

# Fundamentos de Software de Comunicaciones

---

## Tema 1 Introducción

# Características del software de comunicaciones

- Los sistemas de comunicaciones son totalmente dependientes del software
  - Centrales de conmutación
  - Nodos de conmutación en redes de datos (routers de Internet)
  - Estaciones base de redes móviles y equipos de certificación
  - Teléfonos móviles, puntos de acceso WIFI, etc.



# Características del software de comunicaciones

- El comportamiento de los protocolos siempre ha sido complejo y difícil de implementar para conseguir los requisitos deseados
- Algunas características que lo hacen complicado
  - Empotrado en hardware específico (complica el test)
  - Comportamiento reactivo (no termina)
  - Con información de estado (sesiones)
  - Sistema de tiempo real (cumplimiento de tiempos)
  - Sistema distribuido (múltiples nodos heterogéneos)

# Requisitos del software de comunicaciones

- 5 requisitos del software y/o metodología de desarrollo
  - Disponibilidad
  - Fiabilidad
  - Escalabilidad
  - Capacidad
  - Productividad

# Requisitos del software de comunicaciones (II)

## ■ Disponibilidad

- Medida del tiempo en servicio (-> tiempo no disponible)
  - 99% -> 87 horas y 36 minutos año
  - 99,9% -> 8 horas y 46 minutos año
  - 99,99% -> 52 minutos y 32 segundos año
  - 99,999% -> 5 minutos y 15 segundos año (6 seg. semana)
  - 99,9999% -> 32 segundos año ( < 1 segundo semana)
- Diferentes exigencias según el tipo de servicio
  - Servidores de información o reserva
  - Central telefónica
  - ADSL doméstico
  - Router de punto neutro

# Requisitos del software de comunicaciones (III)

- Fiabilidad
  - Funcionamiento correcto, libre de errores de ejecución
  - Su efecto depende del servicio
    - Fallo en una llamada telefónica
    - Error en transferencia de mensajes a nivel IP
    - Fallo en transacción financiera
  - Tema central de muchas herramientas y métodos de desarrollo

# Requisitos del software de comunicaciones (IV)

## ■ Escalabilidad

- Facilidad para incrementar el número de usuarios
- Ejemplo:
  - Gestionar un sistema de 100000 usuarios es menos costoso que gestionar 10 sistemas de 10000 usuarios
- Es necesario que se pueda incrementar el sistema cambiando configuraciones o incrementando el número de elementos

## ■ Capacidad

- El incremento en las prestaciones puede suponer nuevas necesidades de procesamiento u otros recursos
- Problemas cuando las actualizaciones requieren más recursos hardware

## ■ Productividad

- La metodología empleada debe permitir el trabajo de generación del software en un corto plazo, paralelizando su desarrollo
- Conceptos de marcos de trabajo, división en capas, componentes, etc.
- Definición clara de interfaces

# Estrategias para abordar la complejidad

---

- Organización en capas
- Normas de protocolos e interfaces
- Técnicas, herramientas y metodologías de desarrollo



# Organización en capas

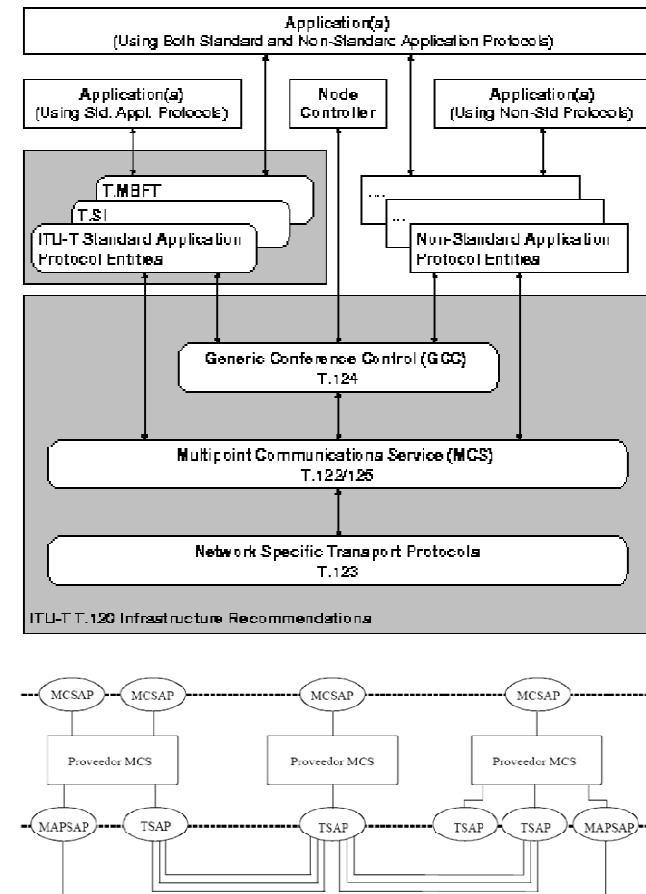
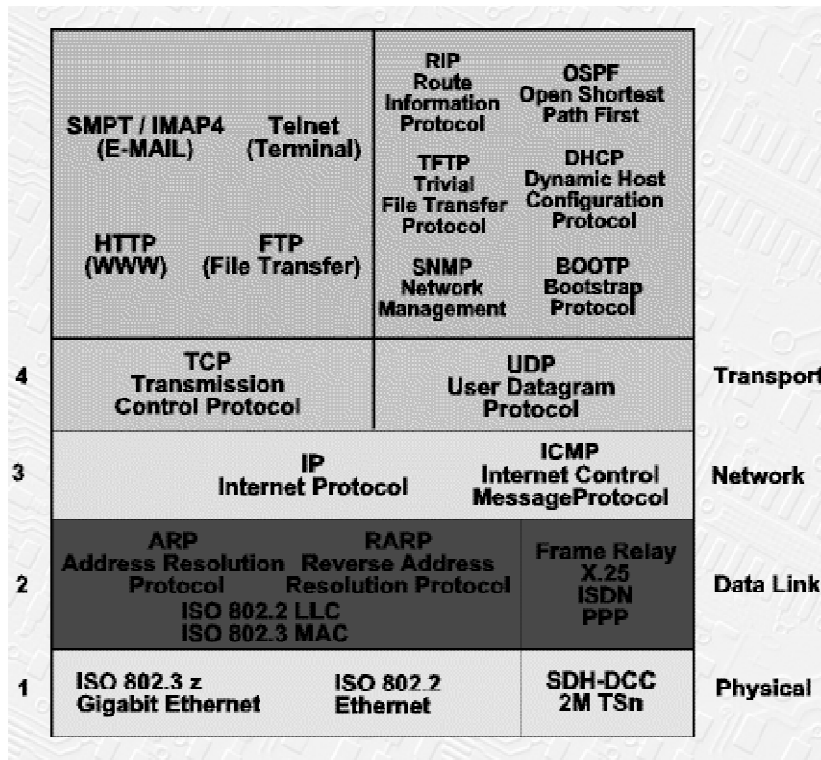
- Organización en capas:
  - La organización en capas permite separar funciones de manipulación de datos y de control

Layer	Data manipulation	Transfer control
Application	copying to appl. address space	
Presentation	encryption, formatting	
Session		
Transport	buffering for retransmission	congestion control, ACKs detection of transmission problems
Network		
Data Link	error detection and correction, buffering	flow control, framing, ACKs
Physical	network access	Multiplexing

[D. Clark, D. Tennenhouse: "Architectural considerations for a new generation of protocols", Computer Communication Review, 20(4), 1990]

# Organización en capas

Organización en capas (TCP/IP izqda., norma OSI drcha.)



# Normas de protocolos

- Normas de protocolos
  - Documentos en lenguaje natural, no formales
  - Diferentes organismos y tipo de normas
    - Internet – RFCs
    - Redes públicas fijas – CCITT, ITU-T
    - Rede móviles – 3GPP, ETSI
- Ejemplos de normas
  - RFC SMTP- Protocolo para correo electrónico
  - Protocolos de señalización en redes de telefonía móvil

# Estrategias para abordar la complejidad

- Técnicas y herramientas y de soporte para el desarrollo
  - Diseño del protocolo a partir de la definición de su objetivo o función
  - Descripción o modelado previo a la programación (informal o con lenguaje normalizado)
  - Programación o generación automática de código
  - Testing, depuración, validación
  - Análisis de prestaciones/rendimiento
- La selección de técnicas y herramientas da lugar a un enfoque de desarrollo diferenciado (metodología), que suele ajustarse a un dominio de aplicación concreto (servicios para la web, protocolos en el núcleo del S.O., ..)

# Técnicas y herramientas de desarrollo

- Diseño del protocolo a partir de la definición de su objetivo o función
  - La solución debe ser simple, evitando sobre especificación o falta de funcionalidad
  - Hay que asegurar que cumple su función correctamente (evitar bloqueos y otros errores)

The screenshot shows the SPIN CONTROL 3.2.3 software interface. The main window displays the SPIN DESIGN VERIFICATION window, which contains the simulation output. The output shows the preparation of a trail, a warning about a global variable, and the execution of two processes (proc 1 and proc 0) with their respective states and mutex values. A verification output dialog is also open, showing the results of the state space search, including the number of states, transitions, and memory usage.

```
SPIN CONTROL 3.2.3 - 20 July 1990 - File: pathfinder
```

```
File Edit Run Help
```

```
SPIN DESIGN VERIFICATION
```

```
Line#: 1 Find:
```

```
Simulation Output
```

```
preparing trail, please wait...done
spin: warning, "pan_in". global, "mtype l_state" variable is never used
1 proc 1 (low) line 40 "pan_in" (state 1) [l_state = waiting]
2 proc 1 (low) line 41 "pan_in" (state 2) [mutex==free]
<merge 1 new B>
3 proc 1 (low) line 41 "pan_in" (state 3) [mutex = busy]
4 proc 1 (low) line 42 "pan_in" (state 5) [l_state = running]
4 proc 0 (high) line 27 "par_in" (state 1) [h_state = waiting]
spin: trail ends after 4 steps
#processes: 2
4 proc 1 (low) line 46 "pan_in" (state 8)
4 proc 0 (high) line 28 "par_in" (state 4)
2 processes created
Exit-Status 0
```

```
Single Step Run Save in: sim.out Clear Cancel
```

```
Verification Output
```

```
pan: invalid endstate (cl Depth 4)
pan: wrote pan_in.trail
(Spin Version 3.0.10 -- 15 March 2003)
Warning: Search not completed

Full state space search for:
never-claim - (not selected)
assertion violations - (disabled by -a flag)
cycle checks - (disabled by -DCASTV)
invalid endstates +

State-vector 16 byte, depth reached 4, errors: 1
5 states, stored
1 states, matched
6 transitions (- stored+matched)
11 atomic steps
hash conflicts: 0 (resolved)
(max size 2019 states)

2.502 memory usage (Mbytes)

deal 0.0
user 0.0
sys 0.0
```

```
Save in: pathfinder.out Clear Close
```

```
Data Values
```

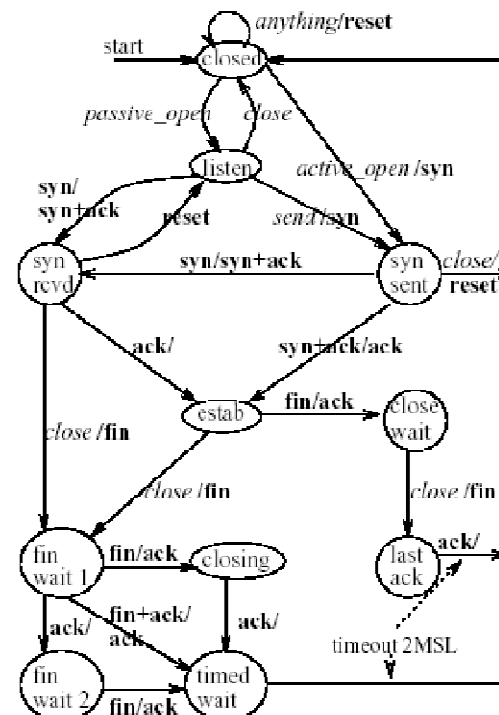
```
h_state = waiting
l_state = running
mutex = busy
```

```
starting simulation:
spin -X p v g o r t j0 pan_in
{at end of trail:
```

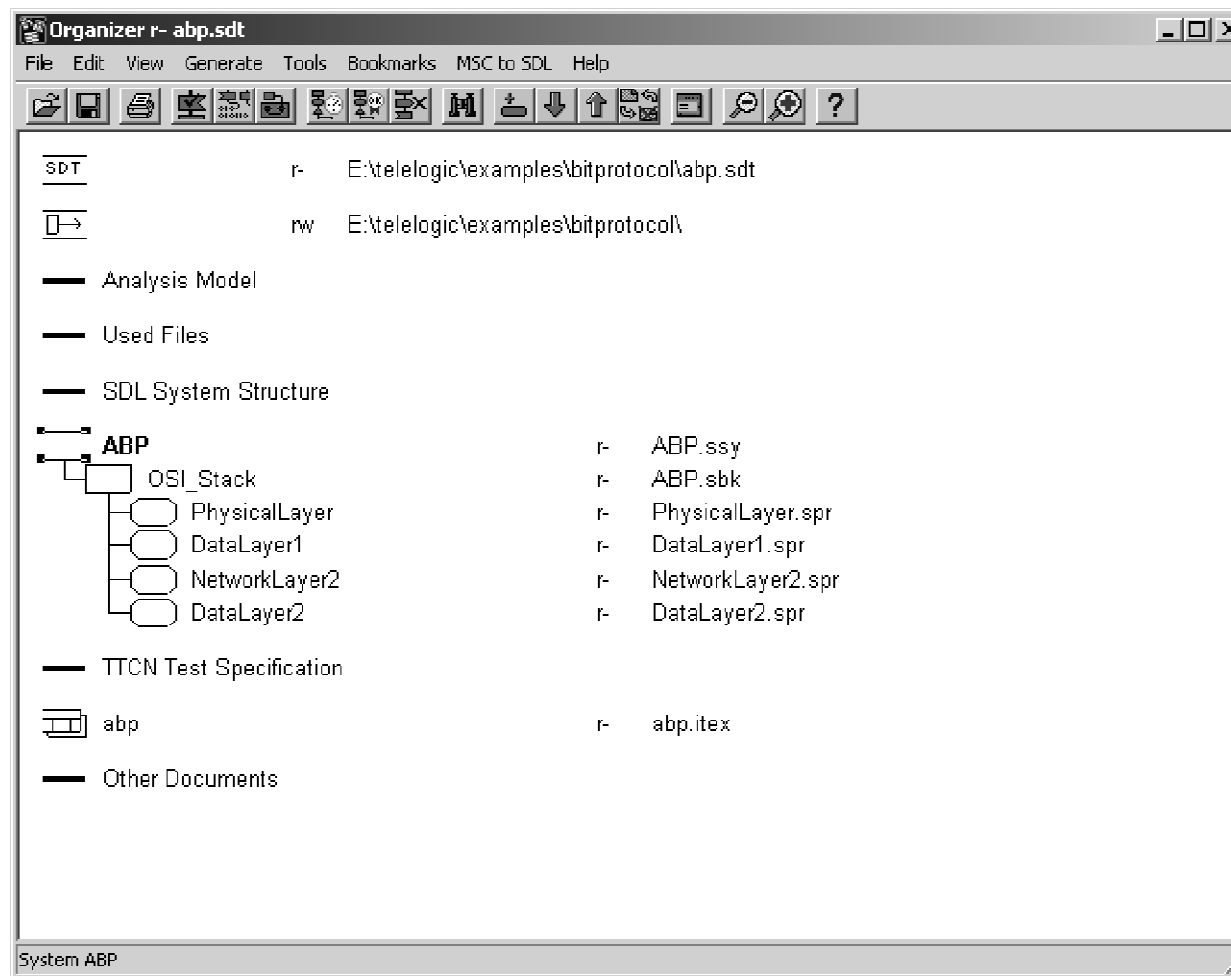
# Técnicas y herramientas de desarrollo

- Descripción o modelado previo a la programación (informal o con lenguaje normalizado)
  - Las normas de los organismos de estandarización pueden considerarse no formales
  - Los lenguajes normalizados cambian mucho respecto a su aceptación y herramientas que los soportan (máquinas de estado, UML, SDL, ..)

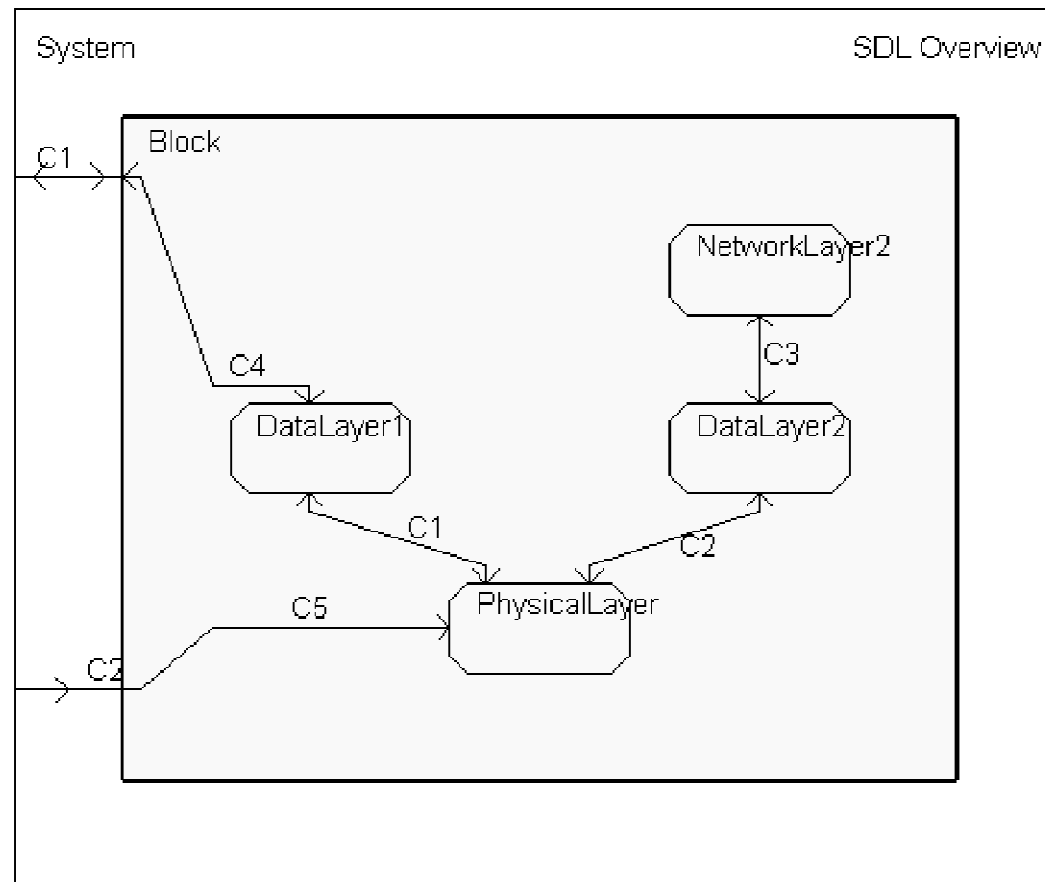
Esquema de TCP con máquina de estados



# Protocolo descrito en SDL

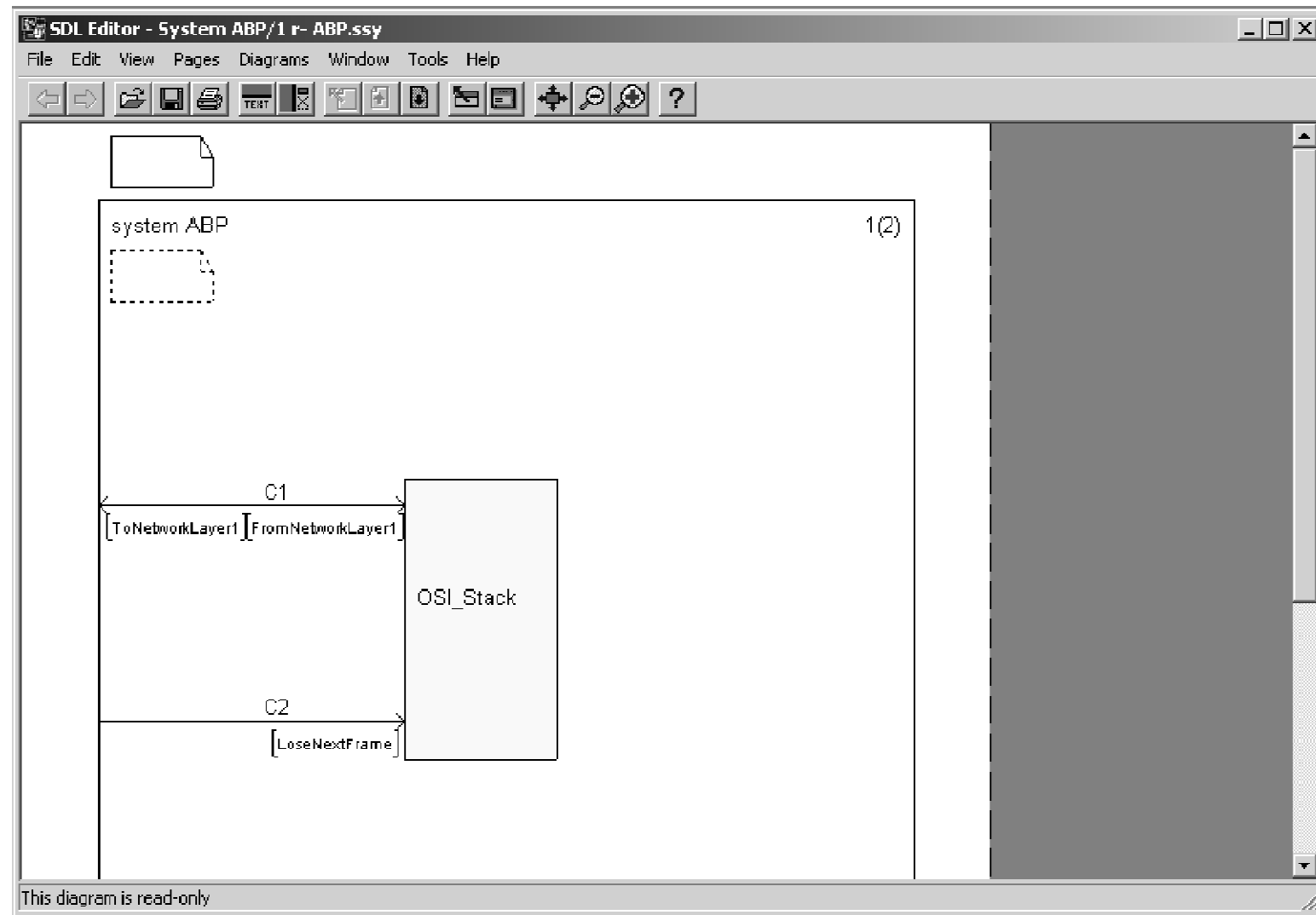


# Protocolo descrito en SDL

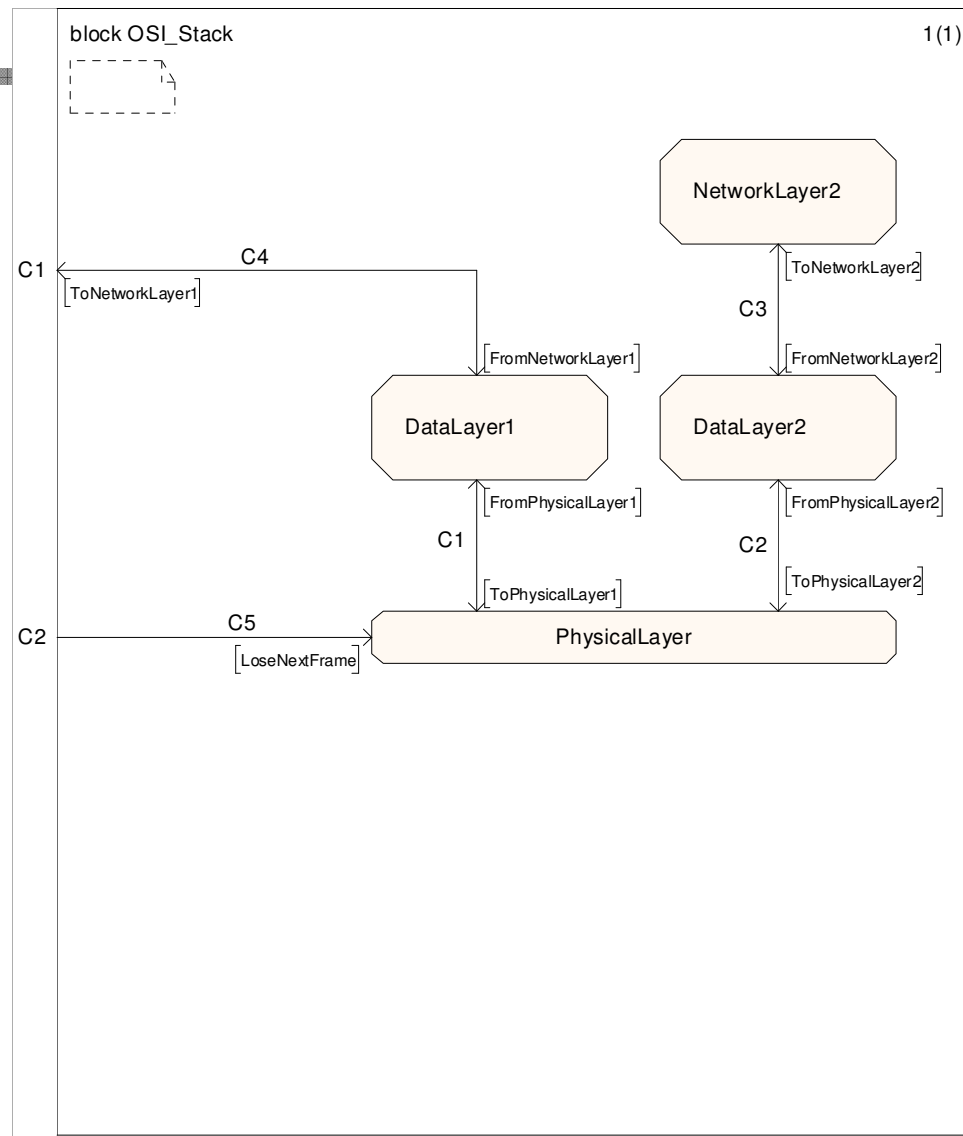




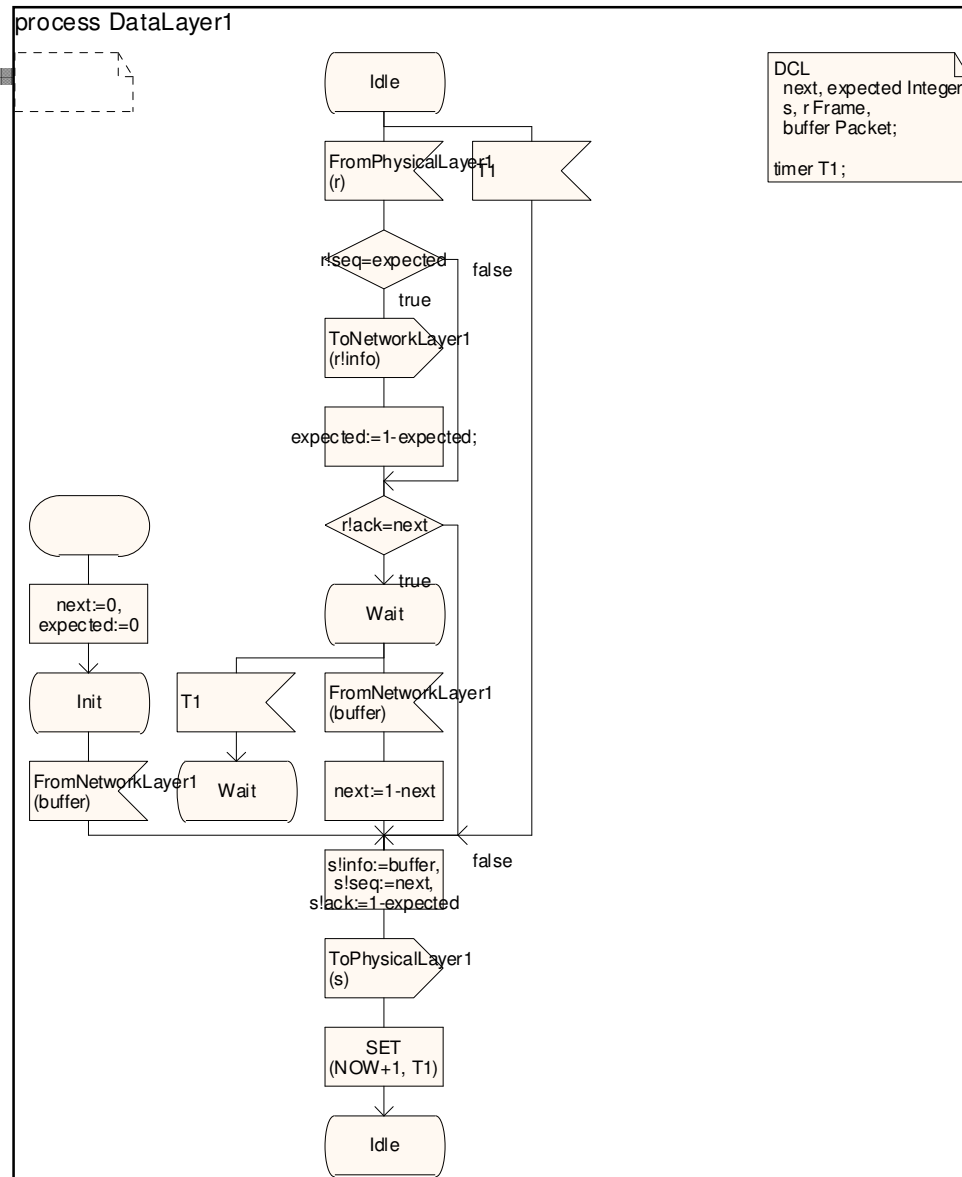
# Protocolo descrito en SDL



# Protocolo descrito en SDL



# Protocolo descrito en SDL



# Técnicas y herramientas de desarrollo

---

- Programación o generación automática de código
  - Elección de modelos de programación (lenguaje estructurado tipo C++ y APIs de sistema operativo, eventos, ..
  - Técnicas específicas para estructuración en capas, implementación de interfaces, manipulación
  - Marcos de trabajo específicos para programar protocolos sobre ellos
  - Herramientas de generación automática de código desde modelos

# Programas (aplicaciones de Internet)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "socks5.h"

main() {
    int          sockfd,n,result;
    struct sockaddr_in  serv_addr;
    char          buf[1024],serv_host_addr[30],username[255], passwd[255];

    printf("Direccion IP del servidor: ");
    gets(serv_host_addr);

    if ((sockfd=socket(PF_INET,SOCK_STREAM,0))<0) {
        perror("cliente no puede abrir stream socket\n");
        return -1;
    }
    /*inicializacion de la variable serv_addr*/
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr(serv_host_addr);
    serv_addr.sin_port=htons(SERV_PROX_PORT);
    bzero(buf,sizeof(buf));

    /*conexion activa desde el cliente */
    if (connect(sockfd,(struct sockaddr*) &serv_addr,sizeof(serv_addr))!=0) {
        perror("cliente no puede conectar con servidor\n");
        return -1;
    }
}
```

# Técnicas y herramientas de desarrollo

## ■ Testing, depuración, validación

- Hay que evitar problemas de uso de memoria, bloqueos, ciclos y otros errores,
- Para lenguajes de programación
  - Básico: depuradores
    - tipo gdb,
    - herramientas tipo netstat, ping ,
    - analizadores de protocolos tipo wireshark
  - Avanzado: herramientas automáticas de chequeo del código como Valgrind,
- Para lenguajes de modelado tipo SDL o UML
  - Básico: simulación visual
  - Avanzado: validación automática (exhaustiva), testing

# Captura con Wireshark

5522	43.582191	192.168.198.3	192.168.1.118	FTP-DATA	FTP Data: 632 bytes
5523	43.582214	192.168.1.118	192.168.198.3	TCP	49829 > ftp-data [ACK] Seq=1 Ack=2113537 Win=65520 Len=0
5524	43.582422	192.168.198.3	192.168.1.118	FTP-DATA	FTP Data: 1260 bytes
5525	43.583138	192.168.198.3	192.168.1.118	FTP-DATA	FTP Data: 1260 bytes
5526	43.583162	192.168.1.118	192.168.198.3	TCP	49829 > ftp-data [ACK] Seq=1 Ack=2116057 Win=65520 Len=0
5527	43.583230	192.168.198.3	192.168.1.118	FTP-DATA	FTP Data: 1260 bytes
5528	43.583359	192.168.198.3	192.168.1.118	FTP-DATA	FTP Data: 1260 bytes
5529	43.583390	192.168.1.118	192.168.198.3	TCP	49829 > ftp-data [ACK] Seq=1 Ack=2118577 Win=65520 Len=0
5530	43.583415	192.168.198.3	192.168.1.118	FTP-DATA	FTP Data: 296 bytes
5531	43.583475	192.168.1.118	192.168.198.3	TCP	49829 > ftp-data [ACK] Seq=1 Ack=2118874 Win=65224 Len=0
5532	43.585886	192.168.1.118	192.168.198.3	TCP	49829 > ftp-data [FIN, ACK] Seq=1 Ack=2118874 Win=65224 Len=0
5533	43.586546	192.168.198.3	192.168.1.118	TCP	ftp-data > 49829 [ACK] Seq=2118874 Ack=2 Win=26460 Len=0
5535	43.782025	192.168.1.118	192.168.198.3	TCP	49821 > ftp [ACK] Seq=280 Ack=888 Win=7304 Len=0
5545	46.473148	192.168.1.118	192.168.198.3	FTP	Request: QUIT
5546	46.474149	192.168.198.3	192.168.1.118	FTP	Response: 221 Goodbye.
5547	46.476305	192.168.198.3	192.168.1.118	TCP	ftp > 49821 [FIN, ACK] Seq=902 Ack=286 Win=25200 Len=0
5548	46.476420	192.168.1.118	192.168.198.3	TCP	49821 > ftp [ACK] Seq=286 Ack=903 Win=7288 Len=0
5549	46.478080	192.168.1.118	192.168.198.3	TCP	49821 > ftp [FIN, ACK] Seq=286 Ack=903 Win=7288 Len=0
5550	46.478869	192.168.198.3	192.168.1.118	TCP	ftp > 49821 [ACK] Seq=902 Ack=287 Win=25200 Len=0

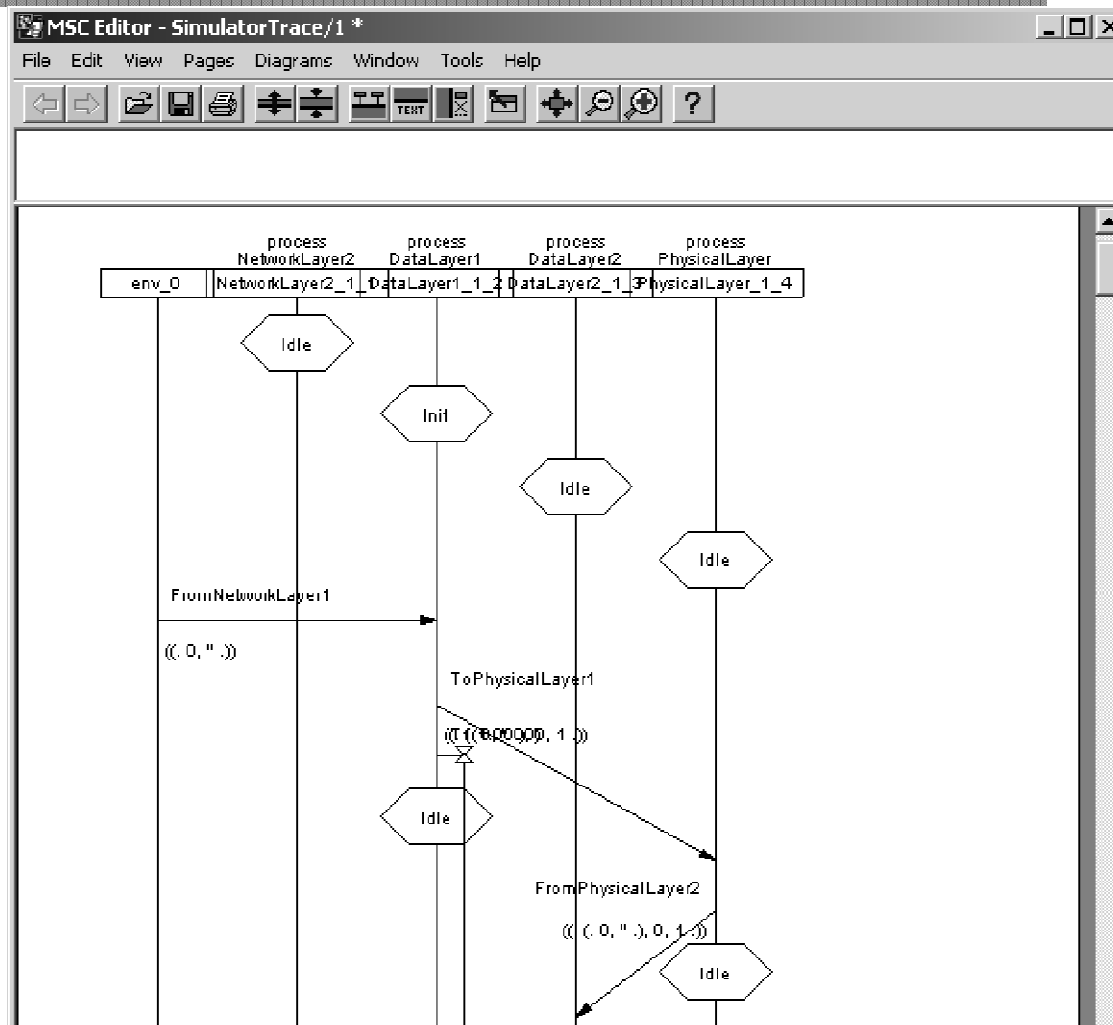
  

Sequence number: 280	(relative sequence number)
[Next sequence number: 286	(relative sequence number)]
Acknowledgement number: 888	(relative ack number)
Header length: 20 bytes	
▼ Flags: 0x18 (PSH, ACK)	
0... .. = Congestion Window Reduced (CWR): Not set	
.0... .. = ECN-Echo: Not set	
..0. .... = Urgent: Not set	
...1 .... = Acknowledgement: Set	
.... 1... = Push: Set	
.... .0.. = Reset: Not set	
.... .0. = Syn: Not set	
.... ...0 = Fin: Not set	
Window size: 7304 (scaled)	
▶ Checksum: 0x5ac8 [validation disabled]	
▶ [SEQ/ACK analysis]	
▼ File Transfer Protocol (FTP)	
▼ QUIT\r\n	
Request command: QUIT	

Respuesta SYN-ACK  
por parte del  
extremo llamado

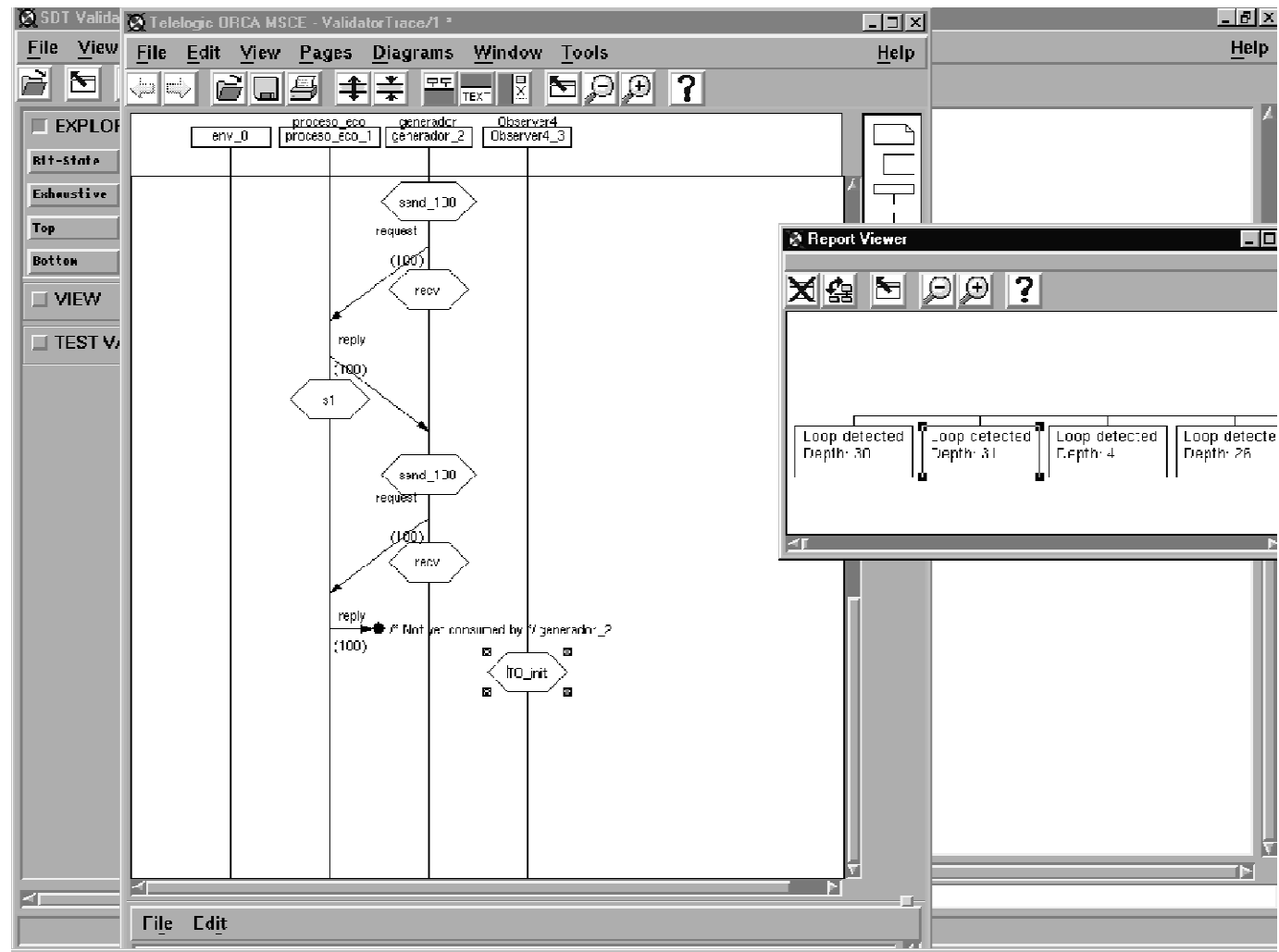
Finalización de  
conexiones  
mediante  
activación de  
flags FIN y ACK.

# Simulación (ejemplo en SDL)





# Validación (ejemplo en SDL)



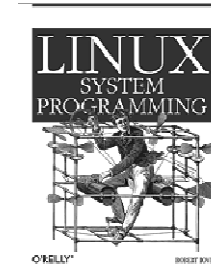
# Técnicas y herramientas de desarrollo

## ■ Análisis de prestaciones/rendimiento

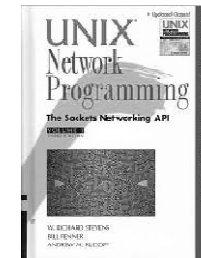
- Hay que asegurar
  - Que se cumplen los tiempos de respuesta
  - Que se puede atender el número de peticiones o de tráfico
  - Que se mantiene el límite de uso de recursos (memoria, CPU, energía,..)
- Algunas posibilidades
  - Simuladores de red tipo ns-2, ns-3, OPNET (en diseño)
  - Pruebas de carga (varios clientes contra un servidor)
  - Herramientas de “profiling” sobre el código
  - Monitorización de tráfico y post-procesado (Wireshark)

# Bibliografía

- Robert Love. "Linux System Programming", O'Reilly, 2007



- Richard Stevens et al. "Unix Network Programming Volume 1: The Sockets Networking API (3rd Edition)", Addison Wesley, 2003



- BEHROUZ A. FOROUZAN , TRANSMISION DE DATOS Y REDES DE COMUNICACIONES (4ª ED.), McGraw-Hill, 2007

