Sistemi Operativi e Reti di Calcolatori (SOReCa)

Corso di Laurea in *Ingegneria Informatica e Automatica (BIAR)*
Terzo Anno | Primo Semestre
A.A. 2024/2025

Esercitazione [10] UDP: Semplici DNS server e VPN “from scratch”

Riccardo Lazzeretti lazzeretti@diag.uniroma1.it
Paolo Ottolino paolo.ottolino@uniroma1.it
Alessio Izzillo izzillo@diag.uniroma1.it

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA UNIVERSITÀ DI ROMA

Sommario

- Soluzioni precedente esercitazione

- Simple DNS Server:
  - Note storiche: Paul Mockapetris
  - RFC 1035: parziale (solo A e AAA)

- Lab 10, es. 1: simple DNS server
  - Ascolto:
  - Risposta:
- Simple VPN Server:
  - Le reti private virtuali
  - RFC 2764: parziale (solo tunneling)

- Lab 10, es. 2: simple VPN server
  - Ascolto:
  - Risposta:

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

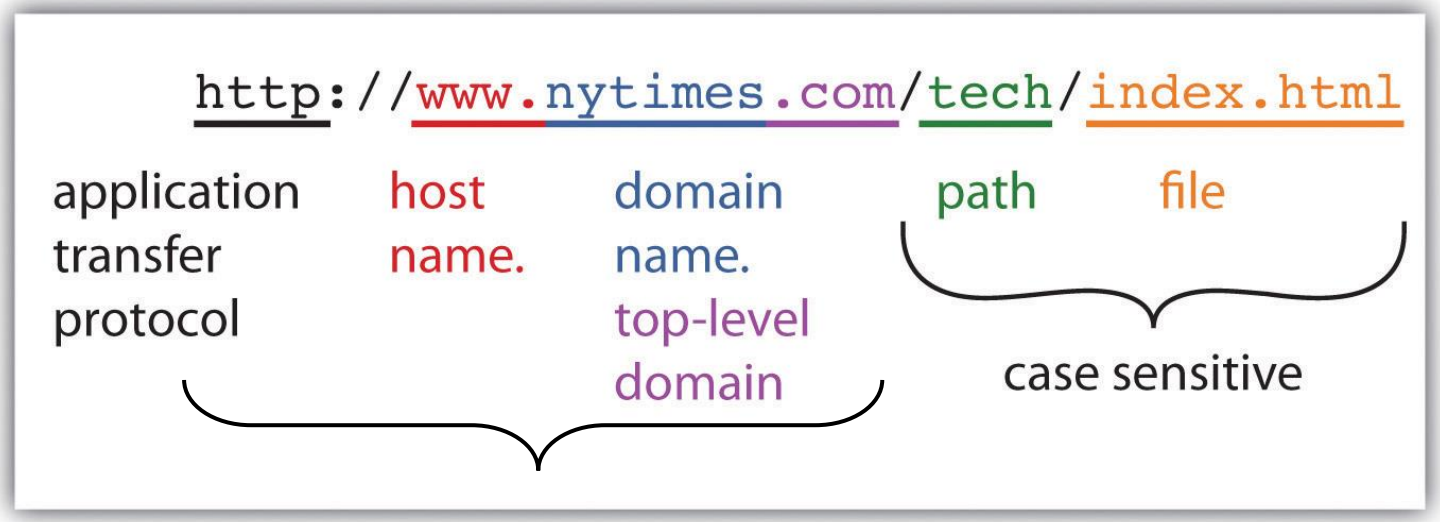
Simple DNS server

Note Storiche: file hosts , Hostname Resolution, Zone of Authority
DNS: RFC 882, 883 (superseeded by 1034 – 1035, nel 1987)

DNS: Domain Name System

Anatomia di un indirizzo Web

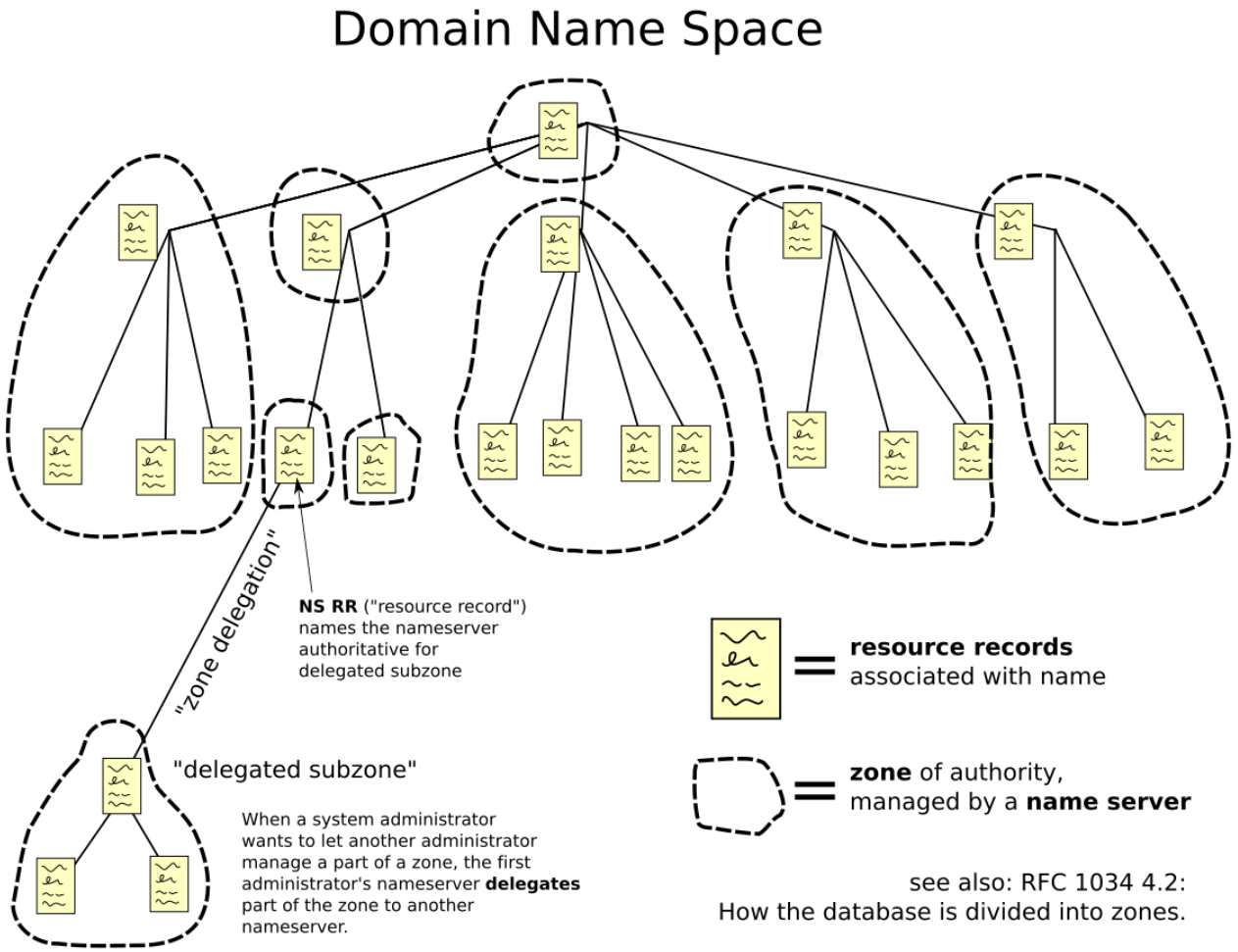
Ognuno dei componenti della URL deve essere tradotto in un elemento comprensibile da Unix.



Item	Unix understandable	Traduzione
`application transfer protocol`	numero di porta (`sin_port = htons(PORT)`)	IANA Service Name Port Number
`host-name, domain-name, top-level-domain`	indirizzo IPv4 (`sin_addr.s_addr = INADDR_ANY`)	Effettuata dal DNS: le pagine gialle degli indirizzi IP
`path, filename`	`file_fd = open(file_name, O_RDONLY);`	Effettuata dall' Applicazione (vedi lab09)

DNS: Domain Name System

Indirizzamento IP e Nomi Mnemonici: automatismo 1/3

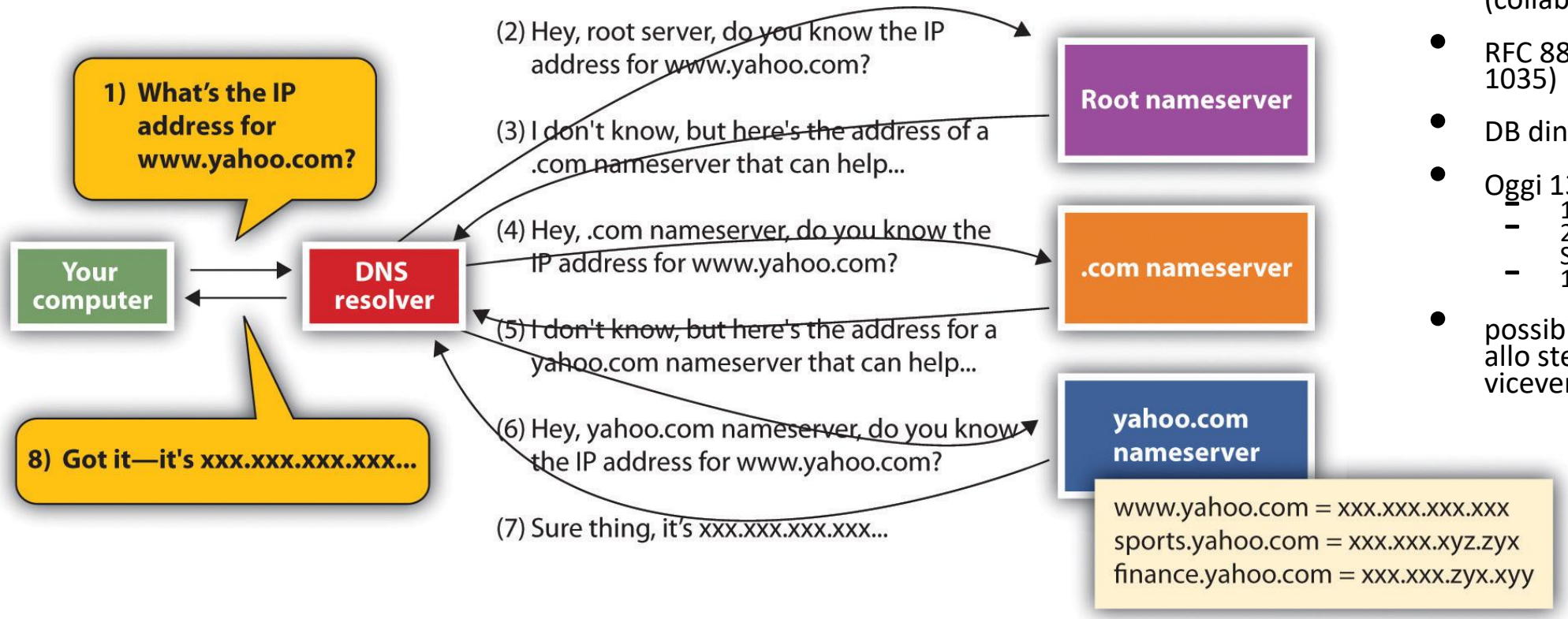


Metodo coerente per **referenziare** le specifiche **risorse** particolari sparse in Internet. Si compone di 3 elementi concettuali:

- **DOMAIN NAME SPACE**: specifica ad **albero** dello spazio dei nomi.➔ **Divide et Impera**: gerarchia di **Zone of Authority**
- **NAME SERVER**: programma server eseguito sui nodi ➔ **Automatismo Server**: detiene le informazioni su **Zone** o **Subzone** (insieme di **Resource Record**) e le fornisce, **rispondendo alle query**.
- **RESOLVER**: programma client eseguito su ogni host ➔ **Automatismo Client**: estrae le informazioni dai name server, **effettuando le query**. (es. `dig`: domain information grouper)

DNS: Domain Name System

Indirizzamento IP e Nomi Mnemonici: automatismo 2/3

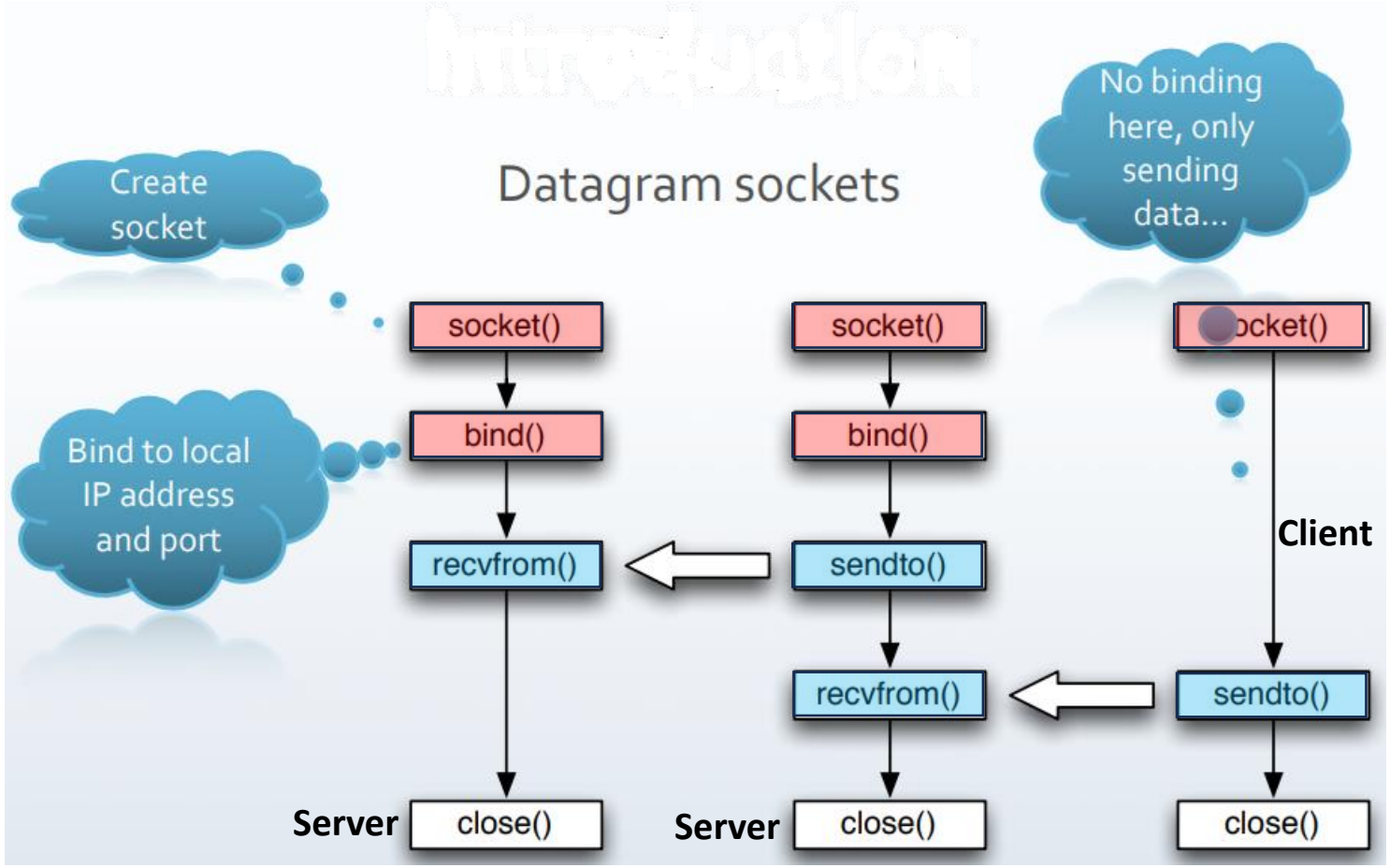


- Primo prototipo DNS nel **1983**, a cura di **Paul Mpckapetris** (collaborazione di Jon Postel)
- RFC 882 e 883 (poi 1034 e 1035)
- DB dinamico, distribuito
- Oggi 13 «root nameserver»:
  - 10 in AM (USA)
  - 2 in EMEA (Inghilterra, Svizzera)
  - 1 APAC (Giappone)
- possibile attribuire più nomi allo stesso indirizzo IP (o viceversa).

- La «risoluzione» dell'indirizzo IP avviene per passi, rispetto alla struttura di dominio, cominciando dal primo livello, a scendere.
- Ogni query viene risolta in modo ricorsivo, interrogando i DNS server autoritativi
- I server DNS seguono una struttura gerarchica di delega e forniscono le informazioni concordemente.

DNS: Domain Name System

Indirizzamento IP e Nomi Mnemonici: automatismo 3/3



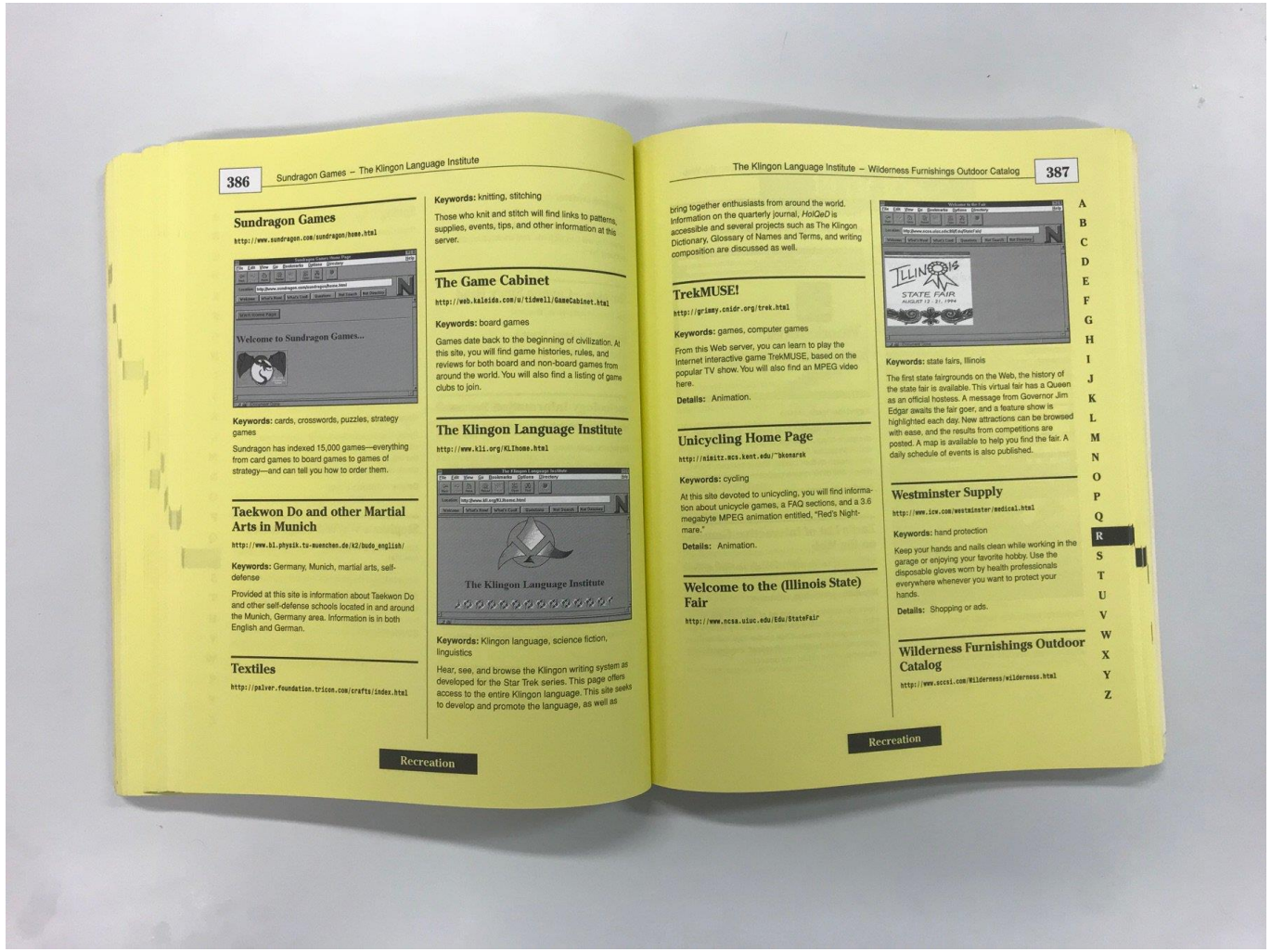
- No connection and unreliable the same socket can be used to send/receive
- Datagrams to/from multiple processes
- uniquely identified by the IP addresses and port numbers (remote and local) for:
  - client (`socket()`)
  - server (`socket()`, `bind()`)
- No syntax distinction between server and client, just:
  - `sendto()`
  - `recvfrom()`

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

DNS: Domain Name System

Prima dei Motori di Ricerca



IYP: Internet Yellow Pages

- I motori di ricerca referenziano i nomi di dominio
- Molto raramente sono elencati indirizzi IP
- Precedentemente, si usava pubblicare annualmente dei libri che elencavano i siti relativi ad una certa specificità (proprio come per gli elenchi telefonici)

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Simple DNS Server

DNS – RFC 1035 (1987): OpCode

Codice dell'Operazione (Metodi del Protocollo): Il client deve inserire nel messaggio di richiesta quale tipo si servigio vuole richiedere al server.

Il simple DNS server accetta:

- solo le operazioni QUERY

```
/* Operation Code */
enum {
  QUERY_OperationCode = 0, /* standard query */
  IQUERY_OperationCode = 1, /* inverse query */
  STATUS_OperationCode = 2, /* server status request */
  NOTIFY_OperationCode = 4, /* request zone transfer */
  UPDATE_OperationCode = 5 /* change resource records */
};
```

Opcode Value	Query Name	Description
0	*QUERY*	A standard query.
1	*IQUERY*	An inverse query; now obsolete. RFC 1035 defines the inverse query as an optional method for performing inverse DNS lookups, that is, finding a name from an IP address. Due to implementation difficulties, the method was never widely deployed, however, in favor of reverse mapping using the IN-ADDR.ARPA domain. Use of this Opcode value was formally obsoleted in RFC 3425, November 2002.
2	*STATUS*	A server status request.
3	(reserved)	Reserved, not used.
4	*NOTIFY*	A special message type added by RFC 1996. It is used by a primary (master, authoritative) server to tell secondary servers that data for a zone has changed and prompt them to request a zone transfer. See the discussion of DNS server enhancements for more details.
5	*UPDATE*	A special message type added by RFC 2136 to implement "dynamic DNS". It allows resource records to be added, deleted or updated selectively. See the discussion of DNS server enhancements for more details.

Simple DNS Server

DNS – RFC 1035 (1987): Resource Record Type

Tipo di Record (tipologia di informazione sulla risorsa): Il client deve inserire nel messaggio di richiesta a quale tipo di informazione si riiferisce il servigio.
Il simple DNS server accetta solo le informazioni:

- A: mapping nome – IP address (32 bit)
- TXT: informazione generica

```
/* Resource Record Types */
enum {
   A_Resource_RecordType = 1,
   NS_Resource_RecordType = 2,
   CNAME_Resource_RecordType = 5,
   SOA_Resource_RecordType = 6,
   PTR_Resource_RecordType = 12,
   MX_Resource_RecordType = 15,
   TXT_Resource_RecordType = 16,
   AAAA_Resource_RecordType = 28,
   SRV_Resource_RecordType = 33};
};
```

Common DNS Record Types	
Record	Description
A	Address record (IPv4)
AAAA	Address record (IPv6)
CNAME	Canonical Name record
MX	Mail Exchanger record
NS	Nameserver record
PTR	Pointer record
SOA	Start of Authority record
SRV	Service Location record
TXT	Text record

Simple DNS Server

DNS – RFC 1035 (1987): RCode

Codice di Risposta: Il server deve inserire il codice opportuno nel messaggio di risposta al client.

```
/* Response Type */
enum {
Ok_ResponseType = 0,
FormatError_ResponseType = 1,
ServerFailure_ResponseType = 2,
NameError_ResponseType = 3,
NotImplemented_ResponseType = 4,
Refused_ResponseType = 5
};
```

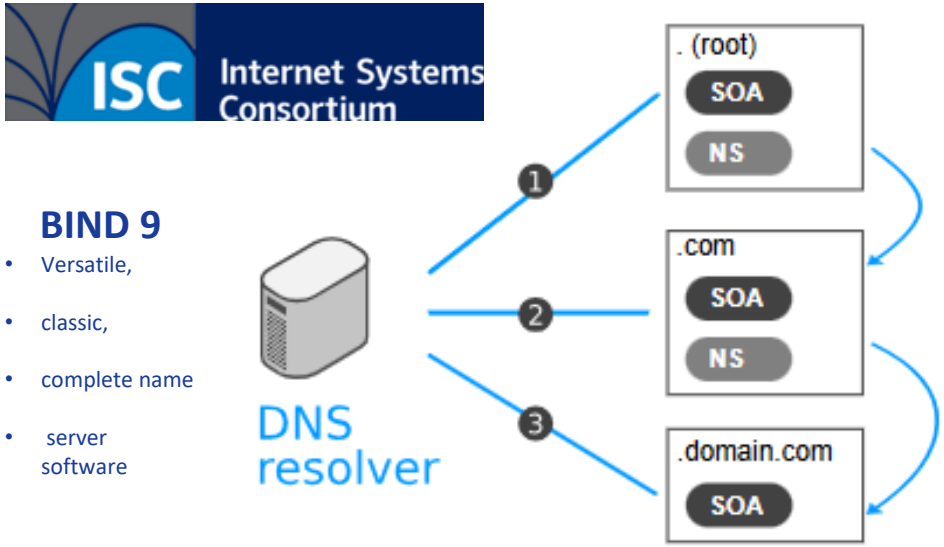
RCode Value	Response Code	Description
0	*No Error*	No error occurred.
1	*Format Error*	The server was unable to respond to the query due to a problem with how it was constructed.
2	*Server Failure*	The server was unable to respond to the query due to a problem with the server itself.
3	*Name Error*	The name specified in the query does not exist in the domain. This code can be used by an authoritative server for a zone (since it knows all the objects and subdomains in a domain) or by a caching server that implements negative caching.
4	*Not Implemented*	The type of query received is not supported by the server.
5	*Refused*	The server refused to process the query, generally for policy reasons and not technical ones. For example, certain types of operations, such as zone transfers, are restricted. The server will honor a zone transfer request only from certain devices.
6	*YX Domain*	A name exists when it should not.
7	*YX RR Set*	A resource record set exists that should not.
8	*NX RR Set*	A resource record set that should exist does not.
9	*Not Auth*	The server receiving the query is not authoritative for the zone specified.
10	*Not Zone*	A name specified in the message is not within the zone specified in the message.

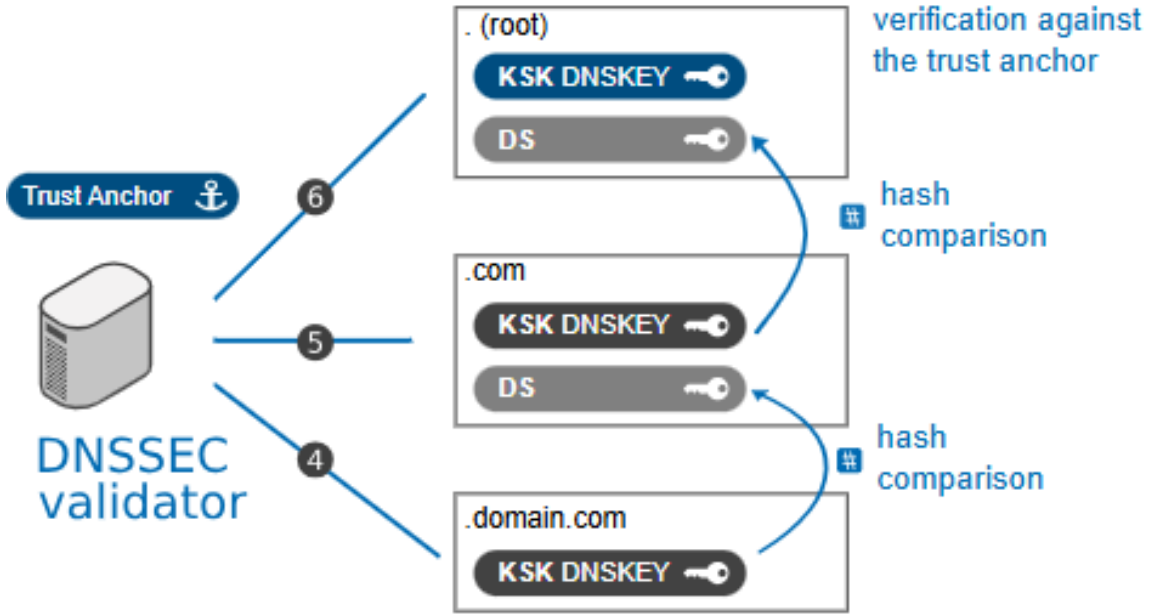
Furthers on DNS Server

Bind – DNSSec

Questa esercitazione è relativa ad un semplice server DNS, non autoritativo, senza capacità di ricerca ricorsiva, scritto «from scratch», in C.
Valido come PoC (Proof of Concept), in modo da poter comprenderne il funzionamento.

Per esigenze operative reali, il server DNS più usato al mondo è Bind (da ISC: Internet System Consortium).





DNSSec (Domain Name System Security Extensions) rafforza la sicurezza del protocollo DNS (intrinsecamente non sicuro by design). Brevemente, un server che offre DNSSEC per le sue zone fa uso di **crittografia a chiave pubblica**, permettendo di:

- **Record Integrity**: verificare l'integrità di ogni record.
- **Record Authenticity**: convalidare il record come proveniente dal server DNS per esso autorevole.
- **Chain of Trust**: convalidare il server DNS in base alla attendibilità asserita dal dominio superiore nella gerarchia DNS.

L'RFC 9364 costituisce un «single reference» per DNSSEC.

Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti
Sapienza Università di Roma

Simple DNS server

Lab10, es.1 DNS server

- Ascolto:
- Risposta:

Simple DNS Server

Caratteristiche

Il server DNS deve effettuare le seguenti operazioni:

1. ascoltare le connessioni in arrivo su una **porta UDP 9000**. → configurare i parametri
2. configurare la porta per l'ascolto → `socket(),bind()`
3. ricevere le query in arrivo → `recvfrom()`
4. Accettare solo **richieste** dal client valide di tipo **A** o **TXT**.
5. estrarre il **nome host** richiesto
6. creare una **risposta DNS** con le intestazioni appropriate
7. Inviare la risposta DNS → `sendto()`

Il server è un demone: continua ad accettare ed elaborare le connessioni in arrivo finché non viene terminato manualmente (Ctrl-C, kill -9 etc.).

Simple DNS Server

Lab10, es1

Completare il codice del DNS Server
Sorgenti

- `Makefile`
- Client: <none> usare l'utilita' dig a riga di comando
- Server: `main.c`

Suggerimento: seguire i blocchi di commenti inseriti nel codice

Funzionamento:

- Avviare il server: `./main &`
- Effettuare una richiesta dal client `dig`

```
$ dig @127.0.0.1 -p 9000 uniroma1.it A
```

Si dovrebbe ottenere la risposta come illustrata nella pagina successiva

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Simple DNS Server

Lab10, es1

```
$ dig @127.0.0.1 -p 9000 uniroma1.it
QUERY { ID: 865a. FIELDS: [ QR: 0, OpCode: 0 ], QDcount: 1, ANcount: 0, NScount: 0, ARcount: 1,
  Question { qName 'uniroma1.it', qType 1, qClass 1 }
}
Query for 'uniroma1.it'
QUERY { ID: 865a. FIELDS: [ QR: 1, OpCode: 0 ], QDcount: 1, ANcount: 1, NScount: 0, ARcount: 0,
  Question { qName 'uniroma1.it', qType 1, qClass 1 }
  ResourceRecord { name 'uniroma1.it', type 1, class 1, ttl 3600, rd_length 4, Address Resource Record { address 151.100.101.140 }}
}

; <<>> DiG 9.18.8-1-Debian <<>> @127.0.0.1 -p 9000 uniroma1.it
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34394
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;uniroma1.it.                   IN      A

;; ANSWER SECTION:
uniroma1.it.            3600    IN      A       151.100.101.140

;; Query time: 0 msec
;; SERVER: 127.0.0.1#9000(127.0.0.1) (UDP)
;; WHEN: Mon Dec 16 00:17:47 CET 2024
;; MSG SIZE  rcvd: 56
```

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Simple VPN

Tunneling
Interfaccia di Rete Virtuale

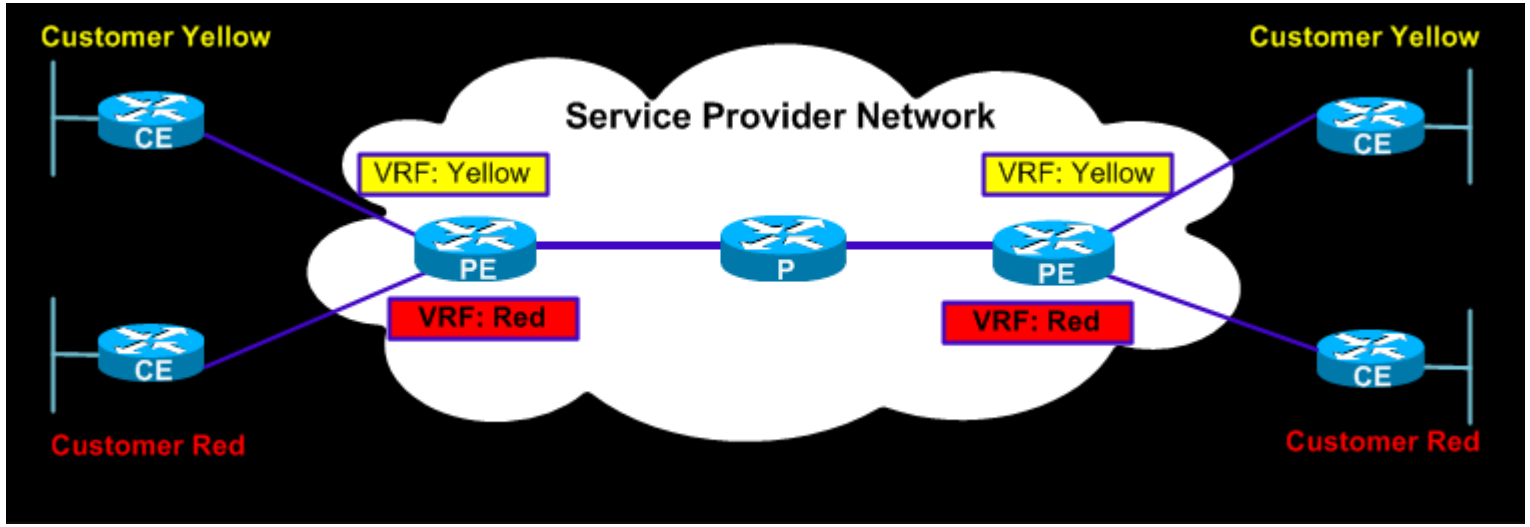
Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

MPLS VPN

Reti Private Virtual a Livello 3 (con controllo dell'ISP: senza encrypt-decrypt)

Le reti Multi-Protocol Label Switching (MPLS) sono reti progettate per consentire ai clienti di creare circuiti end-to-end su qualsiasi tipo di mezzo di trasporto utilizzando qualsiasi tecnologia WAN disponibile. MPLS funziona etichettando il traffico che entra nella rete MPLS. Un identificatore (**etichetta**) viene utilizzato per aiutare a **distinguere** il **Label Switched Path** (LSP) da utilizzare per instradare il pacchetto verso la sua destinazione corretta.



I fornitori di servizi **impediscono** che i router (**P**: Provider, **PE**: Provider Edge) siano **raggiungibili** via **Internet** utilizzando tecniche note come il filtraggio dei pacchetti e ACL: accesso solo alle porte del protocollo di routing (ad esempio BGP) da aree specifiche all'interno della loro rete. **Nessuna informazione** su **LSP** è **rivelata** a terze parti

Se **anche** l'accesso a **Internet** è fornito al cliente tramite il collegamento **MPLS**, gli ISP utilizzano meccanismi simili per **bloccare** i router **CE** (Customer Edge). Inoltre, sono configurati:

- **autenticazione MD5** per i protocolli di **routing** (BGP, OSPF ecc.)
- configurazione del **numero massimo di percorsi** accettati per istanza di Virtual Routing and Forwarding (VRF)

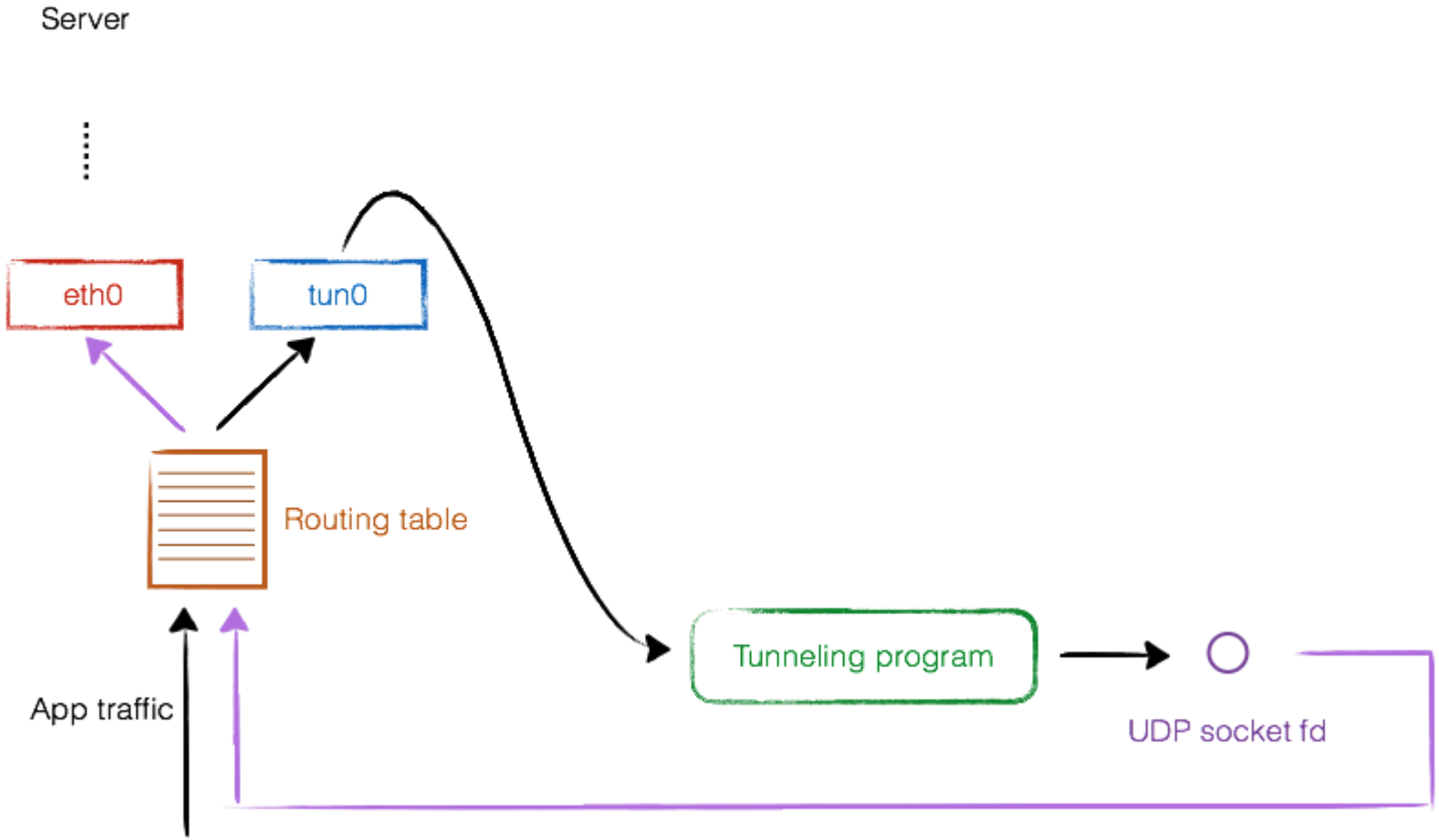
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Simple VPN

Introduzione

VPN sta per Virtual Private Network, ovvero una rete privata, con molti componenti virtualizzati.



Dal punto di vista dell'utente, si necessita di:

1. un'**interfaccia di rete virtuale** (ad esempio `/dev/tun0`). →
2. configurare la **tabella di routing** per instradare tutto o **parte del traffico** che passa **attraverso** quell'**interfaccia**

tutto il traffico va all'interfaccia virtuale, eccetto i pacchetti con i dati di tunneling che vanno direttamente al server VPN tramite normali interfacce di rete.

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Simple VPN

Utilizzo del Tunneling

In sintesi, il processo per il tunneling lato client è il seguente:

1. Aprire un **socket UDP** il cui altro lato è il server/client. → `socket()`
   Solo per il server → `bind()`
2. Creare il **dispositivo `tun`**, configurarlo e avviarlo. → `fd = open("/dev/net/tun", O_RDWR), ioctl(fd, TUNSETIFF, (void *) &ifr)`
3. Configurare la **tabella di routing**. → `run (iptables),run (route add <server> via …)`
4. **Leggere** i pacchetti dal dispositivo **`tun`**. → `read(tun_fd)`
5. **Crittografarli**. → `encrypt()`
6. **inviarli** al server tramite il **socket** creato. → `sendto()`
7. **leggerli** dal **socket**. → `recvfrom()`
8. **Decrittografarli**. → `decrypt()`
9. **scriverli** sul dispositivo. → `write(tun_fd)`

Il server è un **demone**: continua ad accettare ed elaborare le connessioni in arrivo finché non viene **terminato manualmente** (Ctrl-C, kill -9 etc.), pulendo le configurazioni di cui al punto 3., prima di uscire, tramite la funzione `cleanup_when_sig_exit()`.

Lato server, il processo è il converso: del tutto simile ma contrario

Simple VPN

TUN (/dev/net/tun) – ioctl()

TUN (`/dev/net/tunX`) sono driver che permettono la creazione di periferiche di rete virtuali di livello 3 (IP). Rispetto alle comuni periferiche (ad es. eth0) che sono controllate direttamente dalle schede di rete, i pacchetti spediti da o verso dispositivi TUN sono spediti da o verso programmi software. TUN è in grado di simulare una periferica di rete di tipo punto-punto e lavora con pacchetti di tipo IP. La periferica «clonata» `/dev/net/tunX` può essere usata anche da un utente non privilegiato.

ioctl() = Input Output Control (Device Management).

```
#include <sys/ioctl.h>
int ioctl(int fd, unsigned long cmd, ...);
```

Ovvero deve essere referenziato:

- `fd`: file descriptor
- `cmd`: il comando da impartire
- …: un parametro opzionale

Serve per eseguire operazioni specifiche senza il ricorso ad una ennesima system call.
Consente di gestire i dispositivi (device) tramite l'implementazione nel driver.
Si implementa nel driver per operazioni "speciali" che non possono essere eseguite tramite read/write (es. configurazione del MTU).

Simple VPN Tunnel

Lab10, es.2 VPN tunnel

- Ascolto:
- Risposta:

Simple VPN Tunnel

Lab10, es2

Completare il codice del VPN tunnel
Sorgenti

- `Makefile`
- Server e Client: `vpn.c` (contiene sia il codice del server, sia del client. Il Makefile esegue 2 compilazioni distinte, guidate dalle direttive al pre-processore: per la generazione del client sono usate le definizioni:

```
CLNTFLAGS = -DAS_CLIENT=YES -DSERVER_HOST="\"10.8.0.1\""
```

Suggerimento: seguire i blocchi di commenti inseriti nel codice

Funzionamento:

- Avviare il vpn server: `sudo ./vpn &`
- Avviare il vpn client: `sudo ./client`

ATTENZIONE: `ioctl()` richiede privilegi amministrativi → lanciare con sudo!!!

```
$ ifconfig
```

Dovrebbero essere presenti anche le interfacce virtuali di tunneling (`tun0`, `tun1`), oltre alle «classiche»: `eth0`, `lo`.

Simple VPN Tunnel

Lab10, es2

```
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.31.38.88  netmask 255.255.240.0  broadcast 172.31.47.255
        inet6 fe80::215:5dff:fef0:3c27  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:f0:3c:27  txqueuelen 1000  (Ethernet)
        RX packets 2197  bytes 462759 (451.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 103  bytes 10068 (9.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 66  bytes 4952 (4.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 66  bytes 4952 (4.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1400
        inet 10.8.0.1  netmask 255.255.0.0  destination 10.8.0.1
        inet6 fe80::43d0:3279:5a17:2cbf  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 17  bytes 816 (816.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 480 (480.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tun1: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1400
        inet 10.8.0.2  netmask 255.255.0.0  destination 10.8.0.2
        inet6 fe80::6be6:5055:583b:d822  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 336 (336.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA