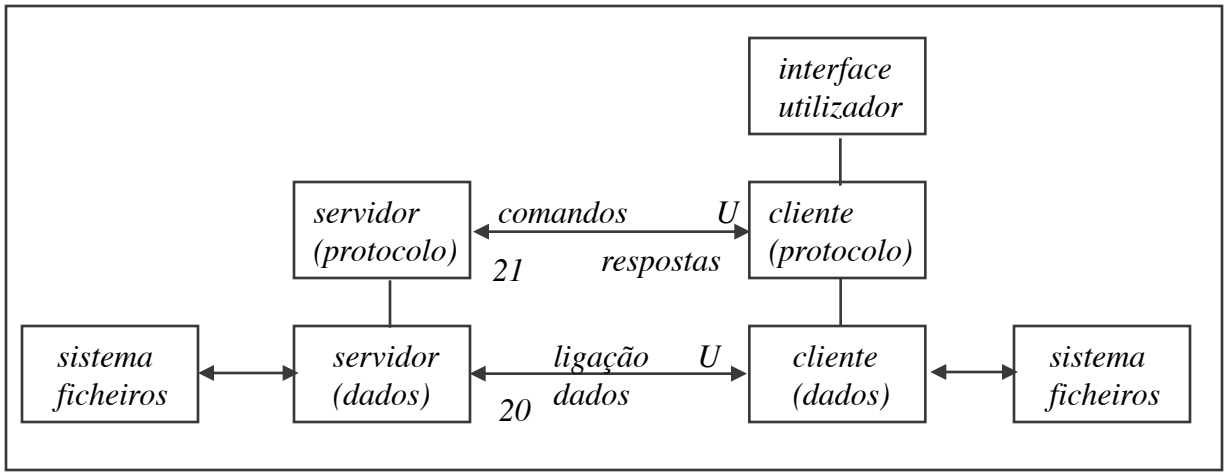
```
struct hostent*
gethostbyname (const char* name);

struct hostent{
      char*  hname;       /*nome oficial*/
      char** haliases;
      int    h_addrtype; /*AF_INET*/
      int    h_length;
      char** h_addr_list;
};

#define h_addr h_addr_list[0]
```

FTP - File Transfer Protocol

- FTP - File Transfer Protocol
  - transferência de ficheiros entre computadores (ASCII e binário)
- Modelo de Comunicação Cliente-Servidor
  - ligações TCP independentes para controlo da ligação e transferência de dados
  - RFC 959



FTP - Exemplo

```
LOCAL COMMANDS BY USER              ACTION INVOLVED

ftp (host) multics<CR>              Connect to host S, port L,
                                    establishing control connections.
                                    <---- 220 Service ready <CRLF>.
username Doe <CR>                   USER Doe<CRLF>---->
                                    <---- 331 User name ok,
                                          need password<CRLF>.
password mumble <CR>                PASS mumble<CRLF>---->
                                    <---- 230 User logged in<CRLF>.
retrieve (local type) ASCII<CR>
(local pathname) test 1 <CR>        User-FTP opens local file in ASCII.
(for. pathname) test.pl1<CR>        RETR test.pl1<CRLF> ---->
                                    <---- 150 File status okay;
                                          about to open data
                                          connection<CRLF>.
                                    Server makes data connection
                                    to port U.

                                    <---- 226 Closing data connection,
                                        file transfer successful<CRLF>.
type Image<CR>                      TYPE I<CRLF> ---->
                                    <---- 200 Command OK<CRLF>
store (local type) image<CR>
(local pathname) file dump<CR>      User-FTP opens local file in Image.
(for.pathname) >udd>cn>fd<CR>       STOR >udd>cn>fd<CRLF> ---->
                                    <---- 550 Access denied<CRLF>
terminate                           QUIT <CRLF> ---->
                                    Server closes all
                                    connections.
```

Network Configuration Examples

Configurações de Rede em Linux

- Re-inicialização do subsistema de comunicação
  - » /etc/init.d/networking restart
- Configuração tuxxy
  - » activar interface eth0
    - – root# ifconfig eth0 up
  - » listar configurações actuais das interfaces de rede
    - – root# ifconfig
  - » configurar eth0 com endereço 192.168.0.1 e máscara 16 bits
    - – root# ifconfig eth0 192.168.0.1/16
  - » adicionar rota para subrede
    - – root# route add -net 192.168.1.0/24 gw 172.16.4.254
  - » adicionar rota default
    - – root# route add default gw 192.168.1.1
  - » listar rotas actuais
    - – root# route -n
  - » echo 1 > /proc/sys/net/ipv4/ip_forward
  - » echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_request

Reboot do switch

- Ligação ao switch
  - » Porta série /dev/ttyS0 em tuxy3, gtkterm
  - » Por telnet ou ssh a 172.16.1.x0
  - » Username – xxxxxx
  - » Password – xxxxxx
- Na shell de comandos
  - » copy tftp://192.168.109.1/2 startup-config
  - » delete flash:vlan.dat
  - » reload

Handling VLANs in Cisco Switch – Cap. 12

- Cap. 12 – Configuring VLANs
- Creating an Ethernet VLAN
  - » configure terminal
  - » vlan x0
  - » end
  - » show vlan id x0
- Deleting a vlan
  - » configure terminal
  - » no vlan x0
  - » end
  - » show vlan brief
- Add port 1 to vlan x0
  - » configure terminal
  - » interface fastethernet 0/1
  - » switchport mode access
  - » switchport access vlan x0
  - » end
  - » show running-config interface fastethernet 0/1
  - » show interfaces fastethernet 0/1 switchport

Configuração do Router

- Interface de rede
  - » interface gigabitethernet 0/0
  - » ip address 192.168.12.2 255.255.255.0
  - » no shutdown
  - » exit
  - » show interface gigabitethernet 0/0
- Rotas
  - » estáticas
    - – **ip route** *prefix mask* {*ip-address* | *interface-type interface-number* [*ip-address*]}
  - » Dinâmicas
    - – **configure terminal**
    - – router rip
    - – version 2
    - – network 192.168.1.1
    - – network 10.10.7.1
    - – no auto-summary
    - – end
    - – **show ip route**

Configuração do Router Cisco com NAT

- Cisco NAT
  http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.shtml

```
conf t
interface gigabitethernet 0/0
ip address 172.16.11.254 255.255.255.0
no shutdown
ip nat inside
exit

interface gigabitethernet 0/1
ip address 172.16.1.19 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.1.19 172.16.1.19 prefix 24
ip nat inside source list 1 pool ovrld overload

access-list 1 permit 172.16.10.0 0.0.0.7
access-list 1 permit 172.16.11.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.10.0 255.255.255.0 172.16.11.253
end
```

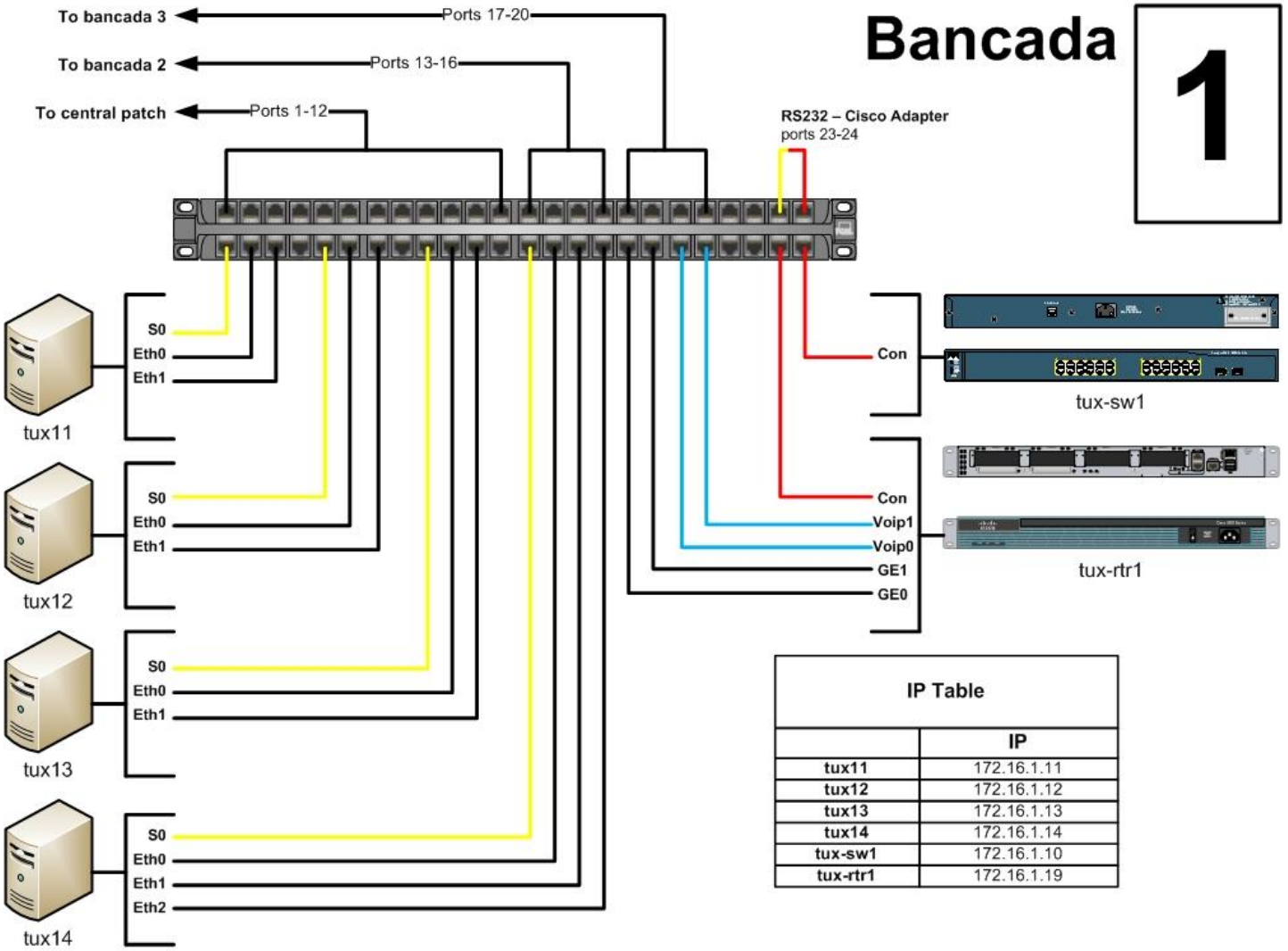
Generic Configurations

- Limpar configurações; y=bancada
  - » SWITCH
    - – del flash:vlan.dat
    - – copy flash:tuxy-clean startup-config
    - – reload
  - » ROUTER
    - – copy flash:tuxy-clean startup-config
    - – reload
  - » TUX
    - – updateimage
    - – /etc/init.d/networking restart
- Changing tux
  - » N - tux number (1..4)
  - » [SCROLL LOCK] [SCROLL LOCK] [N] [ENTER]

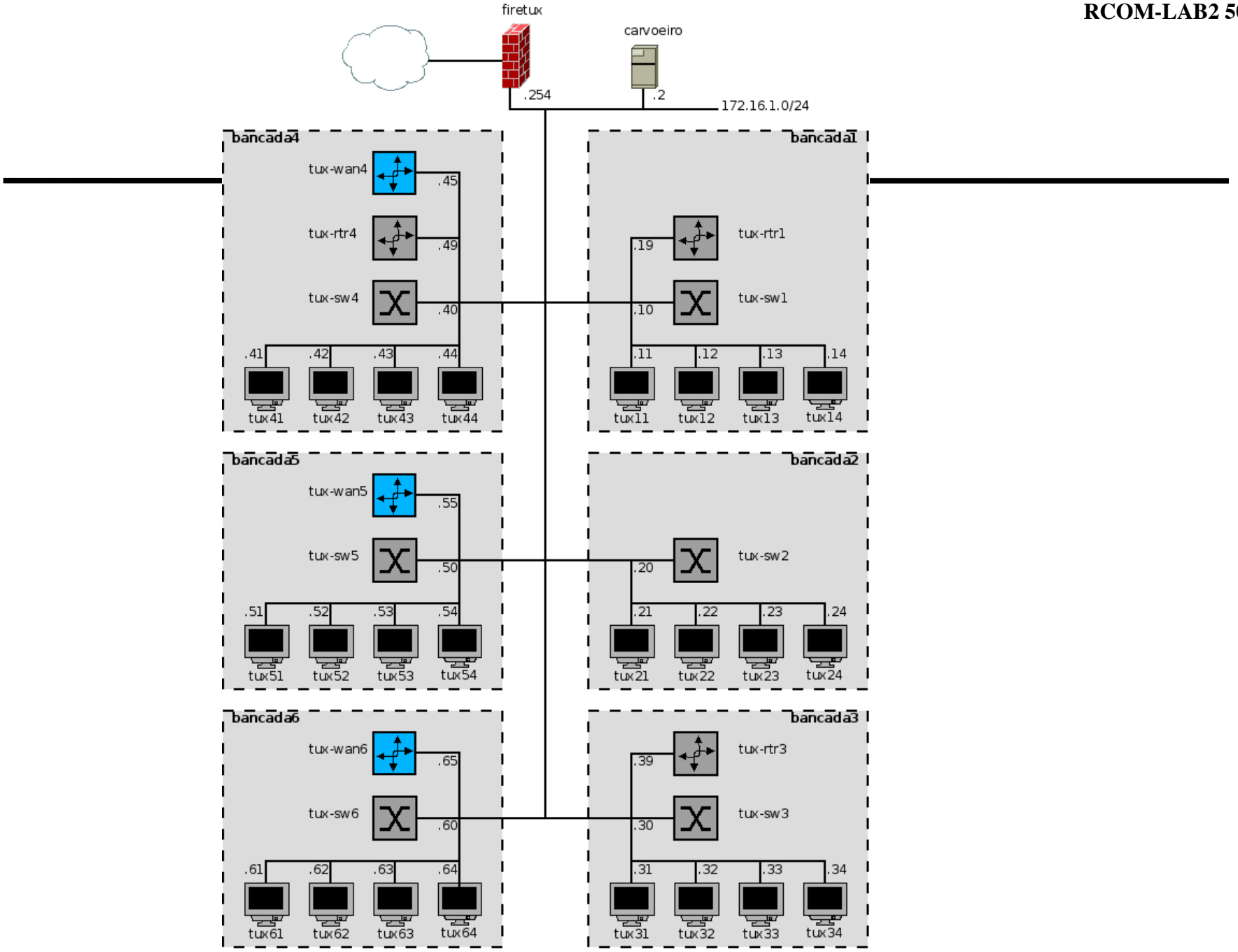
Login

- switch
  - » enable
  - » password: ******
- router
  - » username: root
  - » passwrod: ******
- tux
  - » username: root
  - » passwrod: **********
- re-iniciar net em tux
  - » /etc/init.d/networking restart
- Enabling forwarding in tuxes
  - » echo 1> /proc/sys/net/ipv4/ip_forward
- Enabling echo-reply to broadcast request
  - » echo 0> /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

Connections in Bancada 1



IP Table	
	IP
tux11	172.16.1.11
tux12	172.16.1.12
tux13	172.16.1.13
tux14	172.16.1.14
tux-sw1	172.16.1.10
tux-rtr1	172.16.1.19


firetux
carvoeiro
.254
.2
172.16.1.0/24
bancada4
tux-wan4
.45
tux-rtr4
.49
tux-sw4
.40
.41
.42
.43
.44
tux41
tux42
tux43
tux44
bancada1
tux-rtr1
.19
tux-sw1
.10
.11
.12
.13
.14
tux11
tux12
tux13
tux14
bancada5
tux-wan5
.55
tux-sw5
.50
.51
.52
.53
.54
tux51
tux52
tux53
tux54
bancada2
tux-sw2
.20
.21
.22
.23
.24
tux21
tux22
tux23
tux24
bancada6
tux-wan6
.65
tux-sw6
.60
.61
.62
.63
.64
tux61
tux62
tux63
tux64
bancada3
tux-rtr3
.39
tux-sw3
.30
.31
.32
.33
.34
tux31
tux32
tux33
tux34