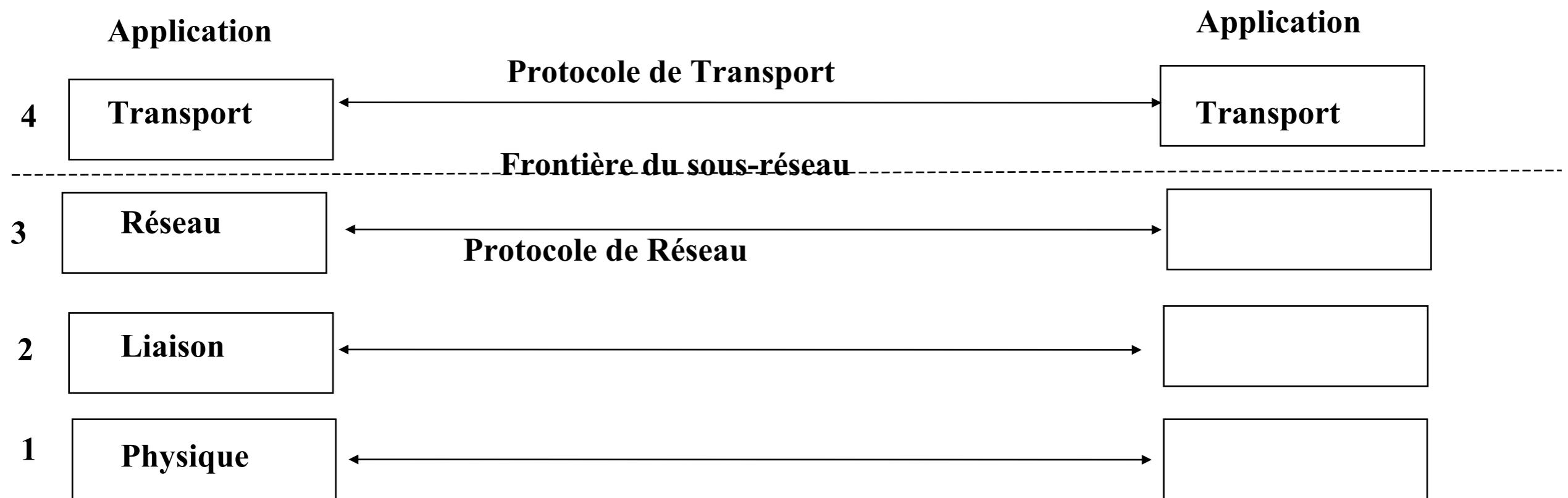


# La couche application sur TCP/IP – accès - outils



# Internet Protocol Suite

**L'ensemble de protocoles de communication utilisés pour Internet**

- **Couche application :**
  - protocoles « outils » : DNS, RIP, ...
  - protocoles « application de base » : HTTP, FTP, SMTP, ...
  - interface de programmation (API) pour vos applications : les Sockets
- **Couche transport :** TCP (connection virtuelle), UDP
- **Couche réseau :** IPv4, outils : ICMP, NAT, DHCP
- **Couche liaison :** MAC / Ethernet, outils : ARP, RARP

# Protocole de Routage RIP utilise UDP

- **RIP (Routing Information Protocol)**
  - échange d'annuaires d'adresses réseau avec coût (hop count) entre routeurs adjacents
  - chaque routeur construit une table avec sa vision des réseaux connus
    - » adresse réseau
    - » routeur pour cette adresse / direct
    - » coût  $\geq 1$  (nombre de routeurs jusqu'au réseau)
  - chaque routeur partage régulièrement (timer 30s) sa table avec ses voisins
    - » Mise à jour du coût si nécessaire
    - » Efface les adresses de réseaux qui n'ont pas été mises à jour (timer 180s)
    - » UDP port 520

# Protocole ICMP utilise IP

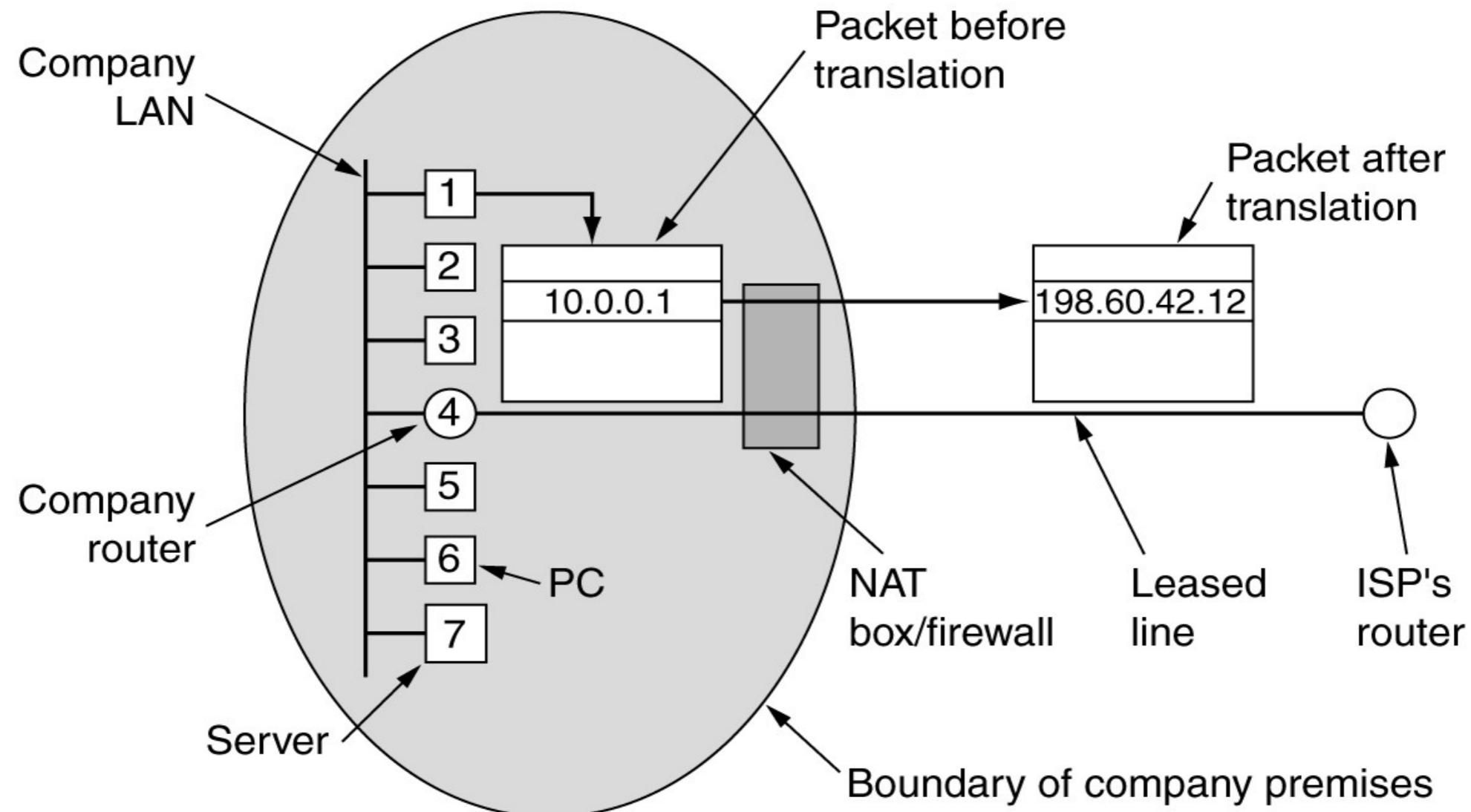
- **ICMP (Internet Control Message Protocol)**
  - protocole pour un message lorsqu'il y a erreur de transmission
  - envoyé à l'adresse source d'un paquet IP si erreur :
    - » champ TTL atteint 0
    - » destination pas atteignable
    - » ...

# Internet Control Message Protocol

<b>Message type</b>	<b>Description</b>
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo request	Ask a machine if it is alive
Echo reply	Yes, I am alive
Timestamp request	Same as Echo request, but with timestamp
Timestamp reply	Same as Echo reply, but with timestamp

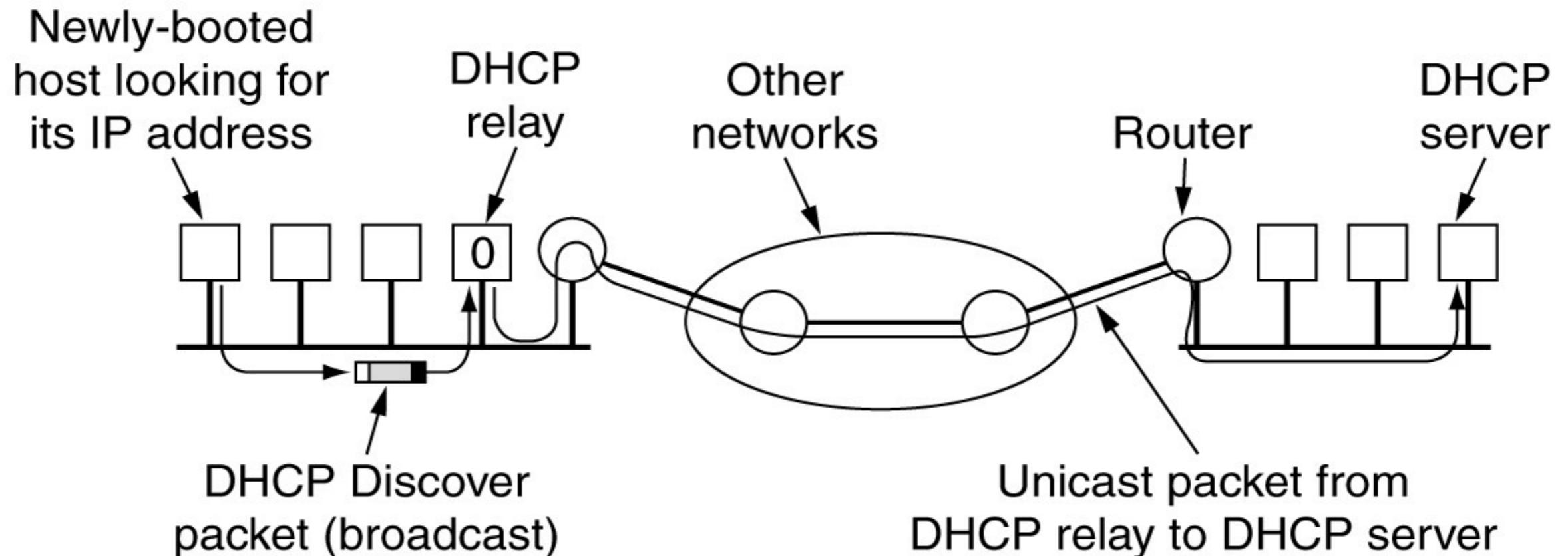
The principal ICMP message types.

# Network Address Translation (NAT), utilise IP



Placement and operation of a NAT box.

# Dynamic Host Configuration Protocol (DHCP), utilise IP



Operation of DHCP.

# Hypertext Transfer Protocol (HTTP), utilise TCP port80

- protocole d'échange de données (ressources) entre un client et un serveur
- utilisé sur le « web »
- Version sécurisé HTTPS
- identification des ressources avec URL (Uniform Resource Locator  
`<protocol>://<domain name>/<file system path>`  
<http://example.org/path/to/resource.txt>
- Commandes GET, POST, ..., envoyés en texte par le client

**requête du client :**      `GET /index.html HTTP/1.1`

**reponse du serveur :**    `HTTP/1.1 302 Found`  
`Location: http://www.google.fr/index.html`  
`Cache-Control: private`  
`Content-Type: text/html; charset=UTF-8`  
`...`  
`<HTML><HEAD><meta http-equiv="content-type"`  
`content="text/html; charset=utf-8">`  
`<TITLE>302 Moved</TITLE></HEAD><BODY>`  
`The document has moved`  
`</BODY></HTML>`

## **FTP (File Transfer Protocol), utilise TCP**

- protocole d'échange de fichiers entre un client et un serveur
- basé sur TCP
- commandes RETR (GET) , CWD (CD) , ... , envoyés en texte par le client
- version sécurisée : SFTP
- avec authentification d'utilisateur

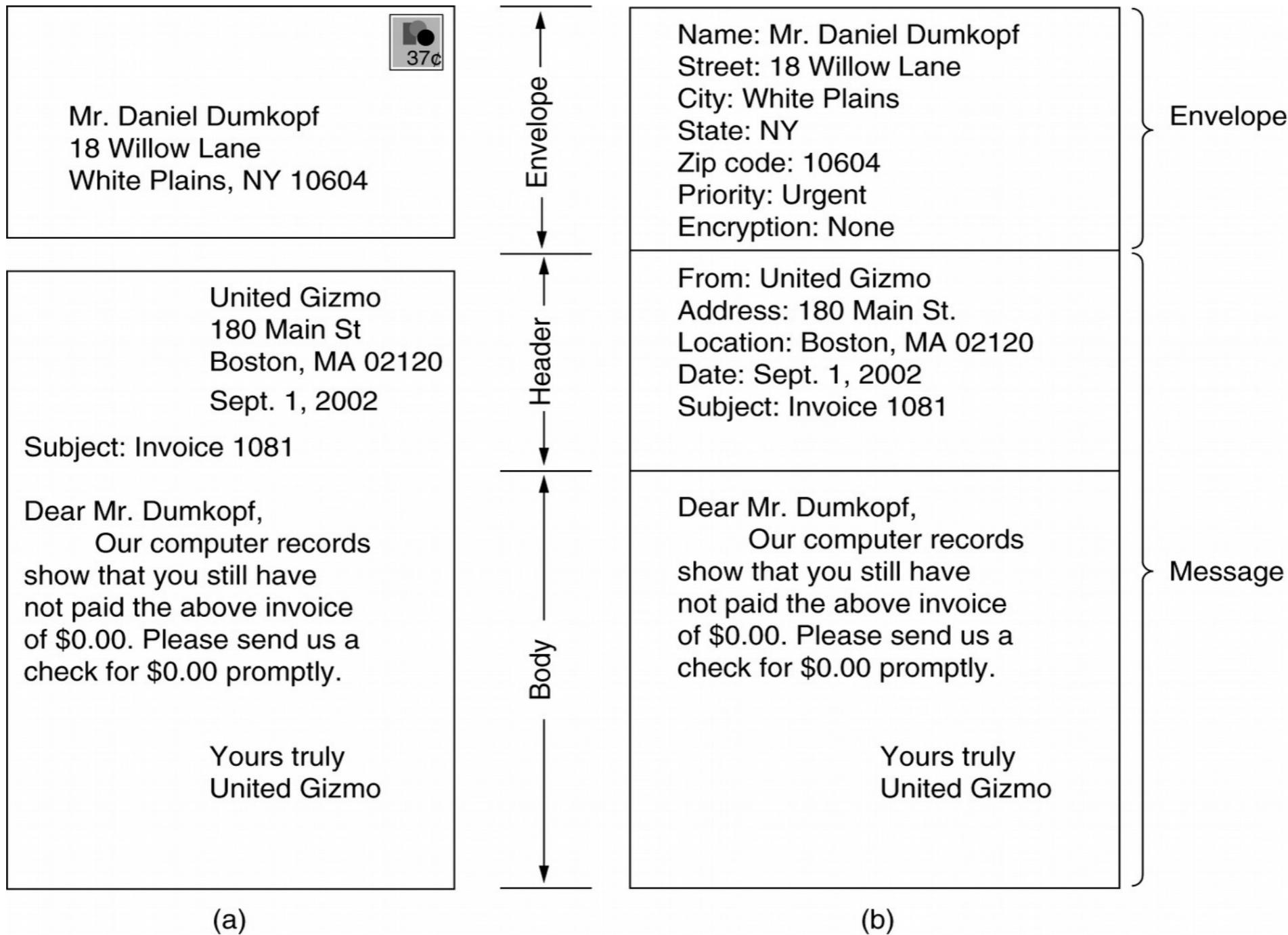
# Electronic Mail

- Architecture and Services
- The User Agent
- Message Formats
- Message Transfer SMTP
- Final Delivery

## **SMTP (Simple Mail Transfer Protocol), utilise TCP**

- protocole d'échange de messages text entre un client et un serveur
- basé sur TCP
- utilisé pour échanger des emails
- entête avec champs : sender, receiver, reply, subject, complementary copy, ...
- commandes MAIL (from), DATA (texte), ..., envoyés en texte ascii par le client
- notion d'agents

# The User Agent Functions and Tools



Envelopes and messages. (a) Paper mail. (b) Electronic mail.

# Message Formats – RFC 822

<b>Header</b>	<b>Meaning</b>
To:	E-mail address(es) of primary recipient(s)
Cc:	E-mail address(es) of secondary recipient(s)
Bcc:	E-mail address(es) for blind carbon copies
From:	Person or people who created the message
Sender:	E-mail address of the actual sender
Received:	Line added by each transfer agent along the route
Return-Path:	Can be used to identify a path back to the sender

RFC 822 header fields related to message transport.

# Message Formats – RFC 822 (2)

<b>Header</b>	<b>Meaning</b>
Date:	The date and time the message was sent
Reply-To:	E-mail address to which replies should be sent
Message-Id:	Unique number for referencing this message later
In-Reply-To:	Message-Id of the message to which this is a reply
References:	Other relevant Message-Ids
Keywords:	User-chosen keywords
Subject:	Short summary of the message for the one-line display

Some fields used in the RFC 822 message header.

# MIME – Multipurpose Internet Mail Extensions

Problems with international languages:

- Languages with accents  
(French, German).
- Languages in non-Latin alphabets  
(Hebrew, Russian).
- Languages without alphabets  
(Chinese, Japanese).
- Messages not containing text at all  
(audio or images).

# MIME (2)

<b>Header</b>	<b>Meaning</b>
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

RFC 822 headers added by MIME.

# MIME (3)

Type	Subtype	Description
Text	Plain	Unformatted text
	Enriched	Text including simple formatting commands
Image	Gif	Still picture in GIF format
	Jpeg	Still picture in JPEG format
Audio	Basic	Audible sound
Video	Mpeg	Movie in MPEG format
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in PostScript
Message	Rfc822	A MIME RFC 822 message
	Partial	Message has been split for transmission
	External-body	Message itself must be fetched over the net
Multipart	Mixed	Independent parts in the specified order
	Alternative	Same message in different formats
	Parallel	Parts must be viewed simultaneously
	Digest	Each part is a complete RFC 822 message

The MIME types and subtypes defined in RFC 2045.

## MIME (4)

From: elinor@abcd.com  
To: carolyn@xyz.com  
MIME-Version: 1.0  
Message-Id: <0704760941.AA00747@abcd.com>  
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm  
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm

Content-Type: text/enriched

Happy birthday to you

Happy birthday to you

Happy birthday dear <b>Carolyn</b>

Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm

Content-Type: message/external-body;

access-type="anon-ftp";

site="bicycle.abcd.com";

directory="pub";

name="birthday.snd"

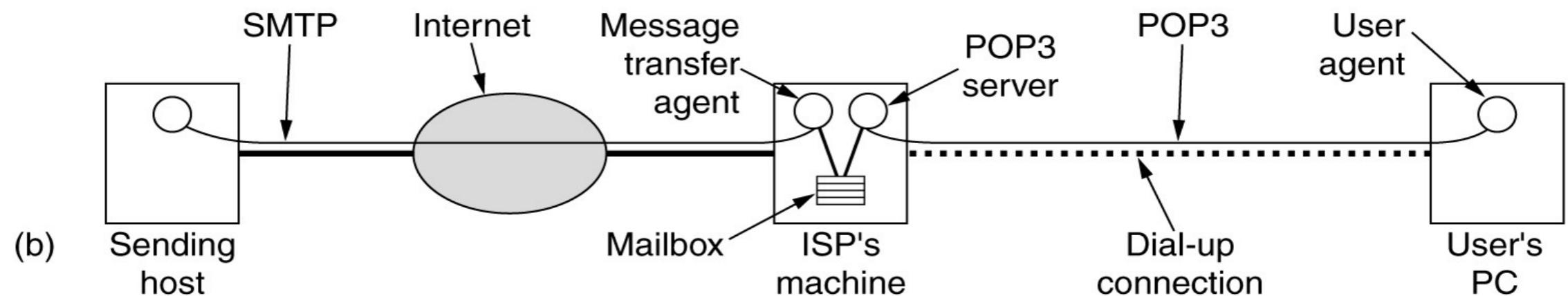
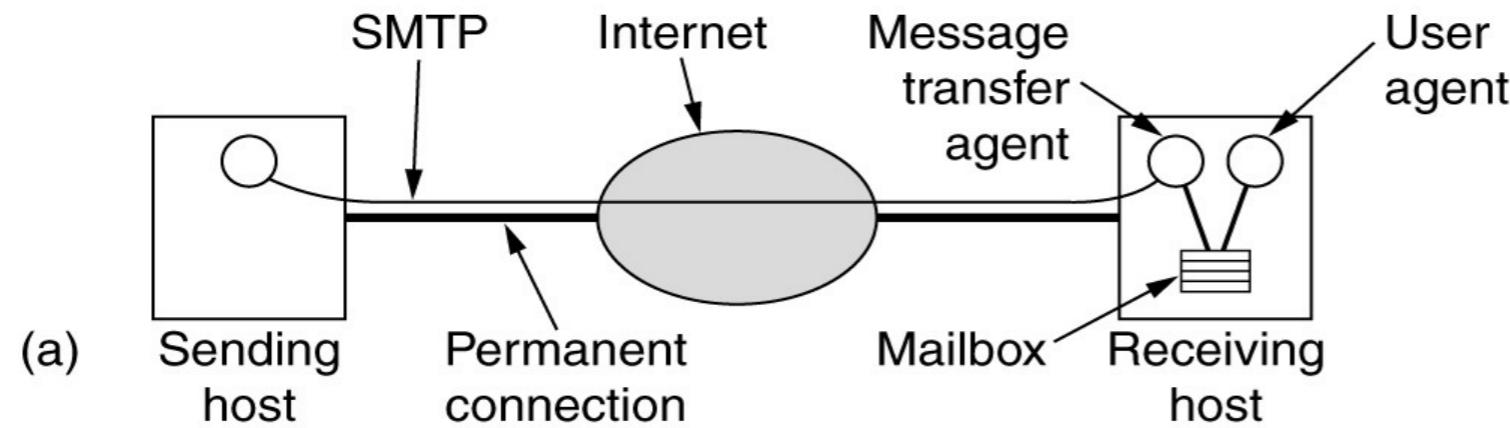
content-type: audio/basic

content-transfer-encoding: base64

--qwertyuiopasdfghjklzxcvbnm--

A multipart message containing enriched and audio alternatives.

# SMTP : Final Delivery with Background Servers



(a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent. (b) Reading e-mail when the receiver has a dial-up connection to an ISP.

# POP3 Email Server

```
S: +OK POP3 server ready
C: USER carolyn
    S: +OK
C: PASS vegetables
    S: +OK login successful
C: LIST
    S: 1 2505
    S: 2 14302
    S: 3 8122
    S: .
C: RETR 1
    S: (sends message 1)
C: DELE 1
C: RETR 2
    S: (sends message 2)
C: DELE 2
C: RETR 3
    S: (sends message 3)
C: DELE 3
C: QUIT
    S: +OK POP3 server disconnecting
```

Using POP3 to fetch three messages.

# SMTP Message Transfer Sequence

S: 220 xyz.com SMTP service ready  
C: HELO abcd.com  
S: 250 xyz.com says hello to abcd.com  
C: MAIL FROM: <elinor@abcd.com>  
S: 250 sender ok  
C: RCPT TO: <carolyn@xyz.com>  
S: 250 recipient ok  
C: DATA  
S: 354 Send mail; end with "." on a line by itself  
C: From: elinor@abcd.com  
C: To: carolyn@xyz.com  
C: MIME-Version: 1.0  
C: Message-Id: <0704760941.AA00747@abcd.com>  
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm  
C: Subject: Earth orbits sun integral number of times  
C:  
C: This is the preamble. The user agent ignores it. Have a nice day.  
C:  
C: --qwertyuiopasdfghjklzxcvbnm  
C: Content-Type: text/enriched  
C:  
C: Happy birthday to you  
C: Happy birthday to you  
C: Happy birthday dear <bold> Carolyn </bold>  
C: Happy birthday to you  
C:  
C: --qwertyuiopasdfghjklzxcvbnm  
C: Content-Type: message/external-body;  
C: access-type="anon-ftp";  
C: site="bicycle.abcd.com";  
C: directory="pub";  
C: name="birthday.snd"  
C:  
C: content-type: audio/basic  
C: content-transfer-encoding: base64  
C: --qwertyuiopasdfghjklzxcvbnm  
C: .  
S: 250 message accepted  
C: QUIT  
S: 221 xyz.com closing connection

Transferring a message from  
*elinore@abc.com* to  
*carolyn@xyz.com*.

# IMAP Email Server

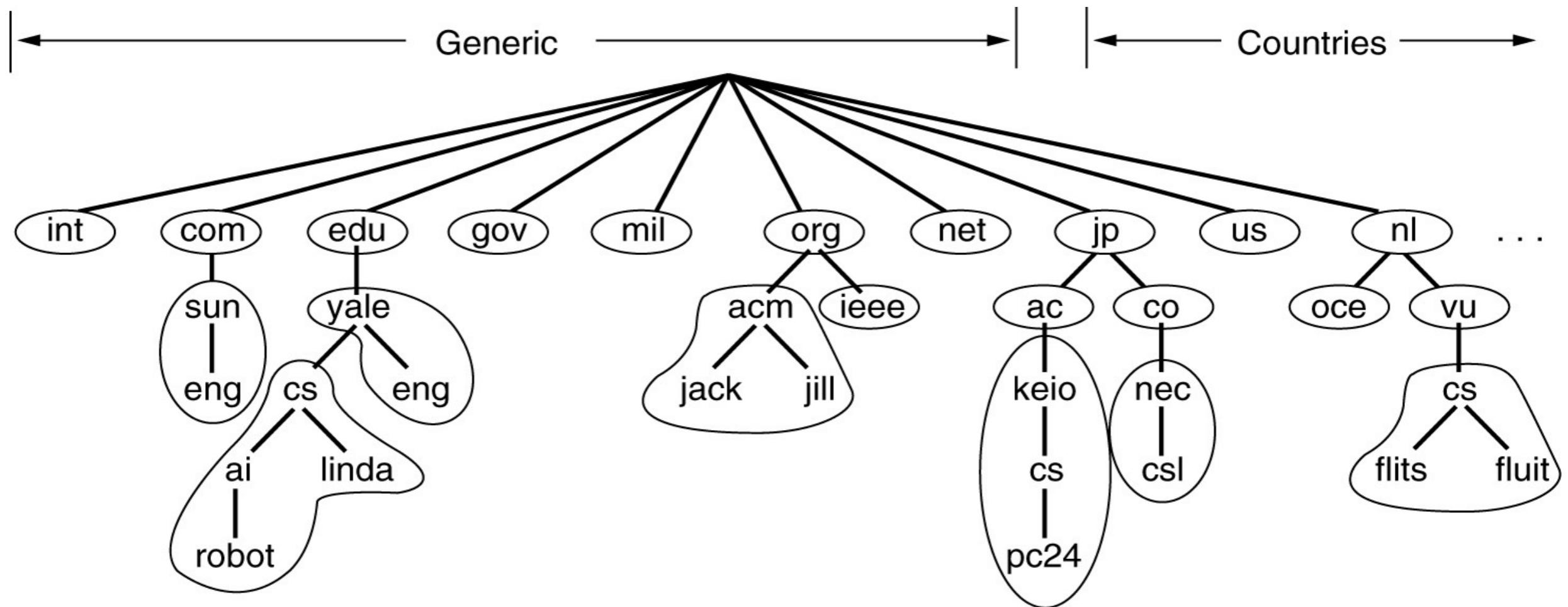
Feature	POP3	IMAP
Where is protocol defined?	RFC 1939	RFC 2060
Which TCP port is used?	110	143
Where is e-mail stored?	User's PC	Server
Where is e-mail read?	Off-line	On-line
Connect time required?	Little	Much
Use of server resources?	Minimal	Extensive
Multiple mailboxes?	No	Yes
Who backs up mailboxes?	User	ISP
Good for mobile users?	No	Yes
User control over downloading?	Little	Great
Partial message downloads?	No	Yes
Are disk quotas a problem?	No	Could be in time
Simple to implement?	Yes	No
Widespread support?	Yes	Growing

A comparison of POP3 and IMAP.

# DNS – The Domain Name System

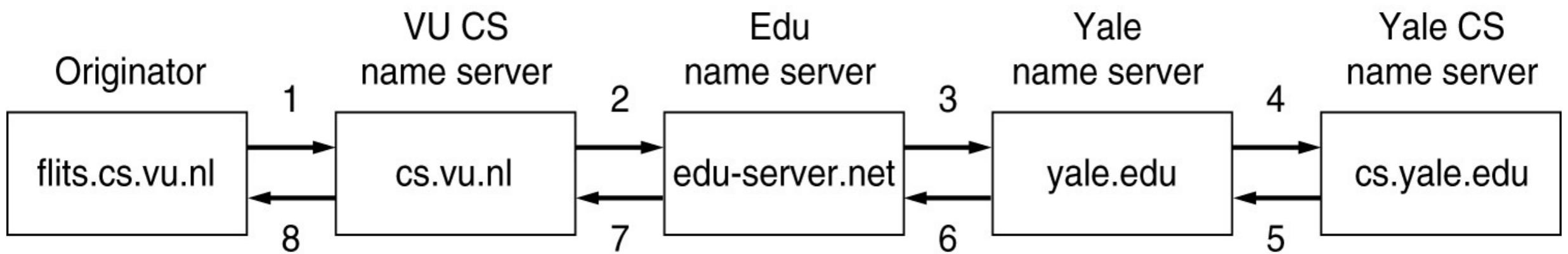
- The DNS Name Space
- Resource Records
- Name Servers
- Distributed database
- Hierarchy of servers

# DNS Name Servers



Part of the DNS name space showing the division into zones.

# Name Servers Hierarchy (2)



How a resolver looks up a remote name in eight steps.

# DNS Resource Records

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

The principal DNS resource records types.

# DNS Resource Records (2)

; Authoritative data for cs.vu.nl

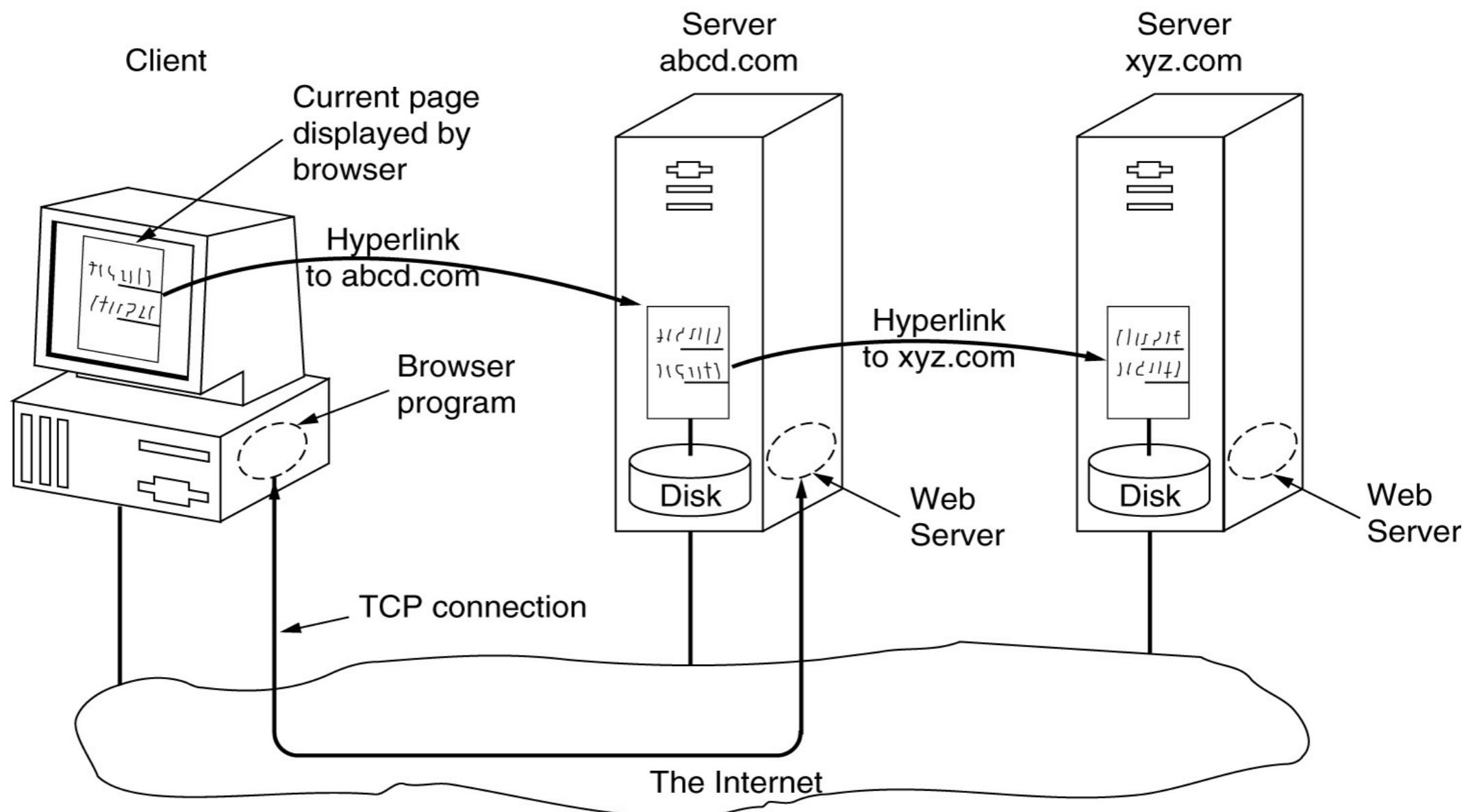
cs.vu.nl.	86400	IN	SOA	star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.	86400	IN	TXT	"Divisie Wiskunde en Informatica."
cs.vu.nl.	86400	IN	TXT	"Vrije Universiteit Amsterdam."
cs.vu.nl.	86400	IN	MX	1 zephyr.cs.vu.nl.
cs.vu.nl.	86400	IN	MX	2 top.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	HINFO	Sun Unix
flits.cs.vu.nl.	86400	IN	A	130.37.16.112
flits.cs.vu.nl.	86400	IN	A	192.31.231.165
flits.cs.vu.nl.	86400	IN	MX	1 flits.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	2 zephyr.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	3 top.cs.vu.nl.
www.cs.vu.nl.	86400	IN	CNAME	star.cs.vu.nl
ftp.cs.vu.nl.	86400	IN	CNAME	zephyr.cs.vu.nl
rowboat		IN	A	130.37.56.201
		IN	MX	1 rowboat
		IN	MX	2 zephyr
		IN	HINFO	Sun Unix
little-sister		IN	A	130.37.62.23
		IN	HINFO	Mac MacOS
laserjet		IN	A	192.31.231.216
		IN	HINFO	"HP Laserjet IISi" Proprietary

A portion of a possible DNS database for *cs.vu.nl*.

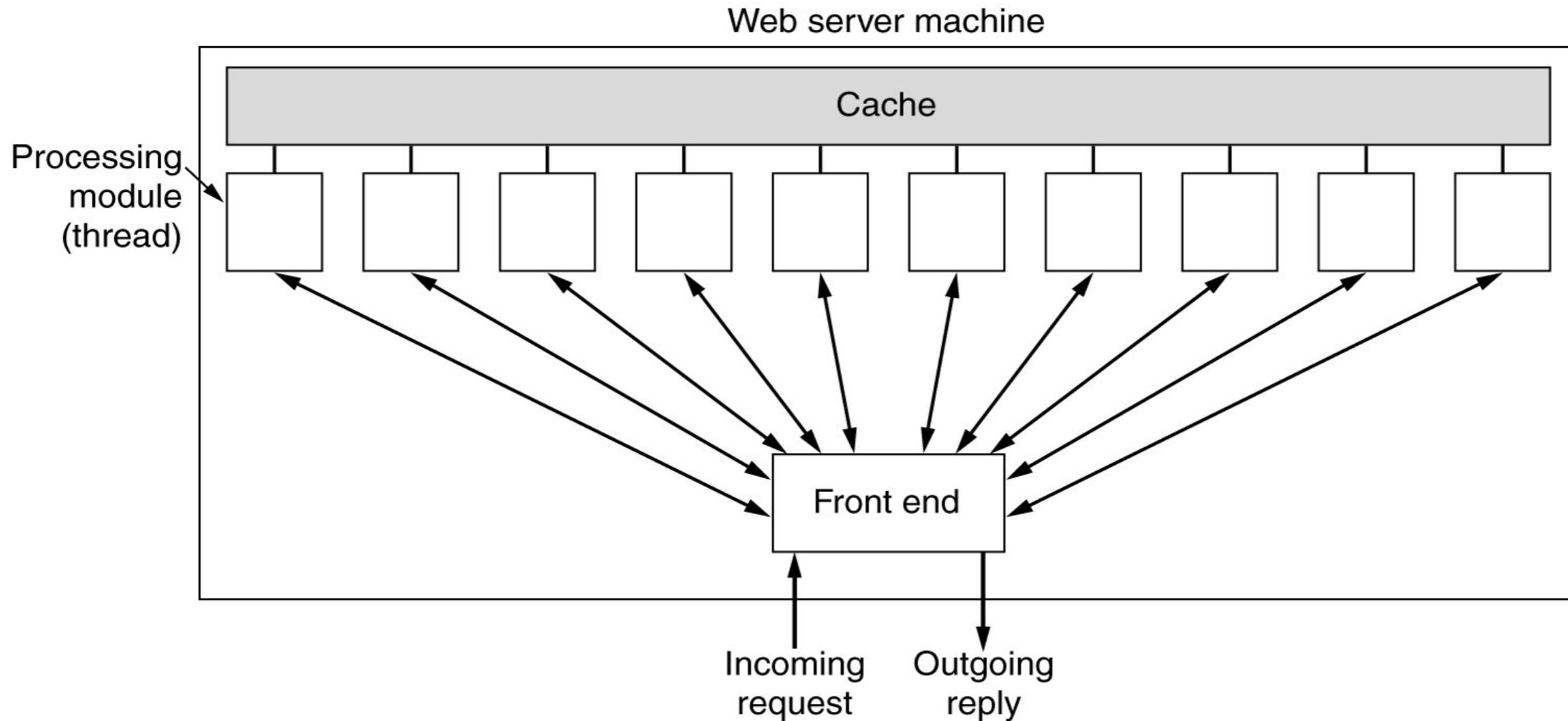
# The World Wide Web

- Architectural Overview
- Static Web Documents
- Dynamic Web Documents
- HTTP – The HyperText Transfer Protocol
- Performance Enhancements
- The Wireless Web

# Architectural Overview of the Internet

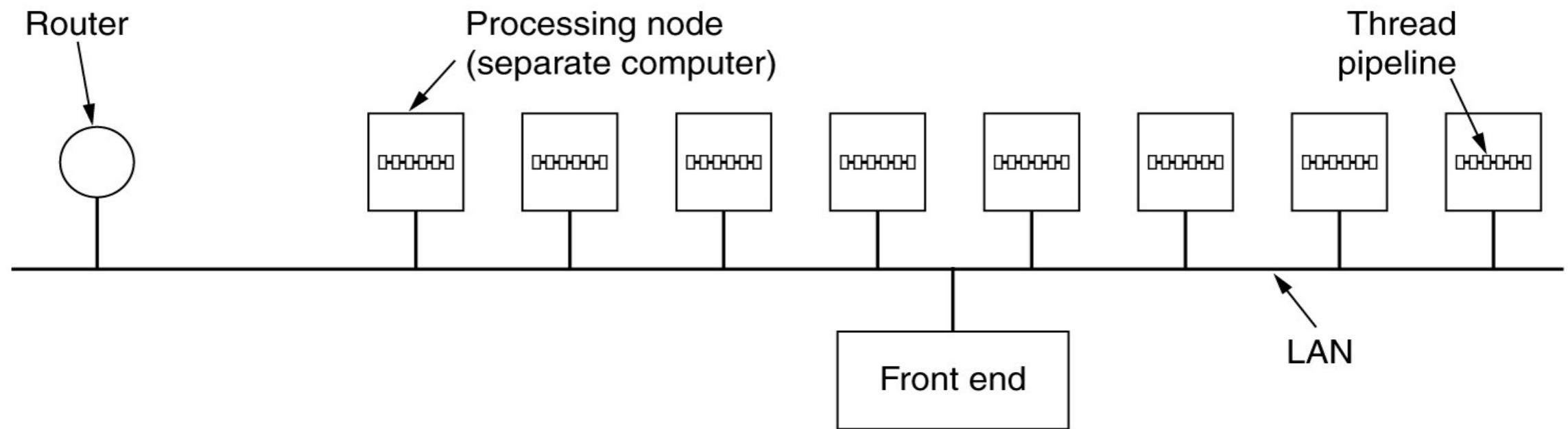


# HTTP The Server Side



A multithreaded Web server with a front end and processing modules.

# HTTP The Server Side with a server farm



A server farm.

# URLs – Uniform Resource Locators

Name	Used for	Example
http	Hypertext (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
news	Newsgroup	news:comp.os.minix
news	News article	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Sending e-mail	mailto:JohnUser@acm.org
telnet	Remote login	telnet://www.w3.org:80

Some common URLs.

# HTML – HyperText Markup Language

```
<html>
<head><title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page</h1>
 <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's </b>
home page. We hope <i> you </i> will find all the information you need here.
<p> Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. </p>
<hr>
<h2> Product information </h2>
<ul>
  <li> <a href="http://widget.com/products/big"> Big widgets </a>
  <li> <a href="http://widget.com/products/little"> Little widgets </a>
</ul>
<h2> Telephone numbers</h2>
<ul>
  <li> By telephone: 1-800-WIDGETS
  <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

(a)

## Welcome to AWI's Home Page



We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope you will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by FAX.

---

**Product Information**

- [Big widgets](http://widget.com/products/big)
- [Little widgets](http://widget.com/products/little)

**Telephone numbers**

- 1-800-WIDGETS
- 1-415-765-4321

(b)

(a) The HTML for a sample Web page. (b) The formatted page.

# HTML (2) a markup language

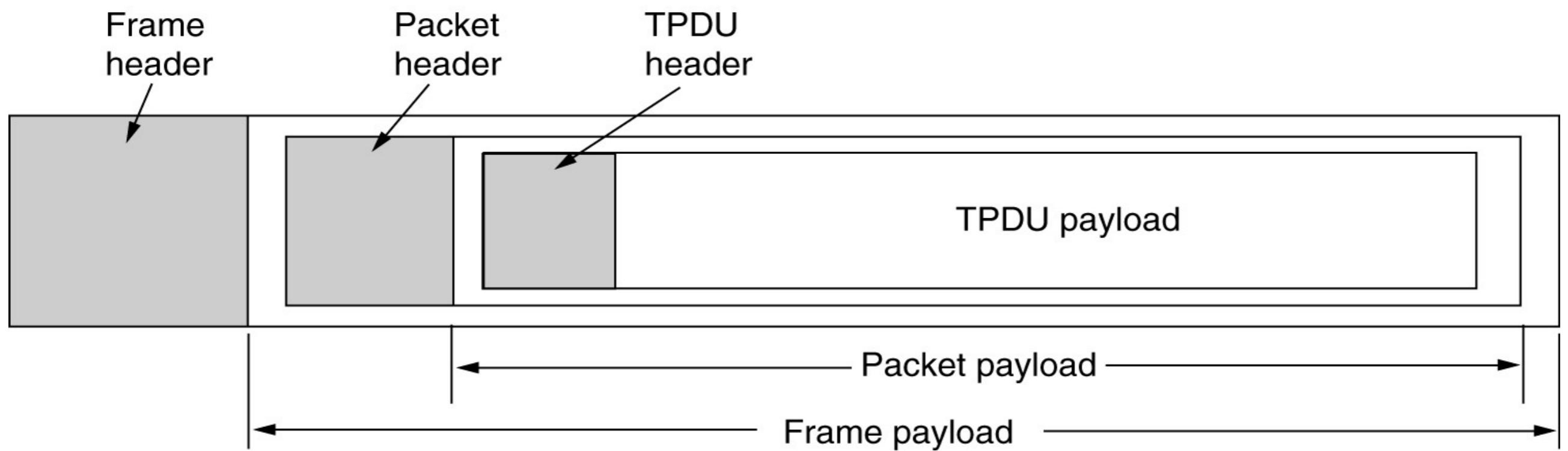
Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h <sub>n</sub> > ... </h <sub>n</sub> >	Delimits a level <i>n</i> heading
<b> ... </b>	Set ... in boldface
<i> ... </i>	Set ... in italics
<center> ... </center>	Center ... on the page horizontally
<ul> ... </ul>	Brackets an unordered (bulleted) list
<ol> ... </ol>	Brackets a numbered list
<li>	Starts a list item (there is no </li>)
 	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a Horizontal rule
	Displays an image here
<a href="..."> ... </a>	Defines a hyperlink

A selection of common HTML tags. some can have additional parameters.

# Résumé Cours

Eléments pour vérifier votre compréhension

# Network control and payload data encapsulation



The nesting of PDUs, packets, and frames.

# Couche Physique

- **Paire torsadée**
  - Telephone fixe, ADSL
  - LAN Ethernet
- **Fibre Optique**
  - WAN
- **Ondes Electromagnétique**
  - WiFi
  - Telephonie mobile 3G 4G
  - IoT
- Signal,
- Modulation,
- Codage,
- Symboles.

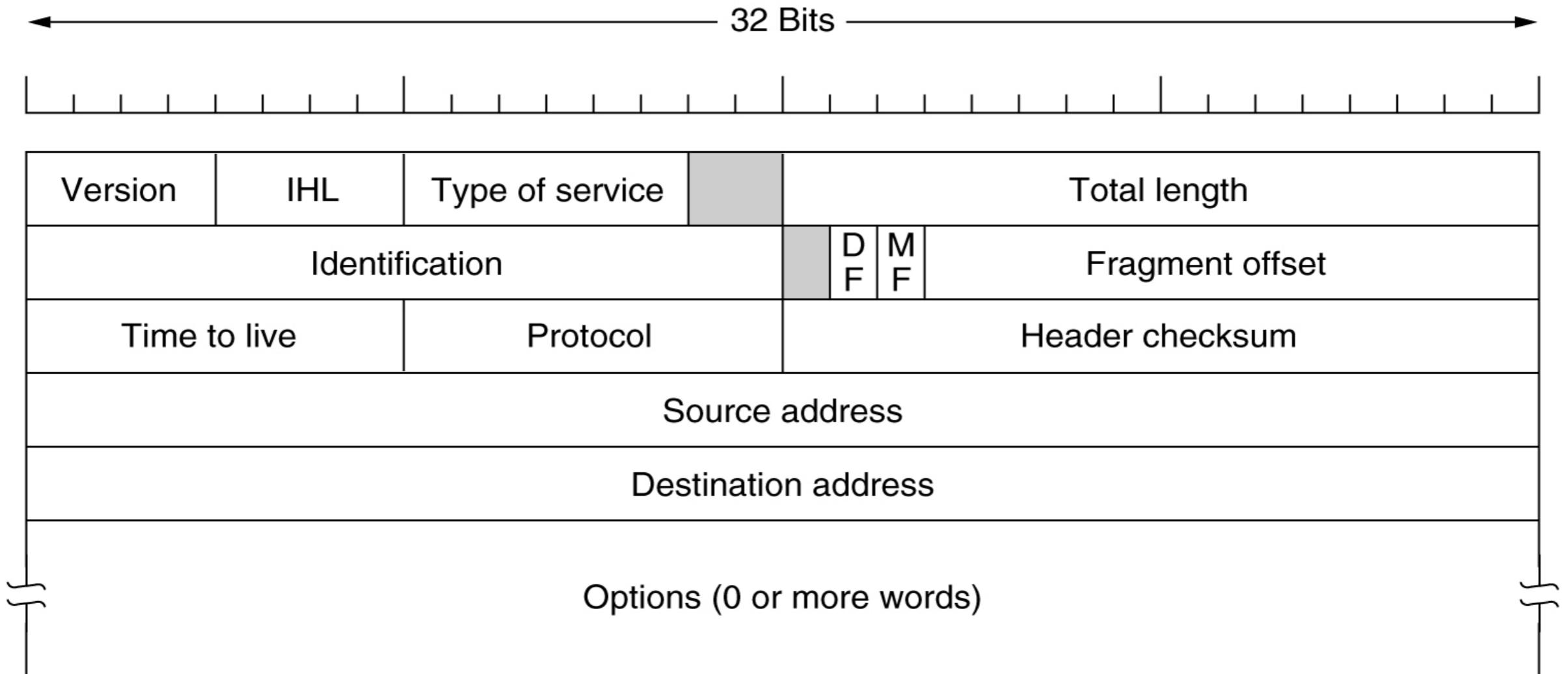
# Couche Liaison

- **MAC (Media Access Control)**
  - protocole de contrôle d'accès aux réseaux partagés
  - adressage physique des stations sur un même réseau point local
- **LLC (Logical Link Control)**
  - trame
  - détection d'erreur
- **Exemple : Ethernet**
  - trames 1500 octets
  - CRC 32 bits
  - adresses MAC de 48 bits a:b:c:d:e:f
  - CSMA (Carrier Sense Multiple Access)
    - » détecte quand quelqu'un émet sur le réseau
    - » suite à une collision on réémet la trame après attente aléatoire pour décaler
- **ARP (Address Resolution Protocol)**
  - Protocole pour obtenir les @physiques (MAC) à partir des @IP

# Couche Réseau

- **IPv4 (Internet Protocol version4)**
  - protocole pour envoyer des données d'une source à une adresse destination au travers de multiples réseaux interconnectés
  - adresses à 4 octets : a.b.c.d
  - adresses de réseau en notation CIDR : a.b.c.d/e
    - » les e bits de poids fort sont l'adresse du (sous-)réseau
    - » **netmask(a.b.c.d/e)** : les e bits p.f. à 1, les autres à 0
    - » l'adresse u.v.w.z **appartient** au réseau a.b.c.d/e si  $(u.v.w.z \text{ ET netmask}) = a.b.c.d$
    - » **localhost** : 127.0.0.0/8 est l'adresse de la station même
    - » **adresse broadcast** : a.b.c.d OU NOT(netmask)  
message à l'adresse broadcast est envoyé à toutes les stations du réseau
    - » **subnetting** : sur un réseau a.b.c.d/e on peut définir des sous-réseaux a.b.c.d/f, f > e
  - Time to Live : nombre de routeurs traversés, décrémenté par chaque routeur
  - fragmentation : paquets trop longs pour la MTU d'un réseau sont fragmentés/rassemblés si besoin
  - protocole non-fiable (best-effort), sans garantie d'ordonnancement, ...

## The IP Protocol

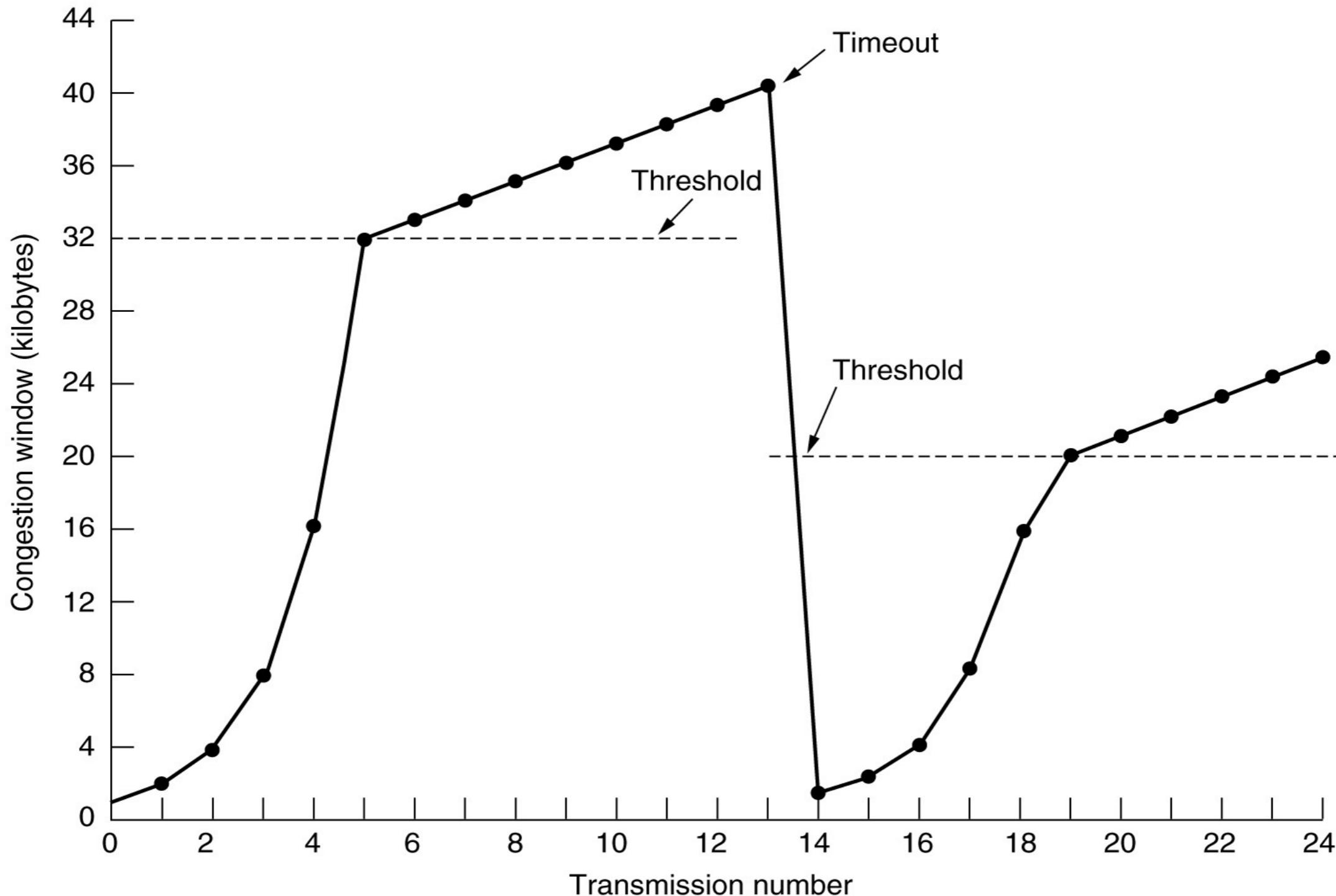


## The IPv4 (Internet Protocol) header.

# Couche Transport

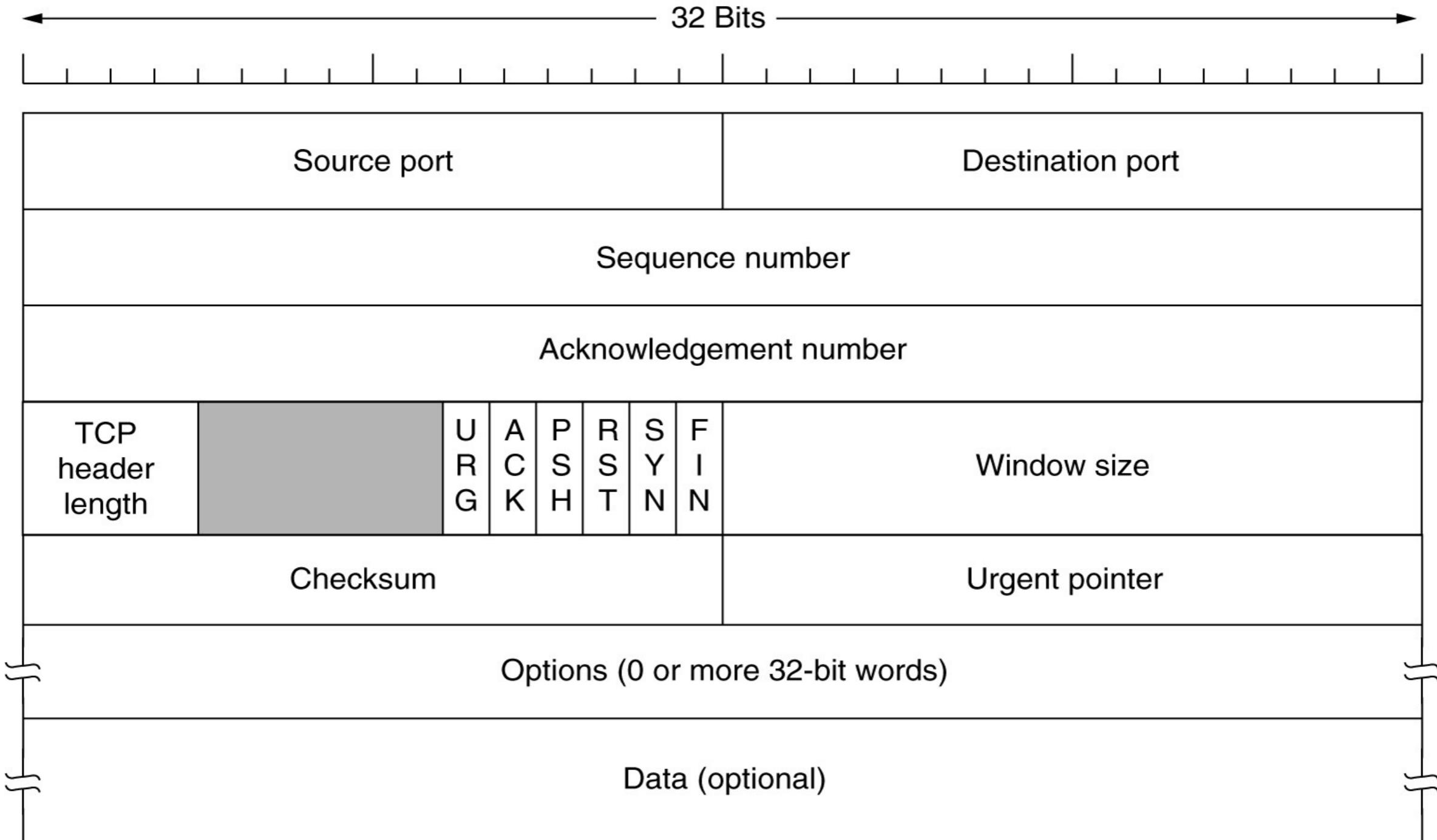
- **TCP (Transmission Control Protocol)**
  - avec numéro de port de source et destination
  - protocole fiable pour envoyer des données en un flux d'octets
    - » avec connexion, drapeaux SYN, ACK,
    - » en séquence ordonnancement garantit pour les applications ,
    - » détection d'erreurs et paquets perdus, ré-émissions si nécessaire
  - Fenêtre glissante d'anticipation
    - » contrôle de flux par le récepteur
    - » envoie des paquets successifs avec ACK cumulatifs
    - » Maximise le débit
  - Fenêtre de congestion
    - » contrôle de congestion avec seuil
    - » Mécanismes de slow-start+montée rapide+montée lente
    - » Répartit la capacité du réseau équitablement entre chaque flux
- **UDP (User Datagram Protocol)**
  - avec numéro de port de source et destination
  - protocole non-fiable pour envoyer des données paquets (datagram)

# TCP Congestion Control



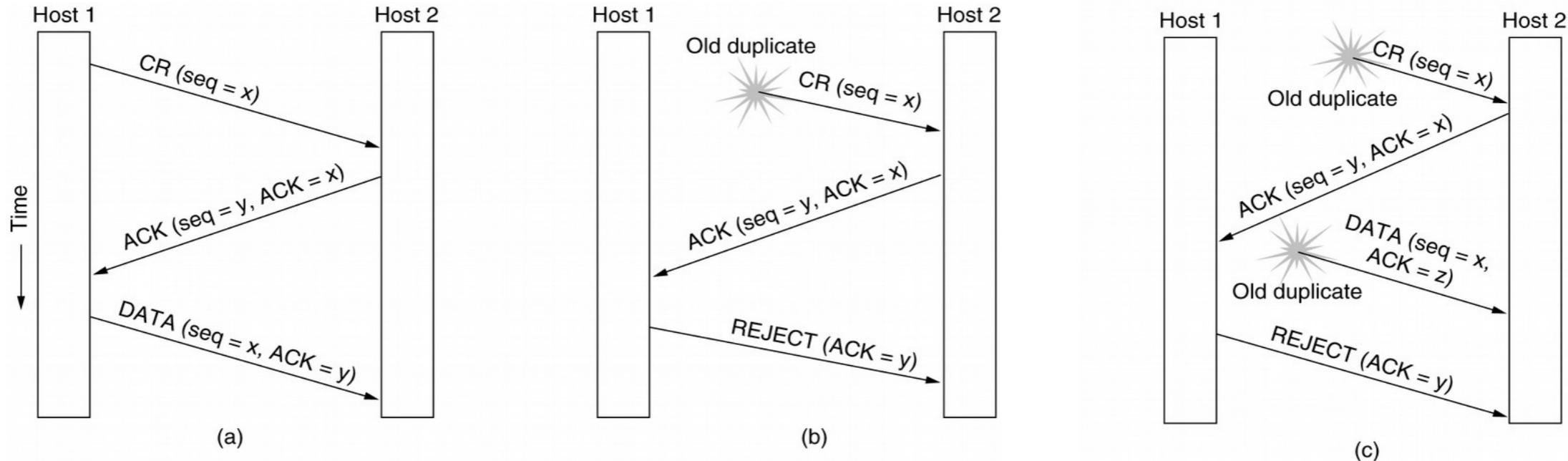
An example of the Internet congestion algorithm.

# The TCP Segment Header



TCP Header.

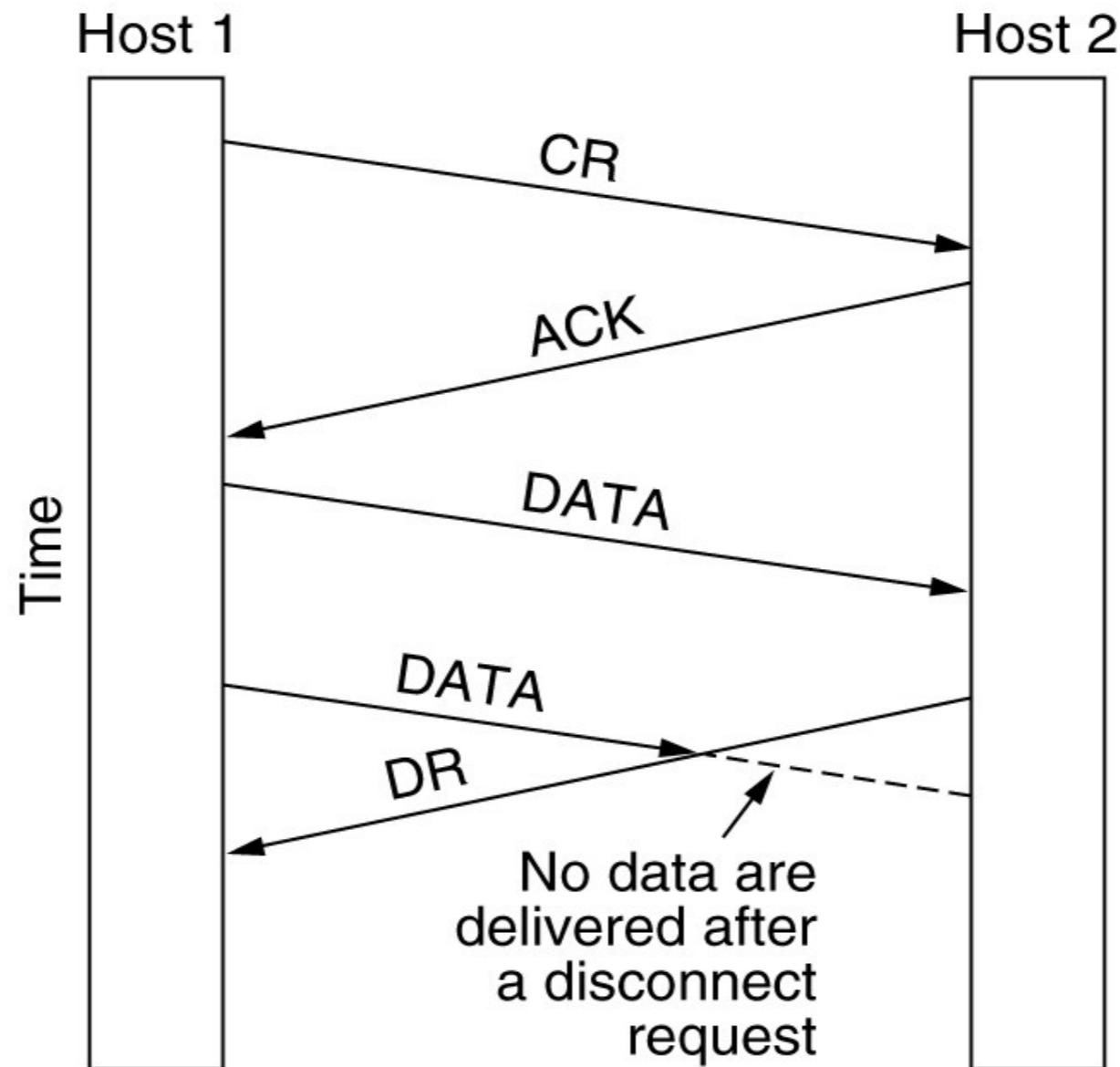
# Connection Establishment Issues



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.

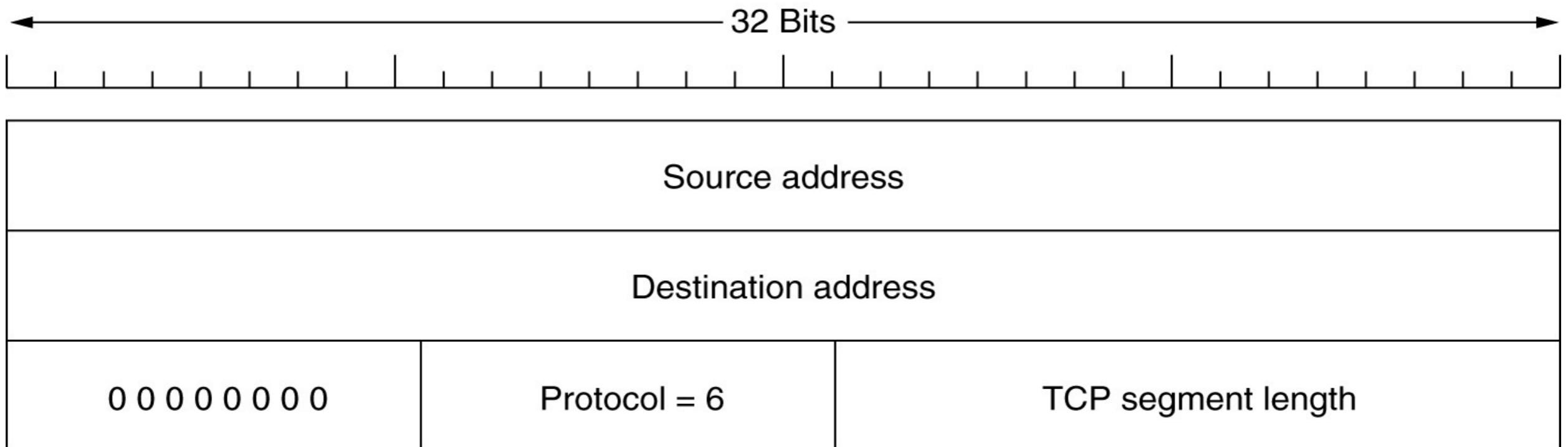
- (a) Normal operation,
- (b) Old CONNECTION REQUEST appearing out of nowhere.
- (c) Duplicate CONNECTION REQUEST and duplicate ACK.

# Connection Release Issues



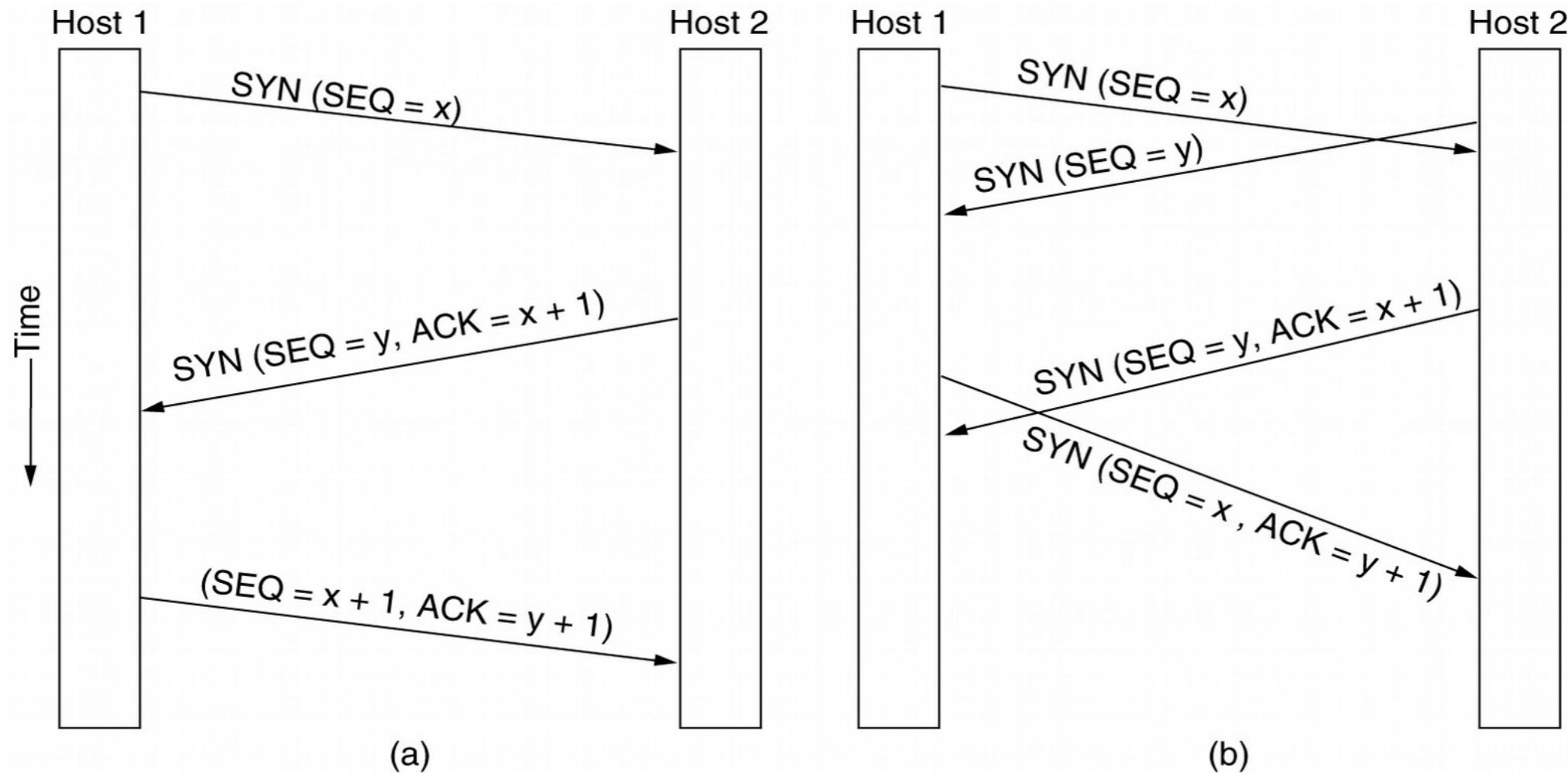
Abrupt disconnection with loss of data.

# The TCP Segment Header (2)



The pseudoheader included in the TCP checksum.

# TCP Connection Establishment



(a) TCP connection establishment in the normal case.

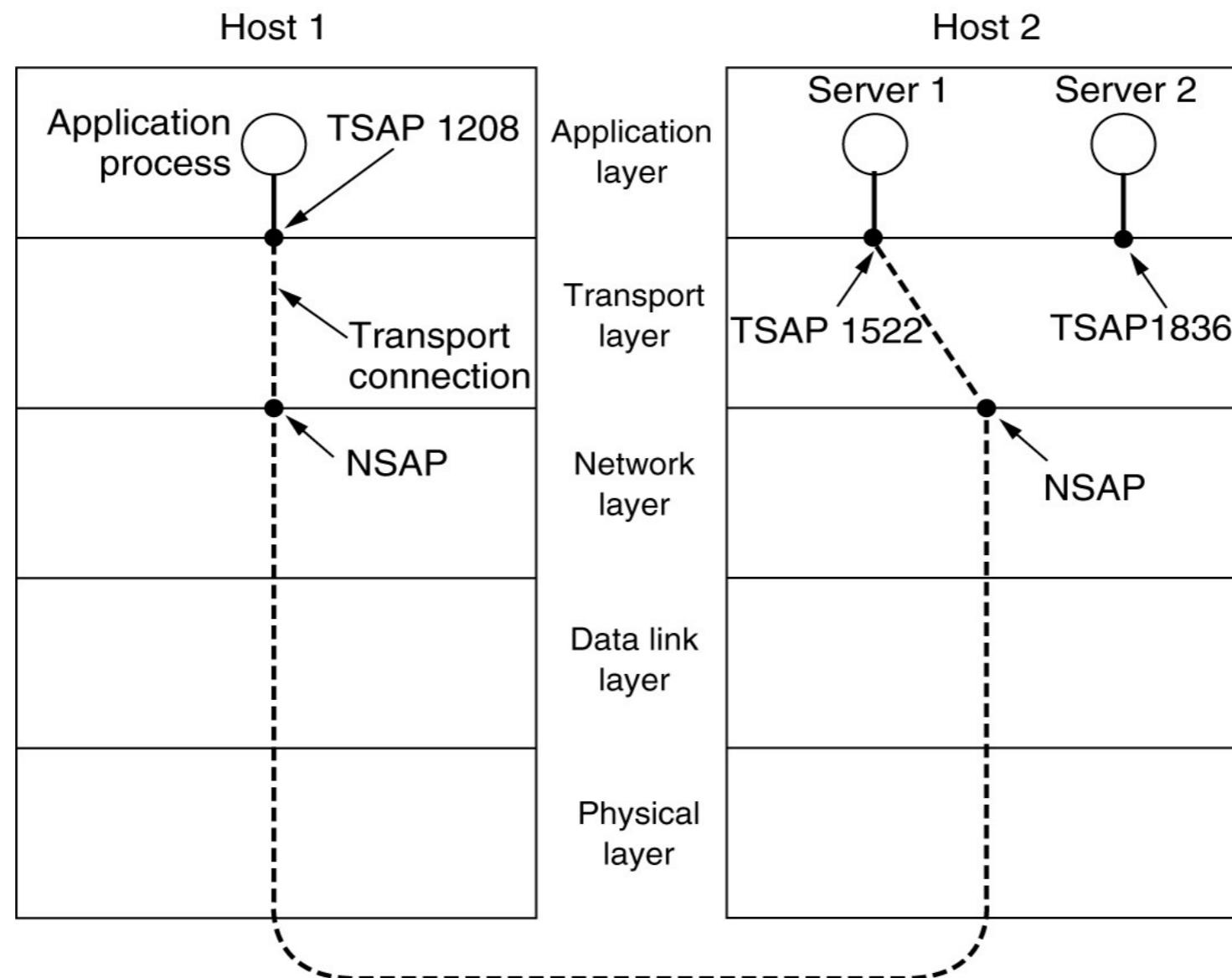
(b) Call collision.

# TCP Connection Management Modeling

<b>State</b>	<b>Description</b>
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

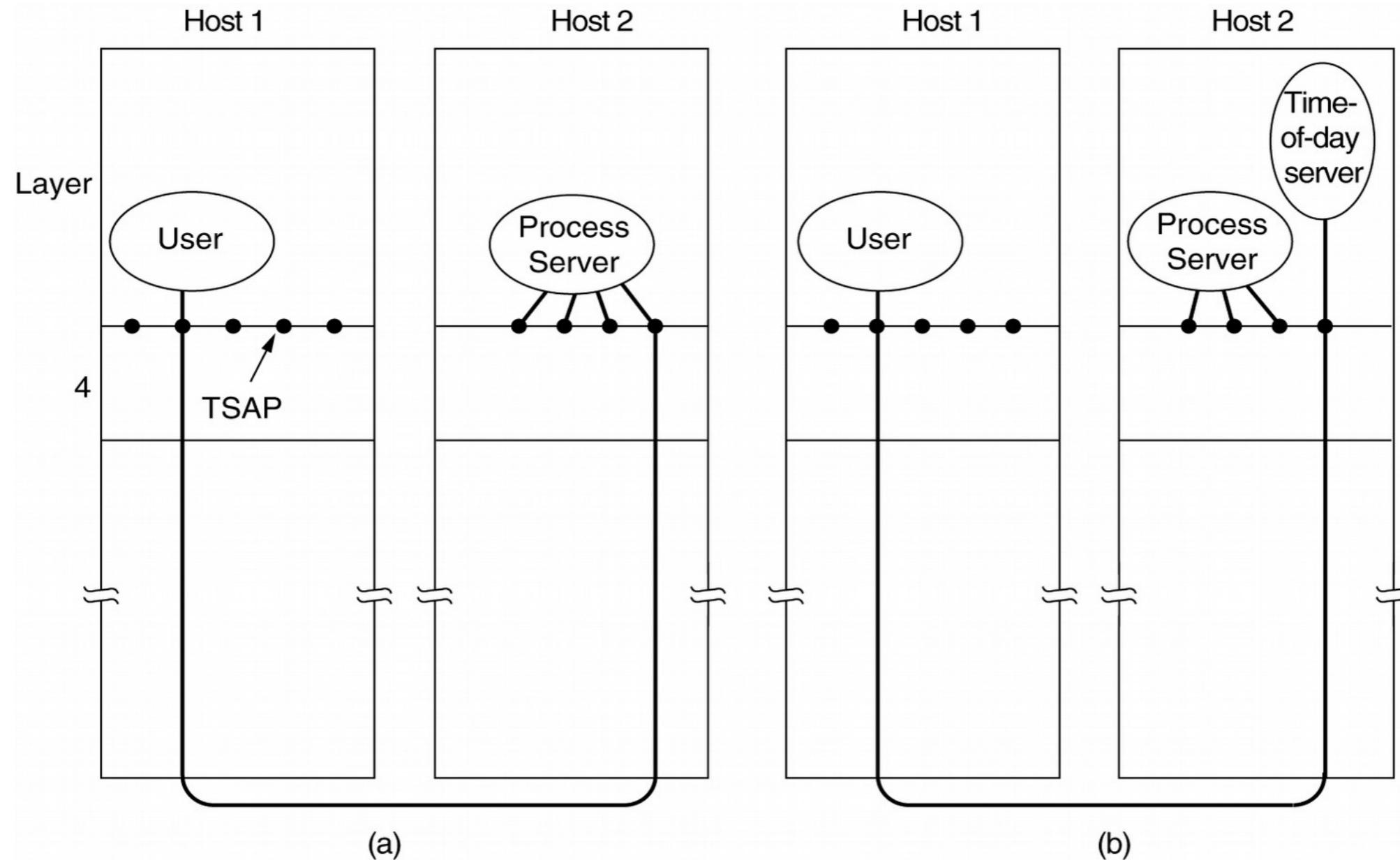
The states used in the TCP connection management finite state machine.

# Addressing (TSAP, NSAP)



TSAPs, NSAPs and transport connections.

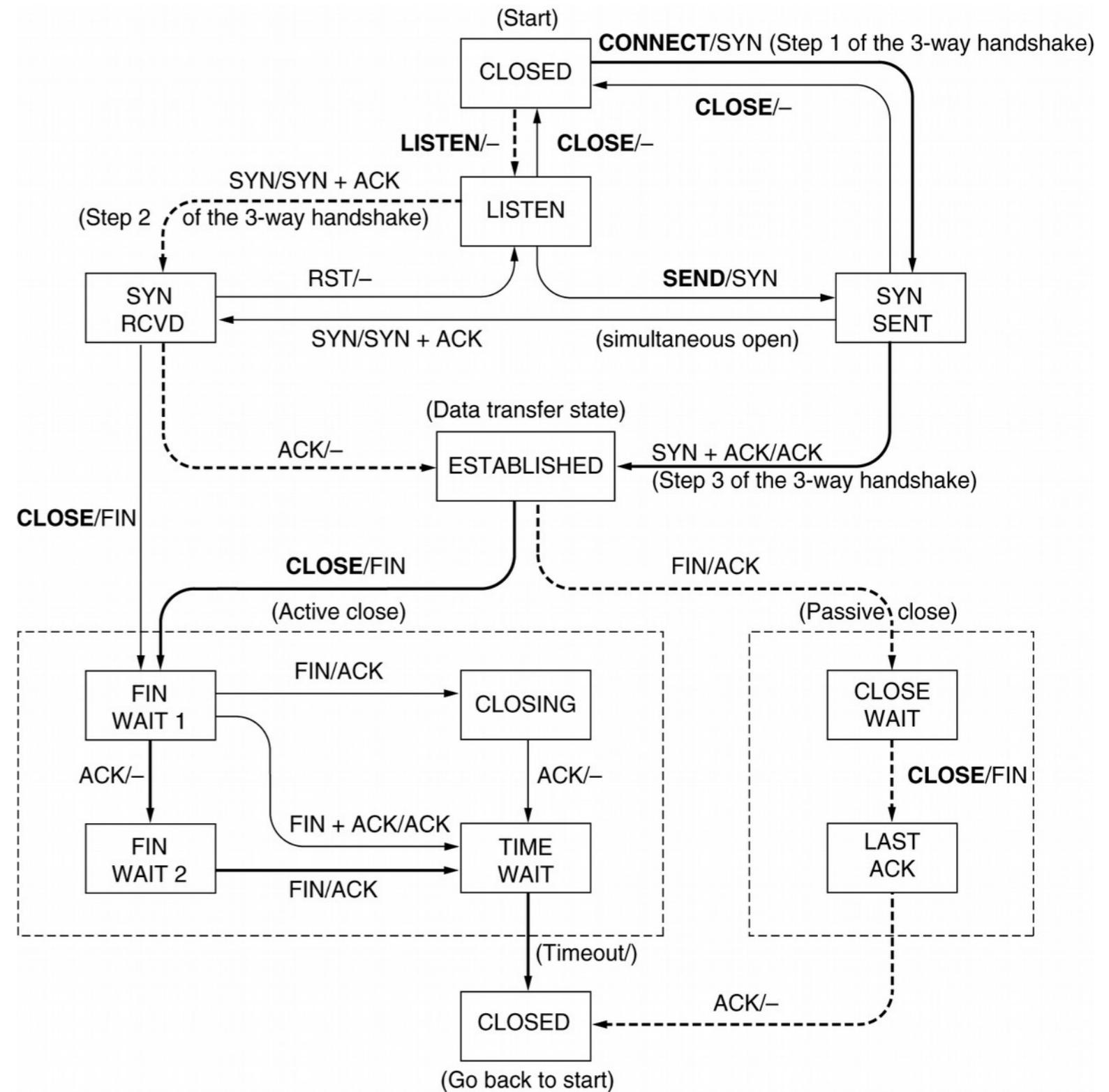
# Connection Establishment



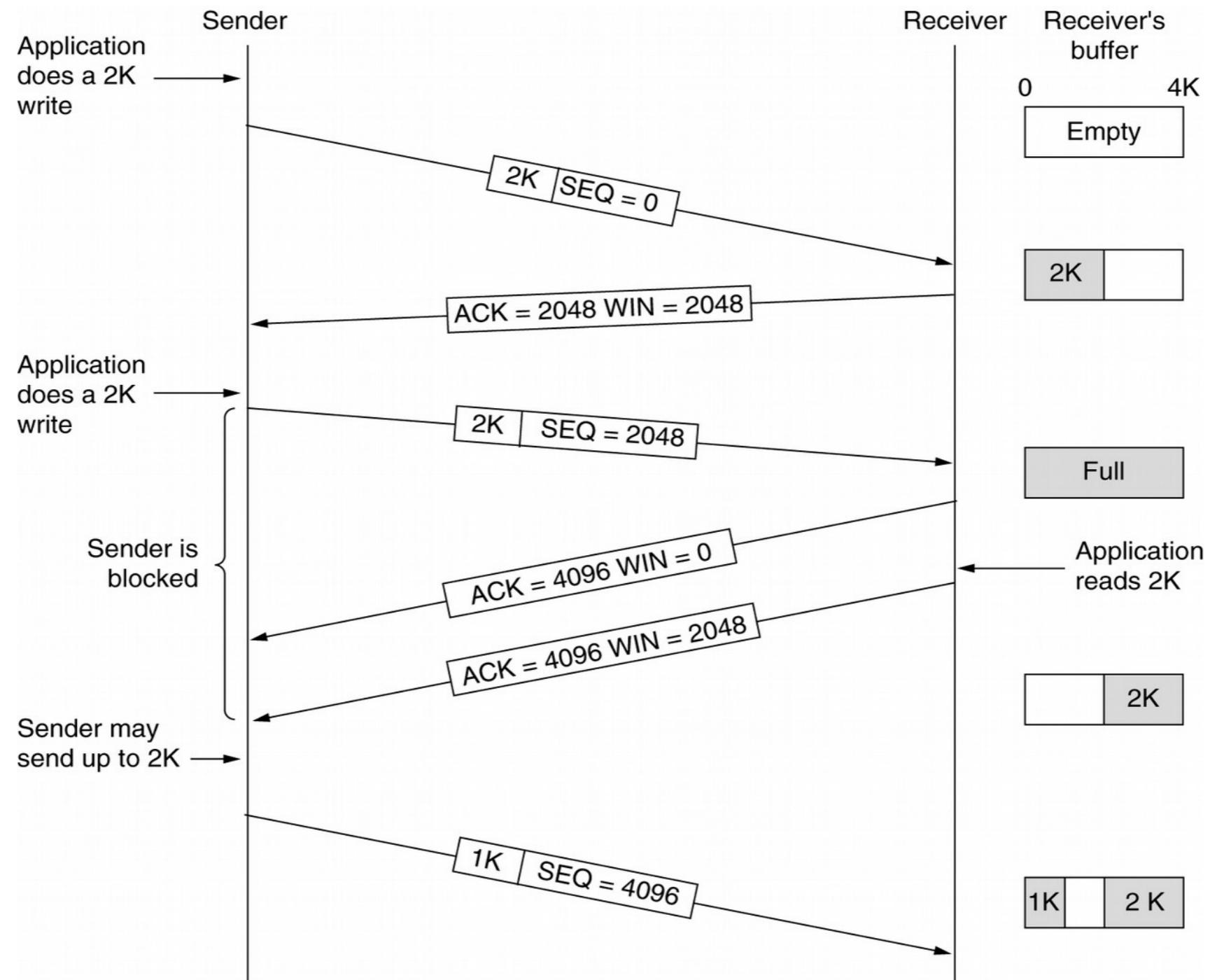
How a user process in host 1 establishes a connection with a time-of-day server in host 2.

# TCP Connection Management Modeling (2)

TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.



# TCP Transmission Policy



Window management in TCP.

# Some protocols on top of the TCP service

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

# **Les sockets en C sous UNIX pour la programmation dans les réseaux TCP/IP**

**(from UNSA, T Tari, Tanenbaum)**

# Fonctionnalité client/serveur des protocoles transport d'Internet

- Protocoles client/serveur:
  - Le serveur se met en attente de demandes
  - Le client initie le dialogue par une demande
- Problème : faire communiquer 2 processus sur des hôtes différents
  - Interface des “sockets”
  - 1 service : 1 adresseIP + 1 numéro de port = un point d'entrée (SAP) prédéfinit sur un système connecté
  - 1 processus est associé à un port
- Côté serveur: ports réservés pour les applications standards
  - cf Fichier /etc/services
  - Exemple: HTTP port 80 en TCP
- Côté client: ports alloués dynamiquement
- Multiplexage du réseau (@IP) vers les applications:
- Adresses Internet source et destination, numéros de ports source et destination
  - construction des entêtes réseau (IP) et transport (TCP ou UDP)

## **SOMMAIRE**

**I/ DEFINITION**

**II/ DESCRIPTION API Socket**

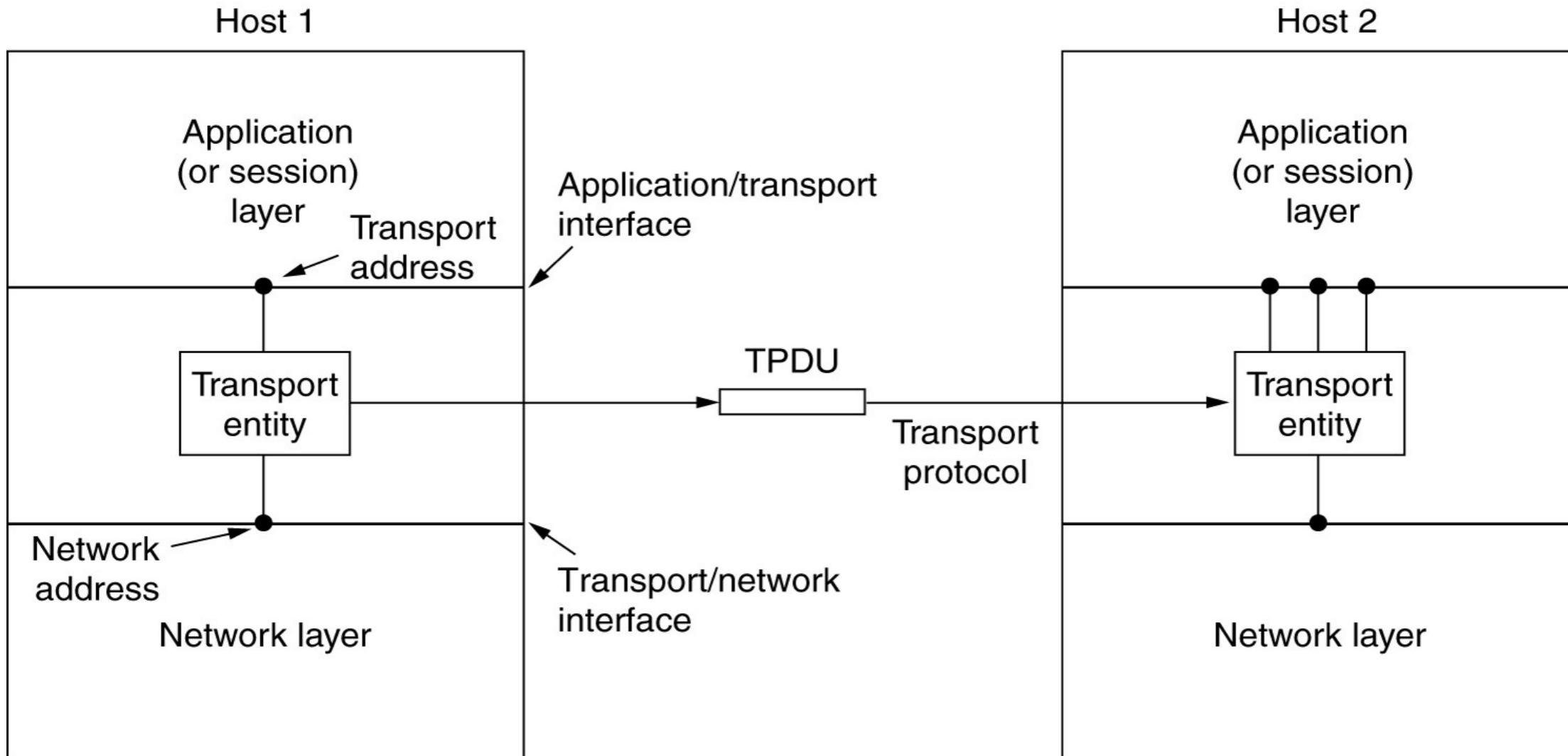
**III/ SOCKETS DU DOMAINE INTERNET**

**1/ Mode déconnecté**

**2/ Mode connecté**

**IV/ SOCKETS DU DOMAINE UNIX**

# SAP Provided by Transport to the Sockets



# Transport Generic Service Primitives

<b>Primitive</b>	<b>Packet sent</b>	<b>Meaning</b>
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

The primitives for a simple transport service.

## I/ DEFINITION

**Les Sockets constituent une des A.P.I. (Application Program Interface) des protocoles de communication au niveau transport**

**Interface application / réseaux Internet au niveau transport**

# OBJECTIFS DES SOCKETS

- ↳ Définir une extrémité de connexion, un point de contact bidirectionnel où un processus peut émettre et recevoir des données selon un protocole choisi.
- ↳ Permettre à plusieurs processus, même distants de communiquer.
- ↳ S'approcher de l'A.P.I. Fichier d'UNIX pour utiliser les mêmes primitives (orthogonalité des concepts) :  
`read/receive, write/send, ioctl, select ...`

## **Une socket est définie par:**

### **- son domaine**

- Local (AF\_UNIX), permet à plusieurs processus de communiquer sur la même machine.
- Internet (AF\_INET), idem au travers d'un réseau IP.

### **Autres réseaux:**

- XeroxNS (AF\_NS)
- SNA – IBM (AF\_SNA)
- AppleTalk (AF\_APPLETALK)

## **Une socket est définie par:**

**- son type**

**Il détermine le type de communication:**

- ↳ **Connecté : Etablissement d'une connexion + fiabilité des échanges + ordre de séquencement ...**
- ↳ **Non-connecté : best effort**

# Socket types

- **Mode non-connecté (SOCK\_DGRAM)**

**Envoie de données par datagrammes indépendant les uns des autres. L'adresse de destination est contenu dans chaque datagramme.**

**Analogue au service postal.**

→ **Support par UDP.**

- **Mode connecté (SOCK\_STREAM)**

**Réalise virtuellement un canal de communication permanent entre le client et le serveur. Associé à un flot de données non interprété.**

**Analogue au téléphone analogique mais géré en logiciel.**

→ **Support par TCP.**

# Socket types

## Autres types moins répandus

- Mode accès direct au couches basses (**SOCK\_RAW**).
- Mode format structuré autre qu'Internet (**SOCK\_SEQPACKET**).

## Combinaison DOMAINE – TYPE

TYPES	DOMAINES	
	AF_UNIX	AF_INET
<b>SOCK_STREAM</b>	OUI	TCP
<b>SOCK_DGRAM</b>	OUI	UDP
<b>SOCK_RAW</b>	NON	IP
<b>SOCK_SEQPACKET</b>	NON	NON

# Berkeley (Unix) Sockets : a programming interface for layer 4

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

The socket primitives for TCP.

# Socket Programming Example: Internet File Server Client side

Client code using sockets.

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345          /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096              /* block transfer size */

int main(int argc, char **argv)
{
    int c, s, bytes;
    char buf[BUF_SIZE];           /* buffer for incoming file */
    struct hostent *h;             /* info about server */
    struct sockaddr_in channel;   /* holds IP address */

    if (argc != 3) fatal("Usage: client server-name file-name");
    h = gethostbyname(argv[1]);      /* look up host's IP address */
    if (!h) fatal("gethostbyname failed");

    s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s < 0) fatal("socket");
    memset(&channel, 0, sizeof(channel));
    channel.sin_family= AF_INET;
    memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
    channel.sin_port= htons(SERVER_PORT);

    c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
    if (c < 0) fatal("connect failed");

    /* Connection is now established. Send file name including 0 byte at end. */
    write(s, argv[2], strlen(argv[2])+1);

    /* Go get the file and write it to standard output. */
    while (1) {
        bytes = read(s, buf, BUF_SIZE);      /* read from socket */
        if (bytes <= 0) exit(0);            /* check for end of file */
        write(1, buf, bytes);              /* write to standard output */
    }
}

fatal(char *string)
{
    printf("%s\n", string);
    exit(1);
}
```

# Socket Programming Example: Internet File Server Server side

```
#include <sys/types.h>                                /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 12345                               /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                                    /* block transfer size */
#define QUEUE_SIZE 10
int main(int argc, char *argv[])
{
    int s, b, l, fd, sa, bytes, on = 1;
    char buf[BUF_SIZE];                                 /* buffer for outgoing file */
    struct sockaddr_in channel;                         /* hold's IP address */
    /* Build address structure to bind to socket. */
    memset(&channel, 0, sizeof(channel));             /* zero channel */
    channel.sin_family = AF_INET;
    channel.sin_addr.s_addr = htonl(INADDR_ANY);
    channel.sin_port = htons(SERVER_PORT);

    /* Passive open. Wait for connection. */
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);    /* create socket */
    if (s < 0) fatal("socket failed");
    setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

    b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
    if (b < 0) fatal("bind failed");

    l = listen(s, QUEUE_SIZE);                          /* specify queue size */
    if (l < 0) fatal("listen failed");

    /* Socket is now set up and bound. Wait for connection and process it. */
    while (1) {
        sa = accept(s, 0, 0);                           /* block for connection request */
        if (sa < 0) fatal("accept failed");
        read(sa, buf, BUF_SIZE);                        /* read file name from socket */
        /* Get and return the file. */
        fd = open(buf, O_RDONLY);                      /* open the file to be sent back */
        if (fd < 0) fatal("open failed");

        while (1) {
            bytes = read(fd, buf, BUF_SIZE);           /* read from file */
            if (bytes <= 0) break;                     /* check for end of file */
            write(sa, buf, bytes);                     /* write bytes to socket */
        }
        close(fd);                                     /* close file */
        close(sa);                                     /* close connection */
    }
}
```

Server code using sockets.

# CARACTERISTIQUES SYSTEMES DES SOCKETS

On accède à une socket par un descripteur de socket, assimilable à un descripteur de fichier afin d'assurer la compatibilité des I/O d'UNIX.

→ Celui-ci est donc dans la table des descripteurs de l'OS.

- Possibilité de rediriger des entrées ou des sorties dans une socket.
- Les fils du processus héritent de la table des descripteurs, donc des sockets.
  - ⇒ Serveur concurrent en mode connecté. Le processus père accepte les connexions, chaque fils créé gère un client via la socket.

# CARACTERISTIQUES SYSTEMES DES SOCKETS (suite)

- Utilisent les primitives et appels systèmes classiques, assurent un polymorphisme des I/O et une abstraction des caractéristiques réseaux.

**Exemple pratique: select()**

=> On peut scruter plusieurs I/O sans s'occuper de la nature du canal de communication grâce aux descripteurs.

# CARACTERISTIQUES SYSTEMES DES SOCKETS (suite)

**Une socket est associée à 2 mémoires tampons gérées par le noyau UNIX:**

- Emission
- Réception

**La gestion des tampons dépend du protocole:**

- ↳ de la couche TRANSPORT pour le domaine INTERNET.
- ↳ du système pour le domaine UNIX.

**Le Noyau UNIX décharge le programmeur des caractéristiques du protocoles désirés. Les Tampons sont lus et écrits de façon concurrente par la socket (l'application) et le noyau.**

# CARACTERISTIQUES SYSTEMES (suite)

## Déroulement de l'écriture

- **Le noyau recopie les données placées dans l'espace mémoire du processus vers le tampon d'émission.**
- **Si le tampon est plein,**
  - **l'écriture est bloquante,**
  - **le processus est endormi en attendant que le tampon d'émission se vide.**
- **Sortie de l'écriture quand la copie des données dans le tampon est achevée**

# CARACTERISTIQUES SYSTEMES (suite)

## Déroulement de la lecture

**Le noyau recopie les données du tampon de réception dans l'espace mémoire alloué pour la variable du processus.**

**Par défaut, la lecture est bloquante sur un tampon vide.**

**On peut rendre la lecture non bloquante :**

⇒ **lecture tampon vide renvoie -1**

⇒ **Il faut alors scruter le descripteur de la socket pour ne lire le tampon que lorsque les données sont présentes.**

# CARACTERISTIQUES SYSTEMES (suite)

## Exemple : pour rendre la lecture non bloquante

```
int optSock;  
  
fcntl(sd, F_GETFL, &optSock); /* Lit options sur la socket */  
optSock |= O_NDELAY;           /* Définit options */  
fcntl(sd, F_SETFL, &optSock); /* Applique options */
```

**sd = Descripteur de la socket**

## Particularités des Network I/O vs File I/O

- En mode connecté, non symétrie de la relation client serveur, le client provoque la connexion, le serveur la gère. Comportements différents.
- En mode déconnecté, une opération (une émission) peut cibler plusieurs processus (multi ou broadcast)

## Particularités des Network I/O vs File I/O

- Une connexion réseau est définie par plus d'éléments qu'un descripteur de fichier:  
(protocole, adresse locale, port processus local, adresse distante, port processus distant)  
Exemple: (tcp, 128.10.0.1, 2500, 128.10.0.8, 9000)

Avec le protocole TCP (Transport Control Protocol), les adresses sont des adresses IP (Internet Protocol), les processus sont atteints par le numéro de port qui leur est attaché.

Un port par processus permet de multiplexer un unique support réseau physique entre plusieurs processus communiquants.

## II/ DESCRIPTION DE L'API SOCKET

**Librairies indispensables:**

**sys/socket.h**

**sys/types.h**

**arpa/inet.h**

**netinet/in.h**

**netdb.h**

## 1/ Création d'une socket

```
int socket(int domaine, int type, int protocole)
```

Renvoie: le numéro de descripteur de la socket,  
-1 en cas d'échec.

domaine: AF\_UNIX, AF\_INET, AF\_ISO, AF\_NS...

type: SOCK\_DGRAM, SOCK\_STREAM

Protocole: 0 pour laisser le système choisir en  
fonction du couple domaine/type

## 2/ Suppression d'une socket

A/ `int close(int sockDesc)`

*Libère les ressources de la socket si elles ne sont plus partagées. En type SOCK\_DGRAM, vide le tampon d'émission.*

`sockDesc`: descripteur de la socket à fermer et à supprimer de la table des descripteurs de fichier.

## 2/ Suppression d'une socket

B/ `int shutdown(int sockDesc, int sens)`

*Permet de fermer partiellement le descripteur de la socket en type **SOCK\_STREAM**, donc full duplex vu comme deux canaux simplex.*

`sockDesc`: descripteur de la socket.

`sens`: 0 ⇒ plus de lecture, `read/recv` renvoient 0  
1 ⇒ plus d'écriture, `write/send` provoque  
    `SIGPIPE`  
2 ⇒ plus de lecture ni d'écriture.

### 3/ Attachement à une adresse

```
int bind(int  sockDesc, struct sockaddr*  adSock,  
        size_t tailleAd)
```

*Une fois créée, la socket n'est visible que par le processus local. bind attache la socket à une adresse extérieure visible par autres processus. NB une machine peut avoir plusieurs cartes réseaux ...*

*L'adresse extérieure dépend du domaine de la socket:*

- en local (*AF\_UNIX*), c'est un fichier.
- en Internet (*AF\_INET*), c'est le nom de la machine plus un numéro de port.

*sockDesc*: descripteur de la socket à attacher.

*adSock*: structure qui contient les éléments de l'adresse externe.

*tailleAd*: taille en octets de *adSock*,  
*sizeof(adSock)*

## Autres fonctions utiles

```
struct hostent* gethostbyname(char* nomMachine)
```

*Alloue et renvoie l'adresse d'une structure de type struct hostent. On peut ensuite adresser le paquet sur le réseau.*

```
struct hostent
{
    char* h_name;          /* nom canonique de la machine */
    char** h_aliases;      /* tableau des autres nom d'alias */
    int h_addrtype;        /* domaine de l'adresse*/
    int h_length;           /* longueur de l'adresse */
    char** h_addr_list;    /* tableau des adresses IP */
}
#define h_addr h_addr_list[0]
```

## Autres fonctions utiles

```
int setsockopt(int sockDesc, int niveau,  
               int option, void* valeurOpt,  
               int longValeurOpt)
```

*Modifie les caractéristiques d'une socket. Par exemple, change la taille des tampons, autorise le broadcast...*

**Plusieurs niveau de paramétrage d'une socket:**

- niveau socket (**SOL\_SOCKET**)
- niveau TCP (**IPPROTO\_TCP**)

# Paramètres utiles

- Niveau Socket (`SOL_SOCKET`)
  - `SO_BROADCAST` : autorise la diffusion (`SOCK_DGRAM/AF_INET`)
  - `SO_DONTROUTE` : court-circuite le routage standard (`SOCK_STREAM/AF_INET`)
  - `SO_KEEPALIVE` : teste l'activité sur une connexion (`SOCK_STREAM/AF_INET`)
  - `SO_LINGER` : temps accordé au tentative d'envoi (`SOCK_STREAM/AF_INET`)
  - `SO_OOBINLINE` : les msg `MSG_OOB` sont placés dans le tampon de réception du socket récepteur (`SOCK_STREAM/AF_INET`)
  - `SO_REUSEADDR` : réutilisation d'une adresse locale pour réaliser la connexion
  - `SO_TYPE` : type du socket
  - `SO_RCVBUF` : taille du buffer de réception du socket
  - `SO_SNDBUF` : taille du buffer d'émission du socket
- Niveau TCP (`IPPROTO_TCP`)
  - `TCP_NODELAY` : force l'envoi des données sans bufferisation (`SOCK_STREAM/AF_INET`)
  - `TCP_MAXSEG` : connaître la taille maximale d'un segment TCP (`SOCK_STREAM/AF_INET`)

## Fonctions d'arrangement de format (marshaling)

**La représentation d'un nombre est dépendante de la machine (architecture interne):**

#IP	Valeur Binaire	Format Endian
132.227.70.77	84 C3 46 4D	Big (standard)
132.227.70.77	4D 46 C3 84	Little

**Pour transférer des entiers, il faut donc respecter le standard réseau universel.**

## Fonctions d'arrangement de format

`short htons(short nombre)`

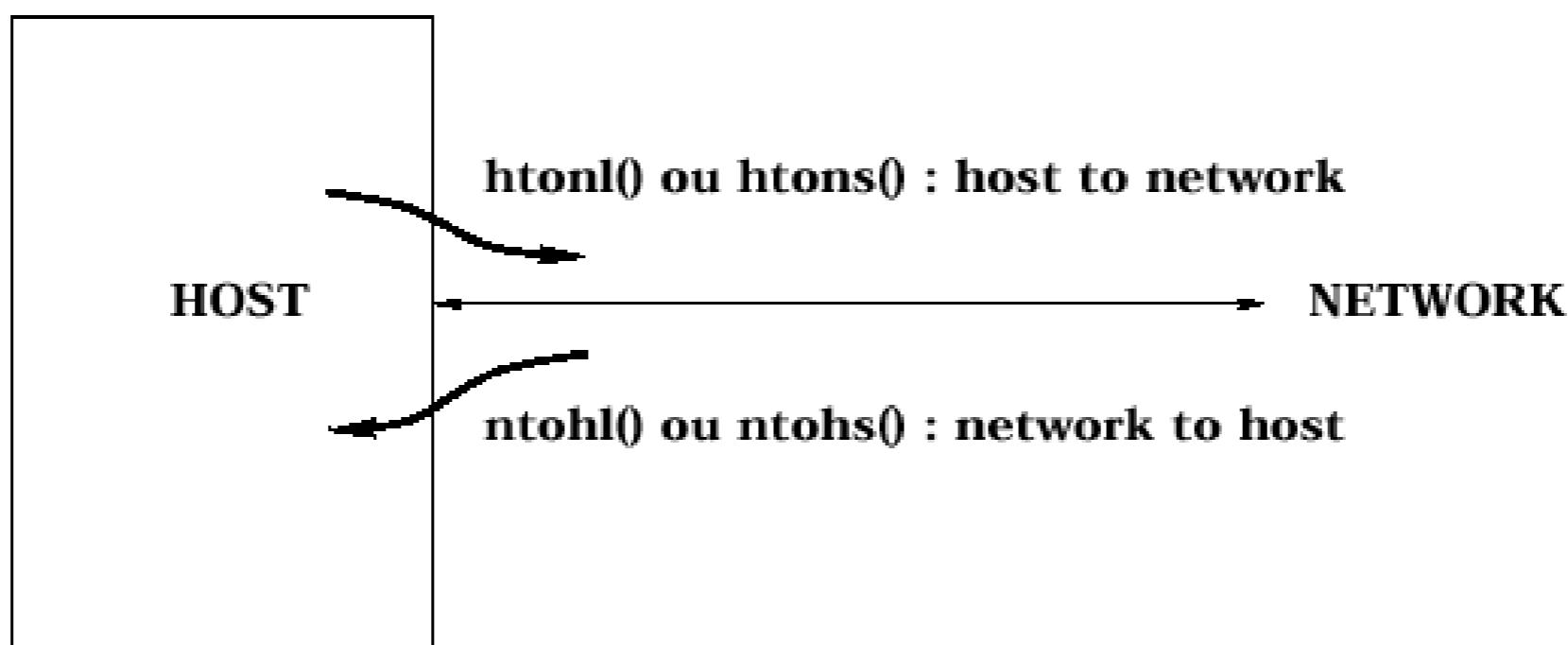
`long htonl(long nombre)`

⇒ convertissent format machine vers format réseau

`short ntohs(short nombreRes)`

`long ntohl(long nombreRes)`

⇒ convertissent format réseau vers format machine



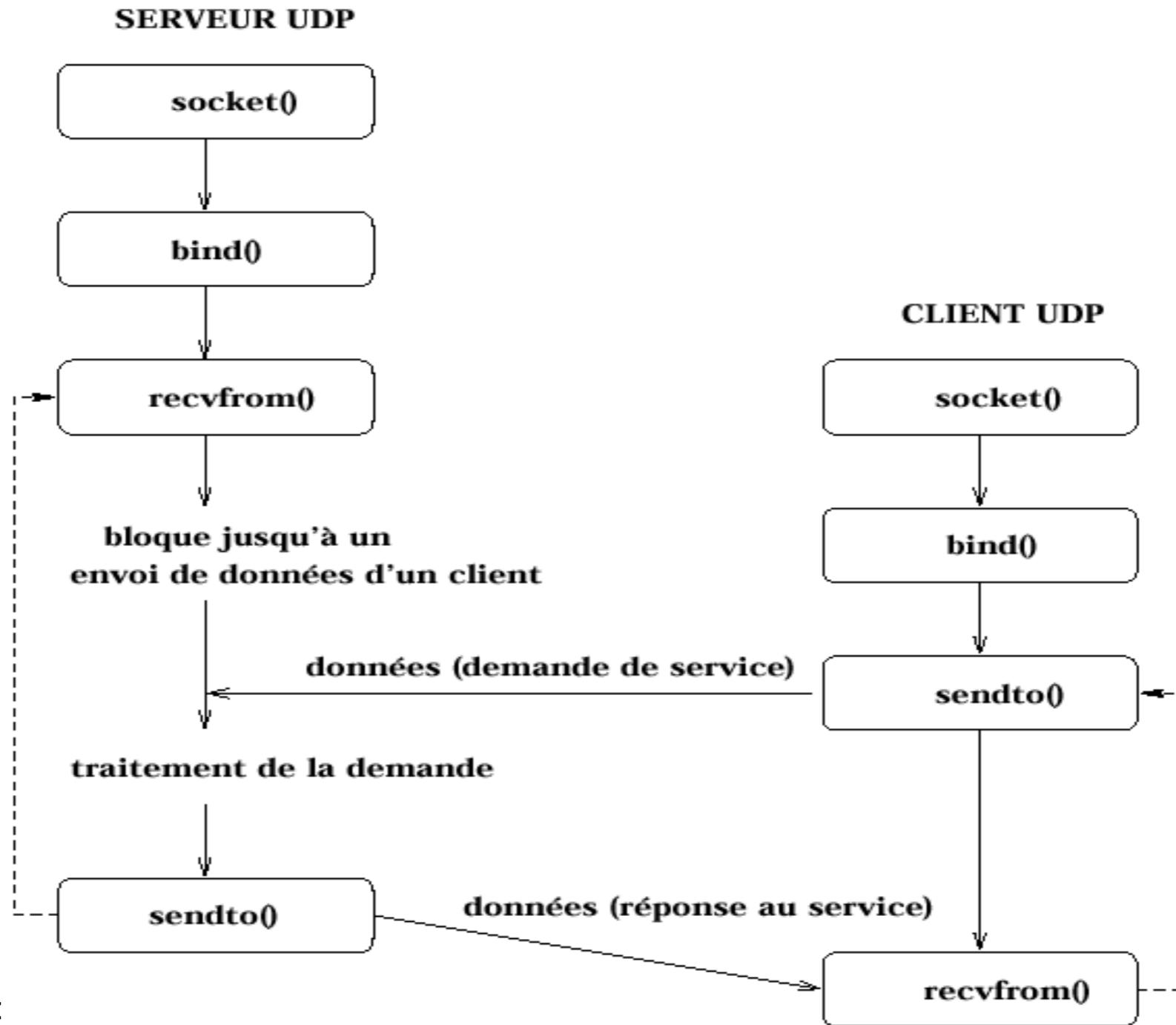
## III/ SOCKET DU DOMAINE INTERNET

### 1/ Mode déconnecté

Permet une communication entre plusieurs processus fonctionnant sur un réseau IP  
Le protocole est UDP

- + Logiciel rapide et petit, pas de surcharge de traffic.
- + Possibilité d'envoyer un paquet à plusieurs destinataires (multicast).
  
- Non fiable, possibilité de perdre des paquets sans le savoir.
- Ordre des paquets peut avoir changé à l'arrivé (si différent routage par paquet ou bufferisation dans un routeur).

## Schéma de communication socket UDP



## FONCTION D'OUVERTURE ET D'ATTACHEMENT D'UNE SOCKET UDP

```
int ouvreSocket(int port)
{
    int skD;
    size_t tailleAd;
    struct sockaddr_in adLocale;

    adLocale.sin_family = AF_INET;           /* Type de la socket (TCP/IP) */
    adLocale.sin_port = htons(port);         /* Affectation du port local */
    adLocale.sin_addr.s_addr = htonl(INADDR_ANY); /* Identificateur de l'hôte */

    skD = socket(AF_INET, SOCK_DGRAM, 0);      /* Crée socket UDP (déconnectée) */

    if (skD == -1)
    {
        perror("Erreur lors de la création de la socket\n");

        return -1;
    }

    tailleAd = sizeof(adLocale);
    retVal = bind(skD, (struct sockaddr*) &adLocale, tailleAd); /* Attache la socket */
    if (retVal == -1)
    {
        perror("Erreur lors du bind\n");
        close(skD);

        return -1;
    }
}
```

## EXEMPLE DE CODE EN UDP

### Envoie d'un long vers un serveur

```
long nb;
long nbNet;
size_t tailleAd;
struct hostent* infosServeur = NULL;
struct sockaddr_in adServeur; /* Structure de l'adresse du serveur */

infosServeur = gethostbyname(argv[1]); /* Récupère infos du serveur */

adServeur.sin_family = infosServeur->h_addrtype; /* Type de la socket du serveur */
adServeur.sin_port = htons(portServeur); /* Port du serveur qui va recevoir datagramme */
memcpy(&adServeur.sin_addr, infosServeur->h_addr, infosServeur->h_length);

skDesc = ouvreSocket(PORTLOCAL);

nbNet = htonl(nb); /* Format machine -> format réseau */

tailleAd = sizeof(adServeur);

RetVal = sendto(skDesc, &nbNet, sizeof(long), 0, (struct sockaddr*) &adServeur, tailleAd);
```

## EXEMPLE DE CODE EN UDP

### Reception d'un long

```
long nb;
long nbNet;
size_t tailleAd;
struct hostent* infosClient = NULL;
struct sockaddr_in adClient; /* Structure de l'adresse du client */
unsigned long adresseClient;

retval = recvfrom(skDesc, &nbNet, sizeof(long), 0,
                  (struct sockaddr*) &adClient, &tailleAd);

adresseClient = adClient.sin_addr.s_addr; /* Adresse IP codée en ulong */
infosClient = gethostbyaddr((char*) &adresseClient, sizeof(long), AF_INET);

printf("**** INFORMATIONS CLIENT ***\n");
printf("Nom: %s \n", infosClient->h_name);
printf("Adresse: %s \n", inet_ntoa(adClient.sin_addr));
printf("Port: %d\n", ntohs(adClient.sin_port));

nb = ntohl(nbNet); /* Format réseau --> format machine */
printf("### Nombre reçu: %ld\n", nb);
```

## EXEMPLE DE CODE EN UDP

### Envoi en broadcast

```
int on;
int portDiff;
struct sockaddr_in adDiffusion;
unsigned long adDiff;

adDiff = inet_addr(argv[2]);           /* Adresse -> broadcast */
printf("Adresse de broadcast = 0x%lx\n", adDiff);

adDiffusion.sin_addr.s_addr = adDiff;    /* Identificateur de l'hote */
adDiffusion.sin_family = AF_INET;        /* Type de la socket du serveur */
adDiffusion.sin_port = htons(portDiff);   /* Port des serveurs qui vont recevoir dgm */

tailleAd = sizeof(adDiffusion);

/* Met la socket en mode broadcast */
on = 1;
setsockopt(skDesc, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on));

sendto(skDesc, &nbNet, sizeof(long), 0, (struct sockaddr*) &adDiffusion, tailleAd);
```

## III/ SOCKET DU DOMAINE INTERNET

### 2/ Mode connecté

**Permet une communication par flux d'octets entre deux processus reliés par un circuit virtuel fiable. C'est le protocole TCP qui est sous-jacent. Le serveur ouvre une socket serveur qui attend des connexions. Dès qu'il y en a une, la fonction accept (bloquante) retourne le descripteur d'une socket de service pour communiquer avec le client.**

- + Fiable.
- + Ordonné.
  
- Consommateur de trafic de contrôle.
- Consommation de ressources mémoire
- Consommateur de cycles CPU pour l'exécution des logiciels de contrôle
- Pas de multi-destinataires.

## Fonction de connexion d'une socket entre le client et le serveur (CLIENT)

```
int connecteSocket(int portLocal, char* nomServeur, int portServeur)
{
    int skD;
    size_t tailleAd;
    struct hostent* infosServeur;      /* Informations du serveur */
    struct sockaddr_in adLocale;
    struct sockaddr_in adServeur;      /* Coordonnées du serveur (nom, port...) */

    adLocale.sin_family = AF_INET;          /* Type de la socket (TCP/IP) */
    adLocale.sin_port = htons(portLocal);   /* Affectation du port local */
    adLocale.sin_addr.s_addr = htonl(INADDR_ANY); /* Identificateur de l'hôte */

    skD = socket(AF_INET, SOCK_STREAM, 0);    /* Crée socket TCP (connectée) */

    tailleAd = sizeof(adLocale);

    retVal = bind(skD, (struct sockaddr*) &adLocale, tailleAd); /* Attache la socket */

    infosServeur = gethostbyname(nomServeur);           /* Récupère infos du serveur */
    adServeur.sin_family = infosServeur->h_addrtype;   /* Type de la socket du serveur */
    adServeur.sin_port = htons(portServeur);             /* Affectation du port destination*/
    memcpy(&adServeur.sin_addr, infosServeur->h_addr, infosServeur->h_length);

    printf("Adresse du serveur: %s \n", inet_ntoa(adServeur.sin_addr));

    tailleAd = sizeof(adServeur);

    retVal = connect(skD, &adServeur, tailleAd);

    return skD;
}
```

## Fonction d'ouverture de socket en attente de connexion (SERVEUR)

```
int ouvreSocket(int port)
{
    int skD;
    size_t tailleAd;
    struct sockaddr_in adLocale;

    adLocale.sin_family = AF_INET;           /* Type de la socket (TCP/IP) */
    adLocale.sin_port = htons (port);         /* Affectation du port local */
    adLocale.sin_addr.s_addr = htonl(INADDR_ANY); /* Identificateur de l'hote */

    skD = socket(AF_INET, SOCK_STREAM, 0);      /* Crée socket TCP (connectée) */

    tailleAd = sizeof(adLocale);

    retVal = bind(skD, (struct sockaddr*) &adLocale, tailleAd);

    if (listen(skD, 5) == -1)                  /* Ecoute sur la socket */

        return skD;
}
```

## Envoie d'un mot à un autre processus distant

```
tailleMot = strlen(argv[i]) + 1;      /* Taille du mot (+1 pour '\0') */

tailleMotRes = htonl(tailleMot);      /* Converti le type long en format réseau */

write(skDesc, &tailleMotRes, sizeof(long));    /* Envoie taille du mot */
write(skDesc, argv[i], tailleMot);        /* Envoie le mot lui-même */
```

## Réception d'un mot

```
skDesc = ouvreSocket(portLocal) ;
tailleAd = sizeof(adClient) ;

skServDesc = accept(skDesc, &adClient, &tailleAd); /* Bloquant */
/* Connexion établie, AdClient renseigné */

printf("\n*** INFORMATIONS CLIENTS ***\n");
printf("Adresse: %s \n", inet_ntoa(adClient.sin_addr)); /* Format u_long -> char* */
printf("Port: %d\n\n", ntohs(adClient.sin_port));
RetVal = read(skServDesc, &tailleMotRes, sizeof(long));

tailleMot = ntohs(tailleMotRes); /* Format réseau --> format machine */

mot = (char*) malloc(tailleMot * sizeof(char));

RetVal = read(skServDesc, mot, tailleMot);
```

## Serveur concurrent

```
skDesc = ouvreSocket(portLocal) ;
tailleAd = sizeof(adClient) ;
while (1)
{
    skServDesc = accept(skDesc, &adClient, &tailleAd) ;
    /* Connexion établie, AdClient renseigné */

    numPID = fork() ;
    if (numPID == 0)      /* Fils, s'occuper du client actuel */
    {
        close(skDesc) ;   /* Le fils n'a pas besoin de socket serveur */

        /* TRAITEMENT DU CLIENT */

        close (skServDesc) ;           /* Ferme socket de service */

        return 0;
    }
    else
    {
        close(skServDesc) ;   /* Le père n'a pas besoin de socket de service */
    }
}
```

## IV/ SOCKET DU DOMAINE UNIX

**Communication inter-processus qui s'exécutent sur la même machine. Alternative aux tubes, mémoire partagée etc...**

- + Aucun partage de données communes.
- + Avantage par rapport aux tubes, communication bi-directionnelles.
- Un échange d'information implique une double recopie des données à transmettre:
  - pour l'envoi, de l'espace d'adressage de l'émetteur vers le noyau
  - pour la réception, du noyau vers l'espace d'adressage du récepteur.
- Plus lent que les tubes

## IV/ SOCKET DU DOMAINE UNIX

Possibilité d'avoir le mode connecté ou déconnecté.

- Pratique pour le développement.
- Attention : comportement différent du domaine Internet : ici les 2 modes sont fiables car il n'y a pas de média physique, le noyau de l'OS est sûr, donc aucune perte.