

MEGA HIPER ARCHI RECONTRA

RESUMEN DE

TÉCNICAS [ANALES] **DIGITALES III**

POR:

MP

FA

MC [este no tanto]

Hecho durante el 2020

[cualquier queja diríjase al decano, para que no haga nada, alguna vez hace algo ese?, como nosotros que ya sacamos este materón jaja pichón, te falta calle]

[es probable que encontrés errores de ortografía y de formateo, algunos son culpa nuestra y otra culpa de pasar del Google docs a Word, no sé por qué tiene la manía de dejar espacios en blanco que no puedo sacar, y algunas imágenes corridas de lugar, asi que lee con atención.]

[los diagramas los hicimos con diagrams.net]

U1: Arquitectura de CPU	14
Pasos del Pipeline de IA32	14
Elementos del Pipeline de IA32	14
Bus Unit:	14
Code Cache:	14
Prefetcher:	14
Prefetch Buffers:	14
Instruction Decode Unit:	14
Control Unit:	15
ALUs:	15
Address generators:	15
Data Cache:	15
Paging Unit:	15
Floating-Ponit Unit:	15
Modo Real	15
Acceso a Memoria	15
Interrupciones y Excepciones	15
Características RISC de Pentium	16
Modo Protegido	16
Traducción de Dirección Lógica a Física en Modo Protegido	16
Selectores	17
Descriptores	17
Tipos de descriptores de segmentos	18
Descriptores de segmentos de NO-SISTEMA (Código y Datos) S=1	18
Descriptores de Segmentos de Sistema	18
a) Descriptor de LDT S = 0 Type = 2	18
b) Descriptor de TSS S = 0 Type = 1, 3, 9, B	18
c) Descriptor de GATE S = 0 TYPE = 4 a 7, C, F	18
Tablas de Descriptores	19
Tabla Global de Descriptores (GDT)	19
Tabla local de Descriptores (LDT)	19
Tabla de Descriptores de Interrupciones(IDT)	20
Segmentación	20
Paginación BIT PG = 1 EN CR0(CONTROL REGISTER 0)	21
Funcionamiento de la paginación	21
PDE y PTE DESCRIPTORS	22
PASO A PASO DE LA TRADUCCIÓN LÓGICA A FÍSICA EN MODO PROTEGIDO SIN TLB	23
TRANSLATION LOOKASIDE BUFFER (TLB)	24

OPERACIÓN DE PAGINACIÓN	24
Protección	27
Protección por segmentación	27
Protección por Niveles de Privilegios	27
Acceso a Segmento de Datos Restringido por Privilegios	27
Acceso a Datos en Segmento de Código Restringido por Privilegios	28
CallGates	28
Descripción funcional del CallGate	30
Alternativa al CallGate	30
Interrupciones y Excepciones	30
Multitareas	31
Compartición de Tiempo	31
Multiusuario vs Multitarea	31
Métodos de Planificación para SO multiusuario	31
Planificación por Porción de Tiempo	31
Planificación por Prioridad	31
Cambio de Contexto	31
Registros y Descriptores para Multitarea en Modo Protegido	31
Task State Segment	31
Set Dinámico	31
Set Estático	32
TSS Descriptor	32
Task Register (TR)	32
LTR: Load Task Register	32
STR: Store Task Register	32
Task Gates y Task Gate Descriptor	33
Cambio de Tareas (Task Switching)	33
Cambio SIN Task Gate (método directo)	33
Cambio CON Task Gate (método indirecto)	34
Tareas Anidadas	34
U2: Sistemas Operativos	35
¿Qué es un SO?	35
Modos de Operación del Software	35
Revisión del Hardware	35
Procesadores	35
Memoria	35
Dispositivos de E/S	36
Conceptos Generales	36

Procesos	36
Espacio de Direcciones	36
Archivos	36
Protección	36
El Shell	36
Llamadas a Sistema	36
Llamadas para Administración de Procesos FORK	37
Llamadas para Administración de Archivos	37
Estructuras de un SO	37
Procesos	38
Modelo	38
Creación	38
Terminación	38
Jerarquías	38
Estados	38
Transiciones entre estados	38
Implementación	39
Hilos	39
Ventajas	39
Especificaciones varias	40
Hilos en espacio de Usuario	40
Hilos en espacio de Kernel	40
Relación entre las implementaciones	40
Planificador [se viene el acv]	40
Cuándo Planificar Procesos?	41
Tipos de Planificación	41
Categorías de los Algoritmos de Planificación	41
Metas de los Algoritmos (dependen del entorno)	41
Algoritmos de planificación por Lotes	41
Algoritmos de Planificación Interactiva	42
Planificación de Hilos	42
Hilos Nivel Usuario	42
Hilos nivel Kernel	42
Planificación en Sistemas de Tiempo Real	43
IPC: Comunicación entre procesos	43
Clasificación	43
Paso de mensajes:	43
Memoria Compartida:	43

Caminos que se pueden tomar para compartir información	44
Persistencia de los mensajes en IPC	44
Persistencia de Proceso	44
Persistencia de Kernel	44
Persistencia de Sistema de Archivos	44
¿Cuándo uso cada una de estas vergas?	44
Pljas, digo... PIPES	45
FIFO (pijas con nombre)	45
Problemas Stream de Datos	45
Cola de Mensajes [no es la cola que estás pensando]	46
Resumen de Códigos en C	46
Sincronización	49
Modelos de Ejecución	49
Definiciones	49
Concurrencia	49
Determinístico	49
Estrategias de Sincronización	49
Serialización	49
Punto de encuentro:	49
Exclusión mutua:	49
Barrera	49
Condición de Competencia o Carrera:	49
Herramientas de Sincronización	50
Herramientas pedorras:	50
Semáforos	50
Operaciones	50
Mutex	50
Inicialización	50
Mutex dinámico	50
Mutex recursivo	50
Interbloqueo:	50
Gestión de Memoria	51
Sin Abstracción	51
Ejecución de múltiples programas sin abstracción	51
ABSTRACCIÓN DE MEMORIA: ESPACIO DE DIRECCIONES [BONJOURRRR MANGA DE SOQUETES]	51
Espacio de Direcciones	51
Registros Base y Límite:	51

Intercambio	52
Fragmentación Externa e interna:	52
Compactación de Memoria:	52
¿Cuánta memoria se asigna a proceso?	52
Administración de Memoria Libre	52
Mediante Mapa de Bits	52
Mediante Listas Enlazadas	53
Algoritmos de Asignación de Memoria por Lista Enlazada	53
Conclusiones	53
Memoria Virtual	53
Paginación	53
Proceso de búsqueda de página	54
Estructura de una entrada en la tabla de páginas	54
Algoritmos de Reemplazo de Páginas	55
Conclusiones de los Algoritmos	55
Segmentación	56
Segmentación y Paginación	56
U2.1: Sistemas Operativos de Tiempo Real	57
Tareas	57
Estados de las Tareas	57
Planificador	58
Tareas y Datos	58
Funciones Reentrantes	58
Semáforos	58
Múltiples Semáforos	59
Semáforos para sincronizar tareas	59
Ventajas	59
Problemas con los semáforos [la mayoría son por boludo]	59
Inversión de Prioridad	59
Abrazo mortal	60
Resumen de métodos para proteger recursos compartidos	60
Colas para comunicar tareas	60
Características de las Colas (¡~(=):	60
Creación de colas en FreeRTOS (sorry es que me la deja picando el título)	60
Envío de datos a la cola [(~) se lo que estás pensando y no, no es eso]	60
Recepción de datos	61
Rutinas de atención a interrupciones	61
Rutinas de atención a interrupciones en FreeRTOS	61

Funciones	61
Gestión de tiempo	62
Funciones	62
Nota sobre EDU-CIAA	62
U4: Numerología [we] [dieron la U4 primero que la 3, cualquier queja diríjase al pela]	65
Representación de enteros	65
Representación en PUNTO FIJO	65
Conversión de Real a Punto Fijo	66
Punto Flotante a Punto Fijo	66
Punto Fijo a Punto Flotante	66
Factor de Escala	66
Rango Dinámico	66
Overflow	67
Saturación	67
Multiplicación en complemento a 2	67
Underflow	67
Esquemas de Redondeo	68
Redondeo hacia -infinito	68
Redondeo hacia el más cercano	68
Errores en redondeo	68
Operación MAC [Multiplicar y Acumular]	68
Desplazamientos Aritméticos	69
Representación de Punto Flotante	69
Precisión Simple 32 bits	69
Valores especiales de Exponente:	70
Métodos de Redondeo	70
Rango Dinámico	70
Precisión	71
Suma de Flotantes	71
Conclusiones	71
Etapas de Procesamiento de Señales	71
Muestreo Periódico	71
Proceso de Muestreo	72
Representación en el Dominio de la Frec	72
Pre Filtrado	72
Oversampling	73
Conversor ADC	73
Cuantizador	73

Errores de Cuantización	73
Relación SNR en ADC	74
Consideraciones	74
SNR para Senoidal Pura	75
SNR para señal genérica	75
Conversor DAC	76
Conversor IDEAL	76
Respuesta en Frecuencia del ZOH	76
Filtros	77
FIR: Ventana Móvil	77
Rta al Escalón	77
Definición	77
Respuesta en Frecuencia	78
Influencia del Orden del Filtro en la RTA en FREC.	78
Cálculo del Orden del Filtro	78
FIR: Ventaneo	79
Estrategia de Filtrado por Seno Cardinal Ventaneando	79
Diferencias entre las ventanas de HAMMING y BLACKMAN [blackman la tiene más grande.. por nekro]	80
Ventana de Kaiser	80
Comparación	81
Estructuras de Implementación de FIR	81
IIR Leaky Integrator	82
Respuesta en Frecuencia	82
Comparación con el FIR	83
IIR: Transformada Bilineal (Método de Tustin) [un mostro el tustin este]	83
Relación entre las frecuencias analógicas y digitales	83
Precombado	84
Pasos de Diseño	84
Formas de implementar [dudo que pregunte esto, es una poronga]	84
Forma directa 1	84
Forma directa 1 (implementación invertida)	85
Forma directa 2	85
Implementación en Cascada	85
U3: Introducción a los sistemas de Comunicaciones de Datos [prepare el ocote]	86
Introducción, Modelos de Referencia de Redes, Capa Física	86
Hardware de Red	86
Tecnología de transmisión:	86

Escala:	86
Redes de área personal	87
Área Personal:	87
Área Local:	87
Redes de área metropolitana	87
Red de Área Metropolitana:	87
Redes de área amplia	87
Red de Área Amplia:	87
Interredes	88
SOFTWARE DE RED	88
Jerarquías de protocolos	88
Aspectos de diseño para las capas	89
Confiabilidad:	89
Evolución de la red:	89
Asignación de recursos:	89
Asegurar la red y defenderla contra amenazas:	89
Comparación entre servicio orientado a conexión y servicio sin conexión.	89
Primitivas de servicios	90
La relación entre servicios y protocolos	91
MODELOS DE REFERENCIA	92
El modelo de referencia OSI	92
El modelo de referencia TCP/IP	93
El modelo utilizado en este libro	94
Comparación de los modelos de referencia OSI y TCP/IP	94
Una crítica al modelo y los protocolos OS	95
Una crítica al modelo de referencia TCP/IP	95
CAPA FÍSICA	95
RESUMEN	95
Capa de Enlace, PPP	96
Cuestiones de Diseño de la Capa de Enlace de Datos	96
Servicios proporcionados a la Capa de Red	96
Entramado	97
Conteo de Bytes:	98
Bytes bandera o Relleno de Bytes	98
Bits bandera con relleno de bits	99
Violaciones de Codificación de la capa física	100
Control de Errores	100
Control de Flujo	100

Detección y Corrección de Errores	100
Código Hamming	101
Distancia de Hamming:	101
Funcionamiento	101
Código Convolucional	102
Código Reed-Solomon	102
LDPC (verificación de paridad de baja densidad)	103
Códigos Detectores de Errores	103
Paridad	103
Suma de Verificación	104
CRC (Comprobación de Redundancia Cíclica)	104
Protocolos de Enlace de Datos	105
Paquetes sobre SONET	105
Diferencias con HDLC	105
Formato de Trama PPP	106
Subcapa de Acceso al Medio, Ethernet	107
El Problema de Asignación del Canal	107
Asignación estática de canal	107
Supuestos para la asignación dinámica de canales	107
Protocolos de Acceso Múltiple	108
ALOHA [los inventaron en Hawái.. muy ocurrentes con los nombres]	108
ALOHA puro	108
ALOHA ranurado	109
Protocolos de acceso múltiple con detección de portadora	110
CSMA persistente y no persistente	110
CSMA con detección de colisiones	110
Protocolos libres de colisiones	111
Un protocolo de mapa de bits	111
Paso de token	112
Conteo descendente binario	112
Protocolos de contención limitada	112
El protocolo de recorrido de árbol adaptable	113
Protocolos de LAN inalámbrica	113
ETHERNET	114
Capa física de Ethernet clásica	114
El protocolo de subcapa MAC de la Ethernet clásica	115
CSMA/CD con retroceso exponencial binario	116
Desempeño de Ethernet	116

Ethernet commutada	117
Fast Ethernet	117
Gigabit Ethernet	118
10 Gigabit Ethernet	119
Retrospectiva de Ethernet	119
Capa de Red	121
ASPECTOS DE DISEÑO DE LA CAPA DE RED	
Conmutación de paquetes de almacenamiento y reenvío	121
Servicios proporcionados a la capa de transporte	121
Implementación del servicio sin conexión	121
Implementación del servicio orientado a conexión	122
Comparación entre las redes de circuitos virtuales	123
INTERCONEXIÓN DE REDES	
Cómo difieren las redes	124
Cómo se pueden conectar las redes	124
Tunelización	125
Enrutamiento entre redes	126
Fragmentación de paquetes	126
LA CAPA DE RED DE INTERNET	
El protocolo IP versión 4	128
Direcciones IP	130
Prefijos	130
Subredes	131
CIDR: Enrutamiento Interdominio sin Clases	132
Direccionamiento con clases y especial	133
IP versión 6	133
El encabezado principal de IPv6	134
Encabezados de extensión	135
Protocolos de control en Internet	136
DHCP: el Protocolo de Configuración Dinámica de Host	138
Capa de Transporte [esto si lo retoco porque está DEMASIADO TEXXXXTO]	140
EL SERVICIO DE TRANSPORTE	
Servicios que se proporcionan a las capas superiores	140
Primitivas del servicio de transporte	140
Sockets de Berkeley	142
Un ejemplo de programación de sockets: un servidor de archivos de Internet [meh, no lo borro por las dudas]	143
ELEMENTOS DE LOS PROTOCOLOS DE TRANSPORTE	
	143

Direccionamiento	144
Establecimiento de una conexión	145
Acuerdo de 3 vías	146
Liberación de una conexión	147
Control de errores y almacenamiento en búfer	148
Multiplexión	150
Recuperación de fallas	151
LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: UDP	152
Introducción a UDP	152
Protocolo de transporte en Tiempo Real [supuestamente no lo toma, pero ya estamos grandes para creernos esas]	153
LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: TCP [hipermegateraMuchoTexto]	154
Introducción a TCP	154
El modelo del servicio TCP	154
El protocolo TCP	155
El encabezado del segmento TCP	155
Establecimiento de una conexión TCP	157
Liberación de una conexión TCP	157
Modelado de administración de conexiones TCP	157
Ventana deslizante de TCP [el tafe suele dar hasta acá, el resto lo anotó el manija el fede porque le pegaban de chiquito]	158
Administración de temporizadores de TCP	159
Control de congestión en TCP	160
El futuro de TCP	163
Capa de Aplicación: DNS, HTTP	165
DNS: Sistema de Nombres de Dominio	165
Espacios de Nombres	165
Registros de Recusos	165
Servicios de Nombres	165
WORLD WIDE WEB	166
Arquitectura	166
Cliente	166
Servidor	167
Cookies	167
Páginas estáticas	167
HTML	167
Páginas Dinámicas y Apps Web	168
Generación del lado del Servidor	168
Generación del lado del Cliente	168

HTTP	168
Conexiones	168
Métodos	168
Encabezados de Mensajes	168
Caché	169
Teoría de Tps de Ana de Redes	170
Redes TP1: Introducción	170
Switch (Comutador)	170
Direcciones IP (modelo IP V4)	171
¿Qué funcionalidad cumple el broadcast?	171
Redes Tp2: Capa de Enlace	173
Redes Tp3: Capa de enlace Ethernet	175
Topologías	175
Codificación: usa Manchester y Manchester Diferencial	175
Redes Tp4: Capa de Red - Ruteo Estático	177
Los servicios que proporciona a la capa de transporte deben cumplir:	177
Direcciones IP	177
Puerta de enlace predeterminada o Gateway	177
Ruteo	177
Ruteo estático:	177
Redes Tp5: Capa de Red y Transporte - NAT y DHCP	178
DHCP (Dinamic Host Configuration Protocol)	181
Redes Tp6: Capa de Transporte - UDP y TCP	183
UDP	186
TCP	186
FTP: Protocolo de Transferencia de Archivos	187

U1: Arquitectura de CPU

Pasos del Pipeline de IA32

1. Prefetch: instrucciones son captadas del caché o memoria y colocadas en el Prefetch buffer.
2. Instruction Decode: el decoder verifica si es posible ejecutar varias instrucciones a la vez (hilos) o en una sola pipeline.
3. Adress Generate: calcula las direcciones de los operandos y datos en memoria.
4. Execute: lee los operandos de memoria y ejecuta operaciones en las ALU's. Verifica predicciones de ramificación de instrucciones.
5. WriteBack: escribe los resultados de la última instrucción y verifica la producción de saltos condicionales.

Ambas pipelines tiene que llegar al punto 5 a la vez, se esperan mutuamente-> Se pueden atrasar mutuamente.

Elementos del Pipeline de IA32

Bus Unit:

Se encarga de comunicar el cpu con el resto del sistema y memoria. 32Bits de dirección y 64 de datos. Controla el bus principal del sistema y puede pedir el uso del bus a otras Unidades de Bus. Controla tambien el funcionamiento del caché L2 externo y el L1 interno.

Code Cache:

Es un caché de instrucciones de 8KB que provee acceso rápido a las instrucciones más usadas, las tiene copiadas. Alimenta las dos pipelines. Puede ser leído por ambas pipes a la vez y por el prefetcher.

Prefetcher:

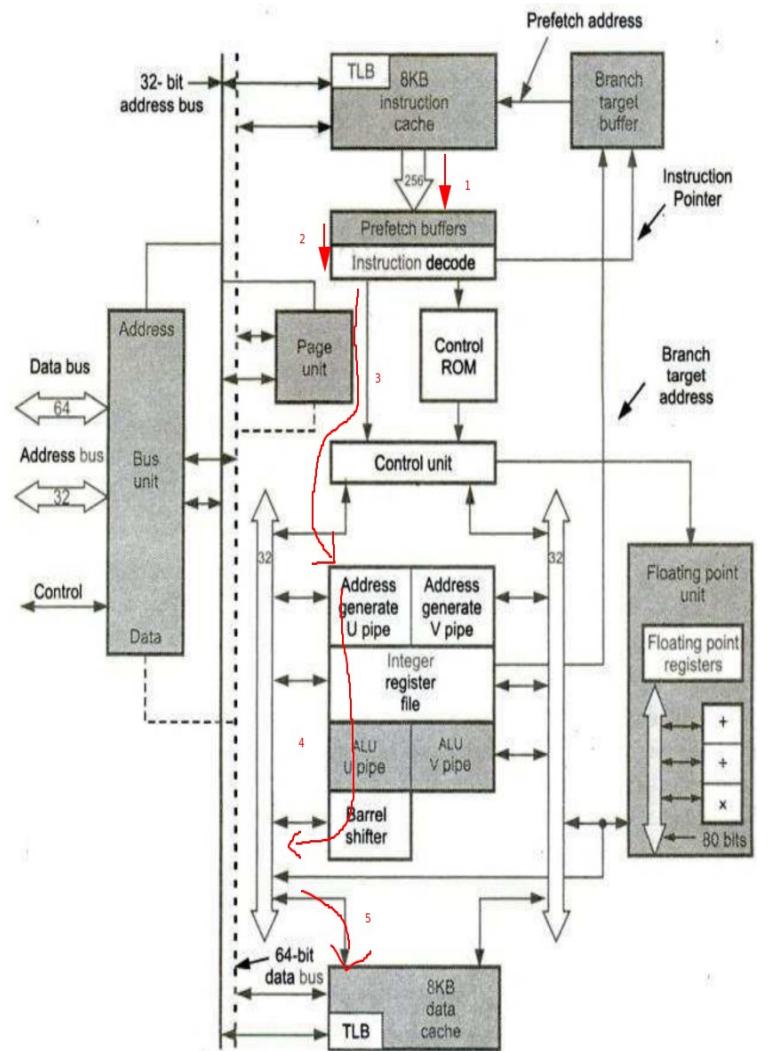
Pide instrucciones al Caché de código. Si no están las busca en memoria para llenar el caché.

Prefetch Buffers:

Contiene 4, trabajan en pares de dos. Cuando se predicen instrucciones del caché se colocan en estos buffers, dejando dos libres. Cuando se predice una bifurcación en el BTB(Branch Target Buffer) las instrucciones de la ramificación predecidas se colocan en los dos espacios que estaban libres.

Instruction Decode Unit:

Decodifica las instrucciones en dos pasos. D1(decode 1): decodifica el OPCODE en ambas pipes y determina si se puede hacer Hilo. Si se puede, las instrucciones se envían en simultaneo a D2(decode 2): calcula las direcciones de memoria de los operandos que le pasó la D1.



Control Unit:

También llamada unidad de Microcódigo. Maneja excepciones, breakpoints e interrupciones. También controla las pipelines de enteros y punto flotante.

ALUs:

Unidades Lógicas Aritméticas.

Address generators:

Generan las direcciones **LINEALES** de memoria especificadas en las instrucciones enviadas a cada pipe.

Data Cache:

Tiene una copia de la información más usada. Es de 8KB.

Paging Unit:

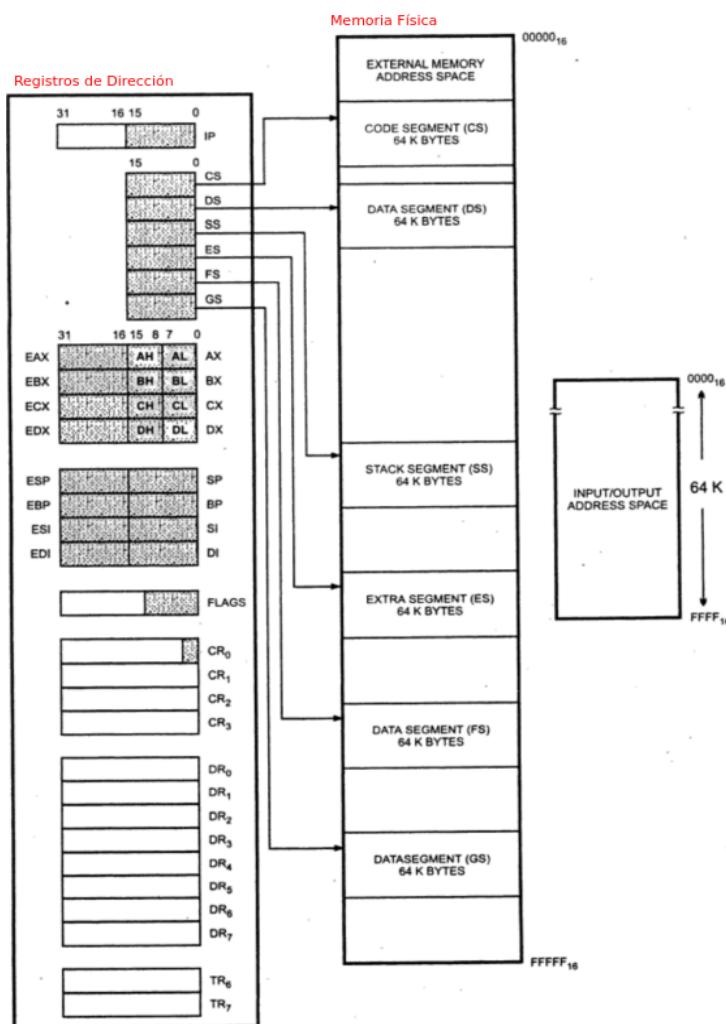
Traduce las direcciones lineales generadas por el Address Generator en direcciones **FÍSICAS**.

Floating-Ponit Unit:

Opera con números en punto Flotante.

Modo Real

Es un modo de compatibilidad con 8086. Trabaja en modo de 16bits, por eso tambien se limita el tamaño de la memoria direccionable.

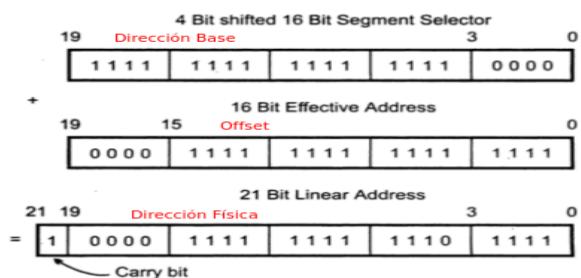


La memoria se limita a 1Mbyte. Bus de direcciones de 20 bits. No toda la memoria se puede acceder a la vez. Se partitiona en bloques de 64Kb. Cada segmento es una unidad direccionable independientemente.

Los registros de segmento contienen la dirección de inicio de cada bloque. En modo real, como la unidad de pag está apagada, las direcciones lineales son iguales a las físicas.

Acceso a Memoria

La dirección física se obtiene de desplazar el selector de segmento 4 bits a la izq. haciéndose ahora de 20bits y se suma la dirección lógica. Si se produce un acarreo este se almacena. Si la dirección obtenida se pasa del máximo se produce una excepción. Todos los registros de segmentos son accesibles al programador.



Interrupciones y Excepciones

Utiliza la IDT (Interrupt Descriptor Table), contiene punteros que apuntan a los inicios de las ISR. Cada puntero es de 4 bytes. 2 de dirección base y 2 de offset. La IDT es reubicable con el registro IDTR y variable en tamaño. IDTR contiene base y limit.

Características RISC de Pentium

- Acceso reducido a memoria: mediante cachés SRAM dentro del CPU. Disminuye acceso a memoria principal, mejora rendimiento.
- Set de Instrucciones reducido: EL PENTIUM NO ES RISC, ES CISC PORQUE TIENE QUE MANTENER RETROCOMPATIBILIDAD.
- Gran número de registros: permite operar dentro del CPU sin usar memoria externa para operaciones simples.
- Pipelining: ejecuta de manera serializada instrucciones antes de terminar la última pq va haciendo etapas.
- Arquitectura Super Escalar: ejecutar instrucciones en paralelo, por ambas pipelines.

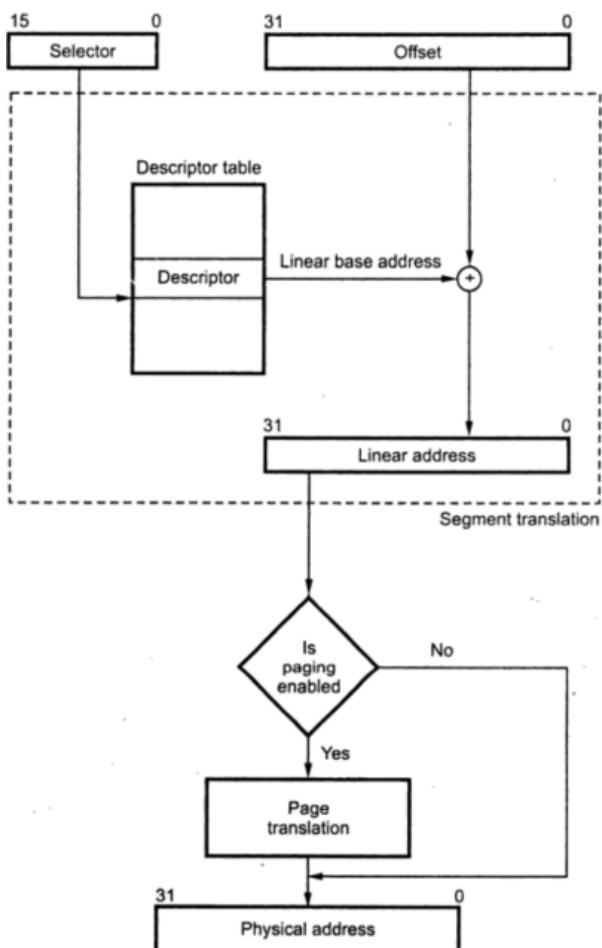
Modo Protegido

Incrementa el direccionamiento lineal a 4GB. Permite ejecutar programas con modos limitados. Además agrega:

1. Memoria Virtual: Segmentación y Paginación
2. Gestión de memoria y protecciones
3. Instrucciones especiales para multitarea
4. Mecanismos de paginación

Implementa memoria virtual, los programas pueden correr como si tuvieran memoria ilimitada. Usa dos mecanismos: Segmentación y Paginación.

Traducción de Dirección Lógica a Física en Modo Protegido



El selector del segmento es usado como índice para buscar el descriptor del segmento en la GTD (global descriptor table). El descriptor contiene la dirección base del segmento en cuestión, sumando el offset a la base se obtiene dirección lineal de 32btis.

Si la paginación está apagada la dirección obtenida es = a la física. Si no, se hace la traducción.

De donde viene el offset?

El offset viene del contador de programa u otro registro que dependa del dato que se quiere manipular y contenga una dirección.

Se explica más en profundidad la segmentación y paginación luego de ver los selectores, descriptores y tablas.

ACLARACIÓN:

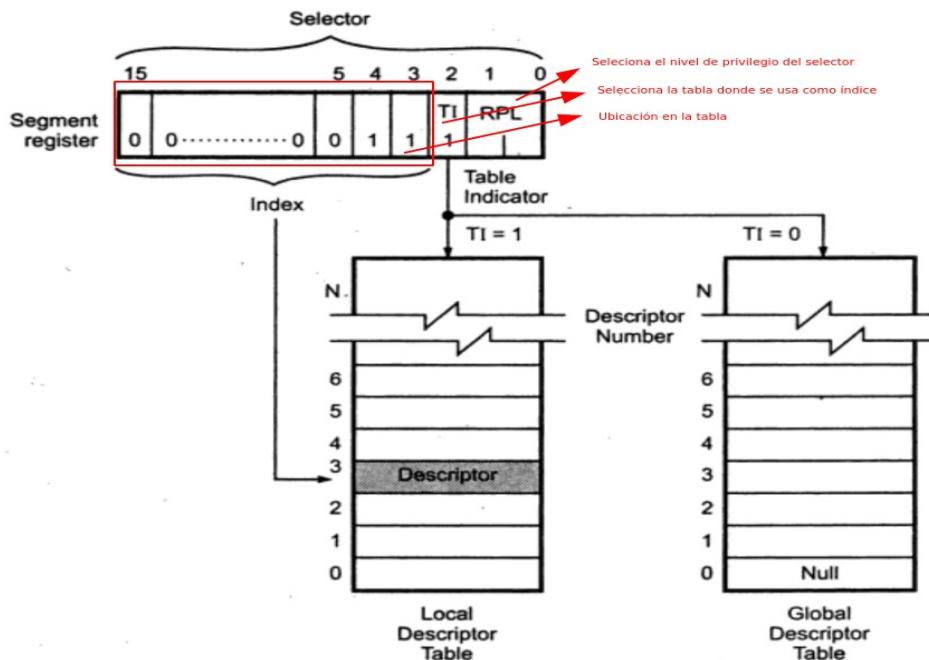
DIRECCIÓN LÓGICA: LA QUE PROGRAMARÍAS EN ASSEMBLER PONIENDO SELECTOR Y OFFSET(LA TOMÁS COMO SI TODO EL CPU FUERA DE 32 BITS)

DIRECCIÓN LINEAL: SI NO HAY PAGINACIÓN ES LA DIRECCIÓN POSTA DE 32 BITS QUE ESTÁ FÍSICAMENTE EN LA RAM, SI HAY PAGINACIÓN NO ES

IGUAL A LA FÍSICA Y HAY QUE HACER LA TRADUCCIÓN.DIRECCIÓN FÍSICA ES LA UBICACIÓN DE 32BITS "ELÉCTRICA" EN LA RAM OSEA LOS 1 Y 0 QUE TENES QUE PONER CON VOLTAJE PARA ENTRAR AL DATO EN LA RAM.

Selectores

Los selectores son los registros de los segmentos de la memoria. Se usan como índices para buscar en las tablas. El selector es una dirección lógica que tiene.



Los selectores tienen una parte invisible no accesible por el programador. Esta se llama **segment descriptor cache register**.

En la región oculta el CPU guarda información sobre el descriptor al que apunta, para hacer próximos accesos más rápidos. La porción visible la usan programas y la no visible el CPU.

Descriptoros

Proveen al CPU con la información que necesita para mapear una dirección lógica en una lineal. Son creados por

SEGMENT BASE 15 0										SEGMENT LIMIT 15 0										Bytes																	
BASE	G	D	0	AVL	LIMIT	P	DPL	S	TYPE	A	BASE	4	+ 4	8																							
31 24																																					
Access Rights Bytes																																					
<p>BASE: Base Address of the segment LIMIT: The length of the segment P: Present Bit : 1 = Present 0 = Not present DPL: Descriptor privilege Level 0 - 3 S: Segment Descriptor : 0 = System Descriptor 1 = Code or Data Segment Descriptor TYPE: Type of segment A: Accessed Bit G: Granularity Bit : 1 = Segment length is page granular 0 = Segment length is byte granular D: Default Operation Size (recognised in code segment descriptors only) 1 = 32-bit segment 0 = 16-bit segment 0: Bit must be zero (0) for compatibility with future processors AVL: Available field for user or OS</p>																																					
<p>Note : In a maximum - size segment (i.e. a segment with G = 1 and segment limit 19 0 = FFFFFH), the lowest 12 bits of the segment base should be zero. (i.e. segment base 11 000 = 000H).</p>																																					

16-bit visible selector	Hidden Descriptor
CS	
SS	
DS	
ES	
FS	
GS	

los compiladores o los SO. Describen genéricamente:

- Dirección base: 32bits
- Límite del segmento: 20 bits. Su interpretación depende del valor del bit G (Granularidad).

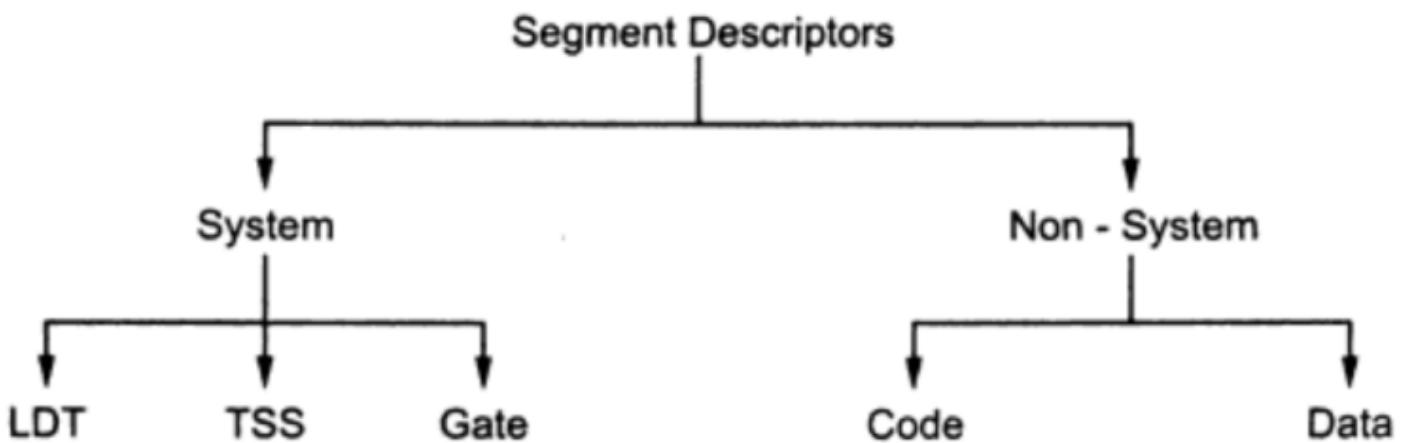
G=0 -> unidades de bytes hasta 1MB 2^{20}
g=1 -> unidades de 4KB hasta 4GB

- Tipo de segmento: S: de sistem o code
- Privilegio: DPL

- Si está presente físicamente en ram: P
- Si fue accedido antes: A
- Granularidad: G
- Tamaño de operandos D: 0:16bits,1:32
- Tamaño por defecto del mismo.

Los bits en TYPE y el bit S, le dicen al CPU el tipo de descriptor que es. El contenido del descriptor varía según lo que describe.

Tipos de descriptores de segmentos



Descriptores de segmentos de NO-SISTEMA (Código y Datos) S=1

Los bits en el byte de **ACCESS RIGHT BYTE** son interpretados por el CPU e incluyen los bits del descriptor genérico:

SEGMENT BASE 15 0								SEGMENT LIMIT 15 0		0	Bytes
BASE 31 24	G	D/B	0	AVL	LIMIT 19 16	ACCESS RIGHTS BYTE	BASE 23 16			4	
								+ 4			
								8			
D/B 1 = Default Instruction Attributes are 32 - Bits 0 = Default Instruction Attributes are 16 - Bits											
AVL Available field for user or OS											
G Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular											
0 Bit must be zero (0) for compatibility with future processors											

- Present P
- DLP
- Descriptor de segmento S
- S=1 -> Segmento de Código o Datos
- S=0 -> Segmento de Sistema o gate
- Executable E. E=0 -> data
- Dirección de expansión: hacia donde crece el segmento en memoria.
- Writable W: solo lectura o lectura y escritura

Descriptores de Segmentos de Sistema

a) Descriptor de LDT S = 0 Type = 2

Presentes en la GDT. Contienen información sobre la LDT específica de una tarea. Solo accesible con privilegio 0.

b) Descriptor de TSS S = 0 Type = 1, 3, 9, B

Se usa en multitarea. Describe el segmento de estado de tarea, donde se almacena el estado completo de los registros del CPU cuando se cambia de tarea. Es de 4 Bytes. Contiene información cómo: ubicación, tamaño y privilegios. También contiene la dirección a la próxima tarea, lo que permite anidar tareas.

c) Descriptor de GATE S = 0 TYPE = 4 a 7, C, F

Permiten al CPU realizar comprobaciones de protección. Existen 4 tipos de GATES:

- Callgates: cambian niveles de privilegios. (Llamadas a sistema, osea, sistema operativo)
- Taskgates: permiten cambiar tareas en ejecución.
- Interruptgate y Trapgate: usados para especificar ISR y ESR.

Tablas de Descriptores

Los descriptores de segmentos están agrupados en tablas de manera continua. Rango máximo de cada tabla de 64Kbytes. Cada tabla tiene su registro con la ubicación en memoria.

Tabla Global de Descriptores (GDT)

Puede ser usada por todos los programas. Puede tener cualquier descriptor de segmento salvo los de interrupciones.

El **GDTR** es el registro de la tabla global. Está en el CPU. Los dos bytes de menor peso dan el límite o tamaño de la tabla. Los 4 bytes superiores dan especifican la dirección **lineal** de la base de la GDT.

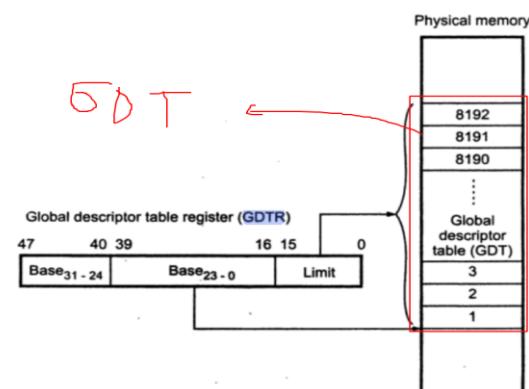


Tabla local de Descriptores (LDT)

Se configuran por el sistema para tareas o procesos individuales. Pueden tener acceso limitado solo al programa que la crea o no.

El **LDTR** es el registro de tabla de descriptores locales. Formado por 16bits. No posee más información. Se usa como índice para buscar el descriptor de la LDT en la GDT. Pasos:

1. Accedo a la GDT con el LDTR
2. Cargo el LDT DESCRIPTOR como base más su offset.
3. Accedo mediante ese offset al descriptor dentro del espacio de direcciones de la LDT.

Si el bit TI en el selector está en 1, se usa como índice en una LDT.

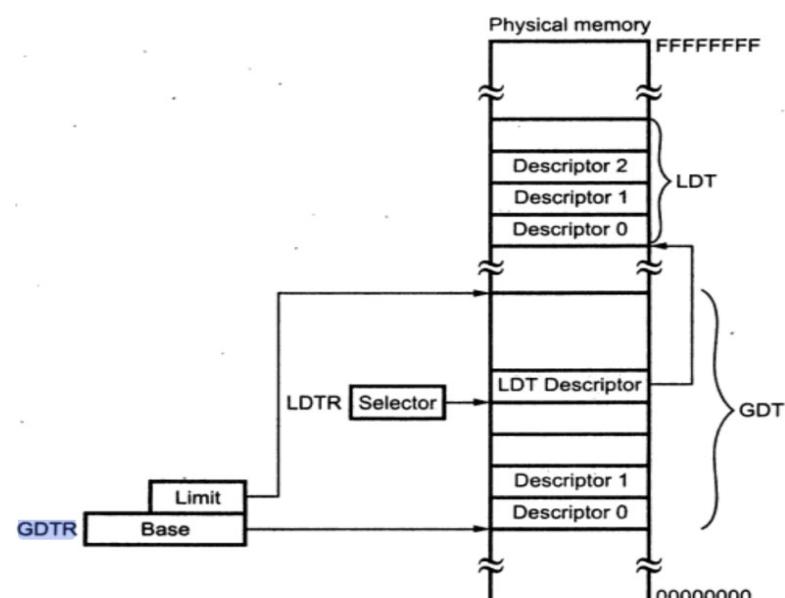
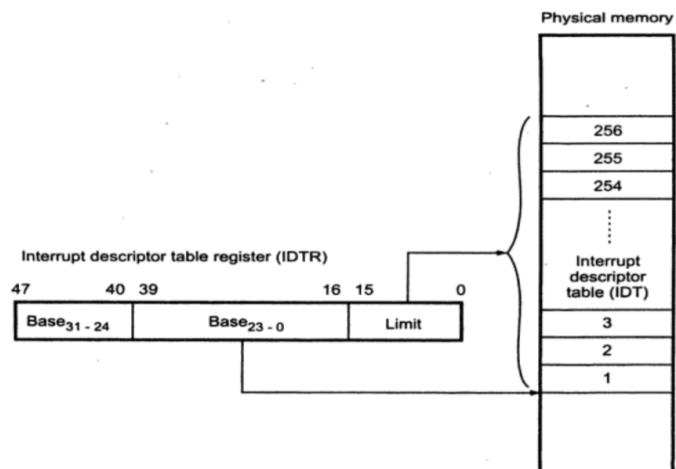


Tabla de Descriptores de Interrupciones(IDT)

Contiene los descriptores de segmentos que definen las interrupciones o las ISR(interrupt service routine) y ESR(exception service routine).

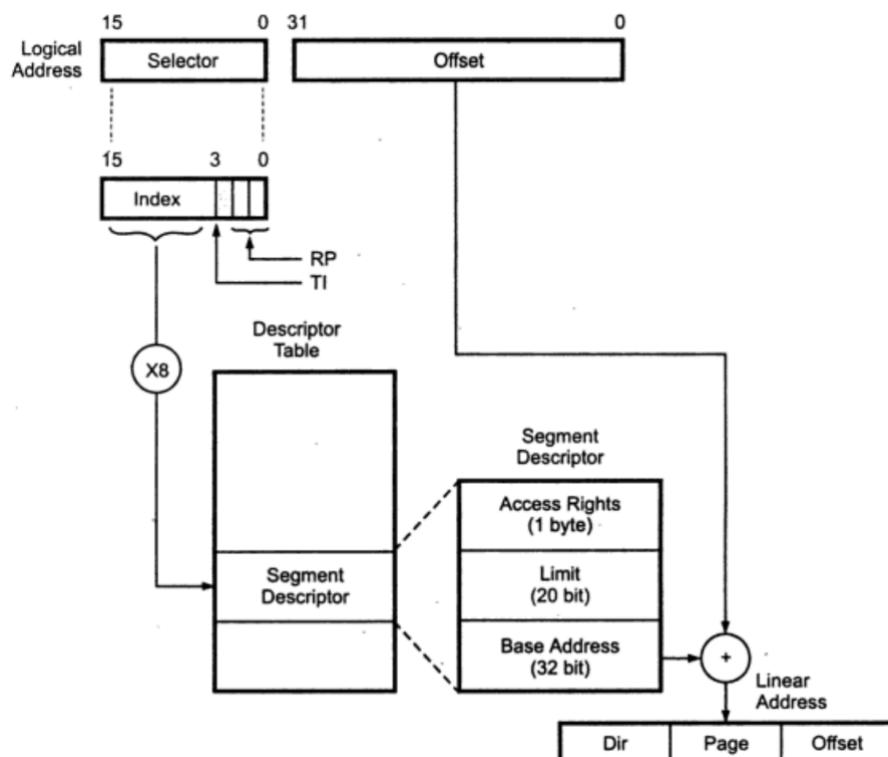
El **IDTR** es de 48bits, 2 bytes de menor peso dan el límite (en bytes). Los 4 bytes de mayor peso dan la base.

Pentium solo soporta 256 interrupciones



Segmentación

Proceso que convierte las direcciones lógicas en lineales.



Los 13 bits de INDEX del selector se usan para acceder al descriptor del segmento en la GDT. El index debe ser multiplicado por 8 porque el descriptor requiere 8 bytes.

EXPLICACIÓN DE LO ANTERIOR

El libro dice x8 pero en realidad lo que hace es volver a colocar los 3 bits que faltan al index para hacerlo de 16bits, es decir: El index por si solo tiene 13 bits pero eso no sirve, entonces le pone 3 ceros al final: 0000000000000000->Posición 0 no hay descriptor. Ahora el index apunta al primer descriptor: 0000000000001000->1er descriptor en decimal es 8, y si hablamos de bytes, apunta al byte 8 con los 8 bytes

que ocupa el descriptor hacia el inicio de la tabla. Ahora si el index vale 2 0000000000001000->2do descriptor en decimal eso vale 16, dejando atrás 8 bytes que son los que el cpu toma como el descriptor dos, finalmente si vamos al index 3 00000000000011000->3er descriptor en decimal eso vale 24, dejando de nuevo 8 bytes atrás y así sucesivamente hasta 111111111111000->descriptor 8192

El CPU suma la dirección base en el descriptor de segmento con el offset que puede ser el contador de programa u otro registro que depende del dato que se quiere manipular.

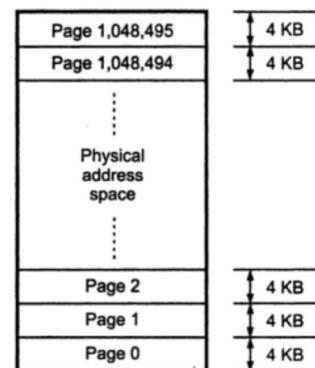
La dirección lineal si no hay paginación es igual a la física.

Además hace la comprobación de privilegios, donde:
 RPL \geq DPL \rightarrow Da
 RPL < DPL \rightarrow No da acceso + interrupción

Paginación BIT PG = 1 EN CR0(CONTROL REGISTER 0)

Transforma la dirección lineal dada por la segmentación en la dirección física. Se usa cuando el SO quiere llevar a cabo múltiples tareas virtuales, memoria virtual o protección por paginación .

La paginación organiza el espacio físico direccionable en “páginas” de 4096 bytes de longitud o 4KB. \rightarrow Las páginas ya son espacios físicos de memoria.

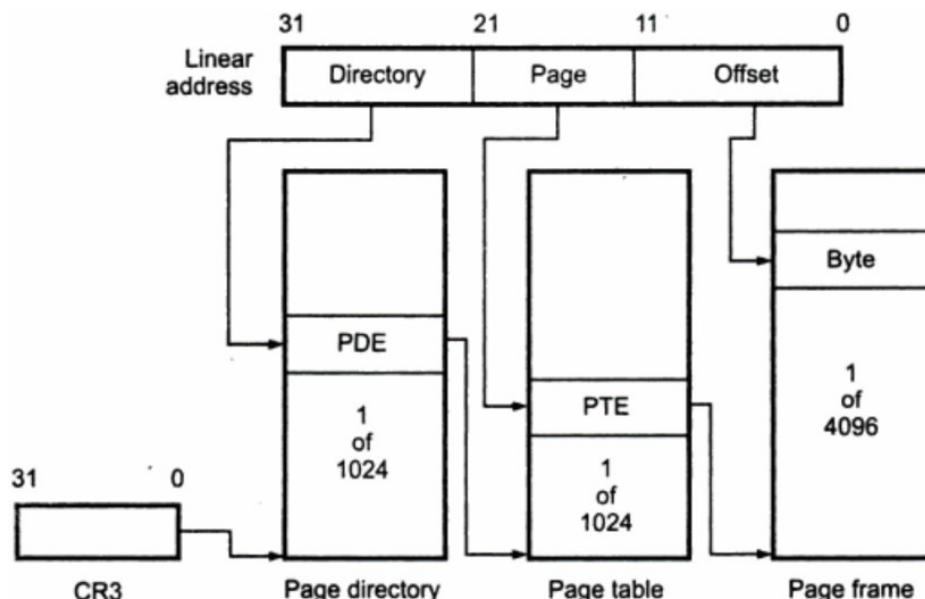


La dirección lineal se separa en 3 partes:

- Directory: índice en la página de directorio
- Page: índice en la tabla de páginas
- Offset: determina cuál de los 4096 bytes de la memoria del cuadro de páginas seleccionado por la tabla de páginas será usado.

MEMORIA VIRTUAL: LOGRA QUE EL PROGRAMADOR SE ABSTRAIGA DE LA MEMORIA FÍSICA Y SOLO VEA UN CONJUNTO DE MEMORIA LINEAL.->definición que le gusta al taffe.

Funcionamiento de la paginación



El CPU usa dos niveles de tablas (almacenadas en RAM también) para traducir la dirección lineal en la física.

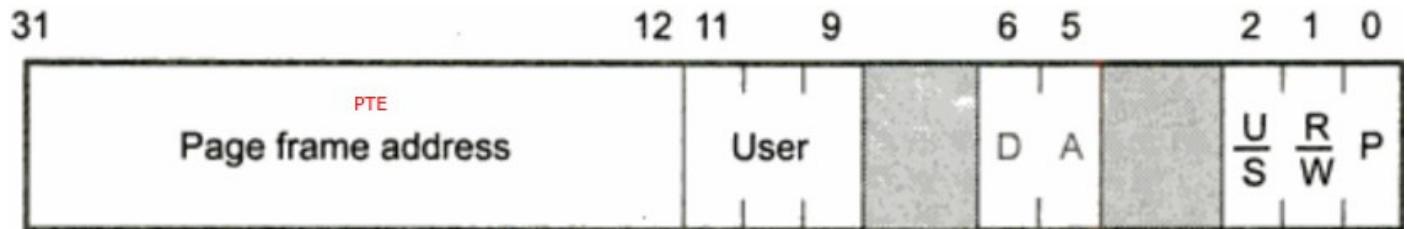
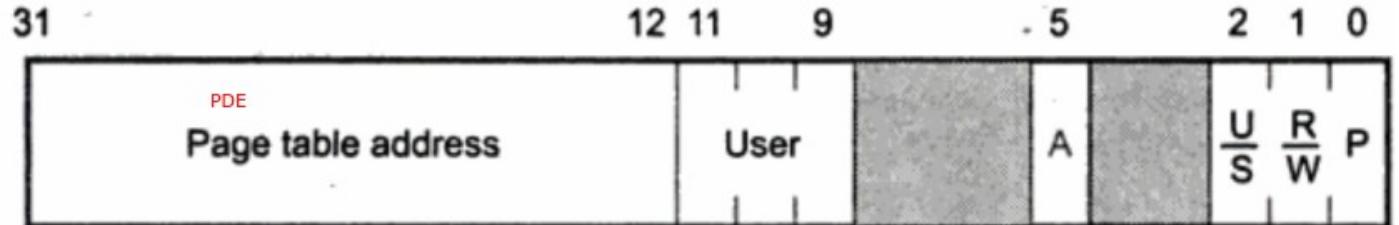
Los 10 bits más significativos, **DIRECTORY**, se usan de índice en un directorio de tablas de páginas.

Los bits **PAGE** se usan de índice en la tabla de páginas determinada por el directorio de páginas.

El **OFFSET** selecciona una de los 4096 bytes de memoria del cuadro de página seleccionado por la tabla de páginas.

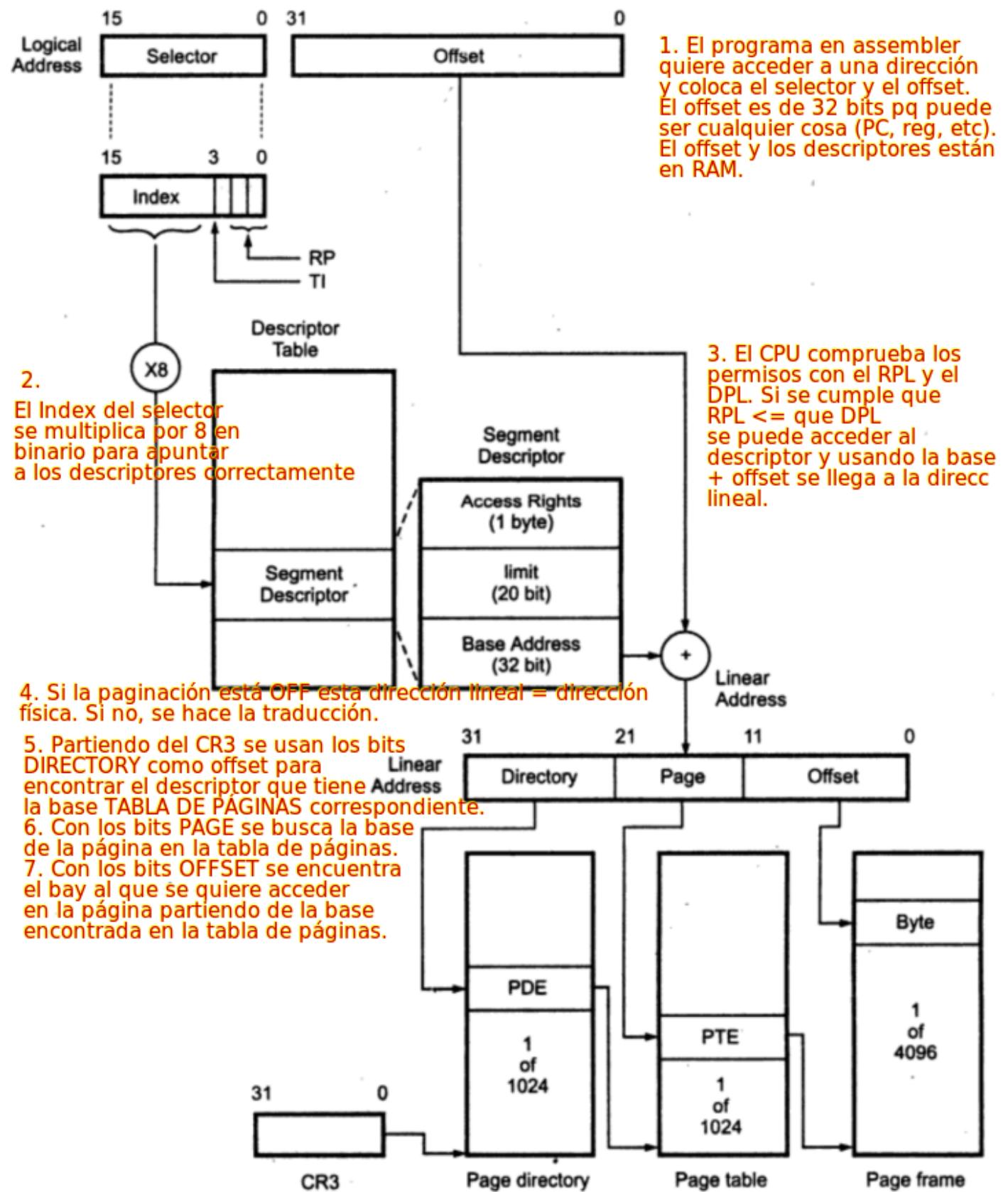
La base del directorio de páginas se guarda en el CR3.

PDE y PTE DESCRIPTORS



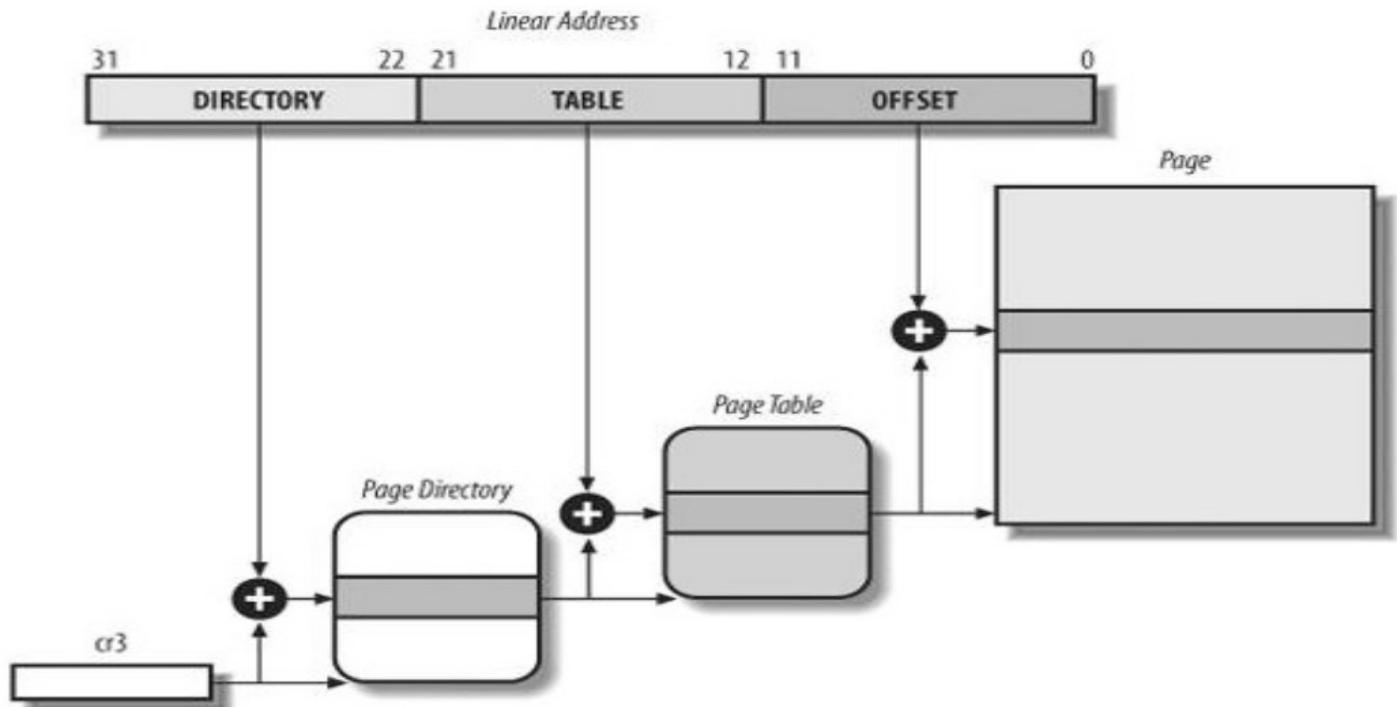
Contienen las direcciones base de las tablas de páginas y de los marcos de páginas correspondientes. Además contienen bits de uso libre para el usuario y bits de control de privilegios y lectura / escritura. El bit D en el PTE lo pone en 1 el cpu cuando se escribe en ese marco de página. De esta forma sabe que lo debe guardar en disco cuando lo vaya a cambiar por otro. El bit P indica si está presente actualmente en RAM o no.

PASO A PASO DE LA TRADUCCIÓN LÓGICA A FÍSICA EN MODO PROTEGIDO SIN TLB



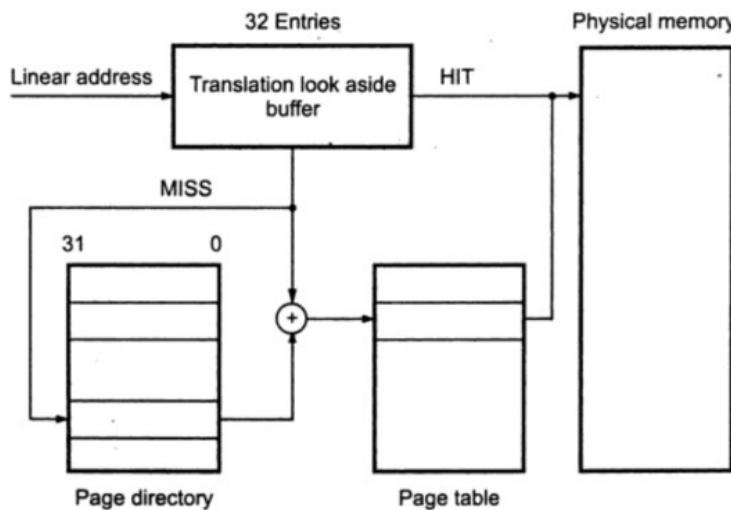
Donde dice bay quise poner byte. Paja arreglarlo.

Otra imagen de paginación un poco más entendible:



TRANSLATION LOOKASIDE BUFFER (TLB)

Para evitar que el rendimiento del sistema se degrade demasiado por el método de memoria virtual de paginación, pq el CPU tiene que hacer dos lecturas de tablas hasta llegar al dato deseado, se implementa un caché dentro del CPU, el TLB.



El TLB almacena hasta 32 entradas de tablas de páginas con sus correspondientes páginas de 4KB, con lo que almacena 128K bytes de direcciones de memoria.

ACLARACIÓN: GUARDA LAS DIRECCIONES TRADUCIDAS DE DIRECCIONES LINEALES ACCEDIDAS RECENTEMENTE, NO LOS DATOS, ESTOS SIGUEN ESTANDO EN RAM

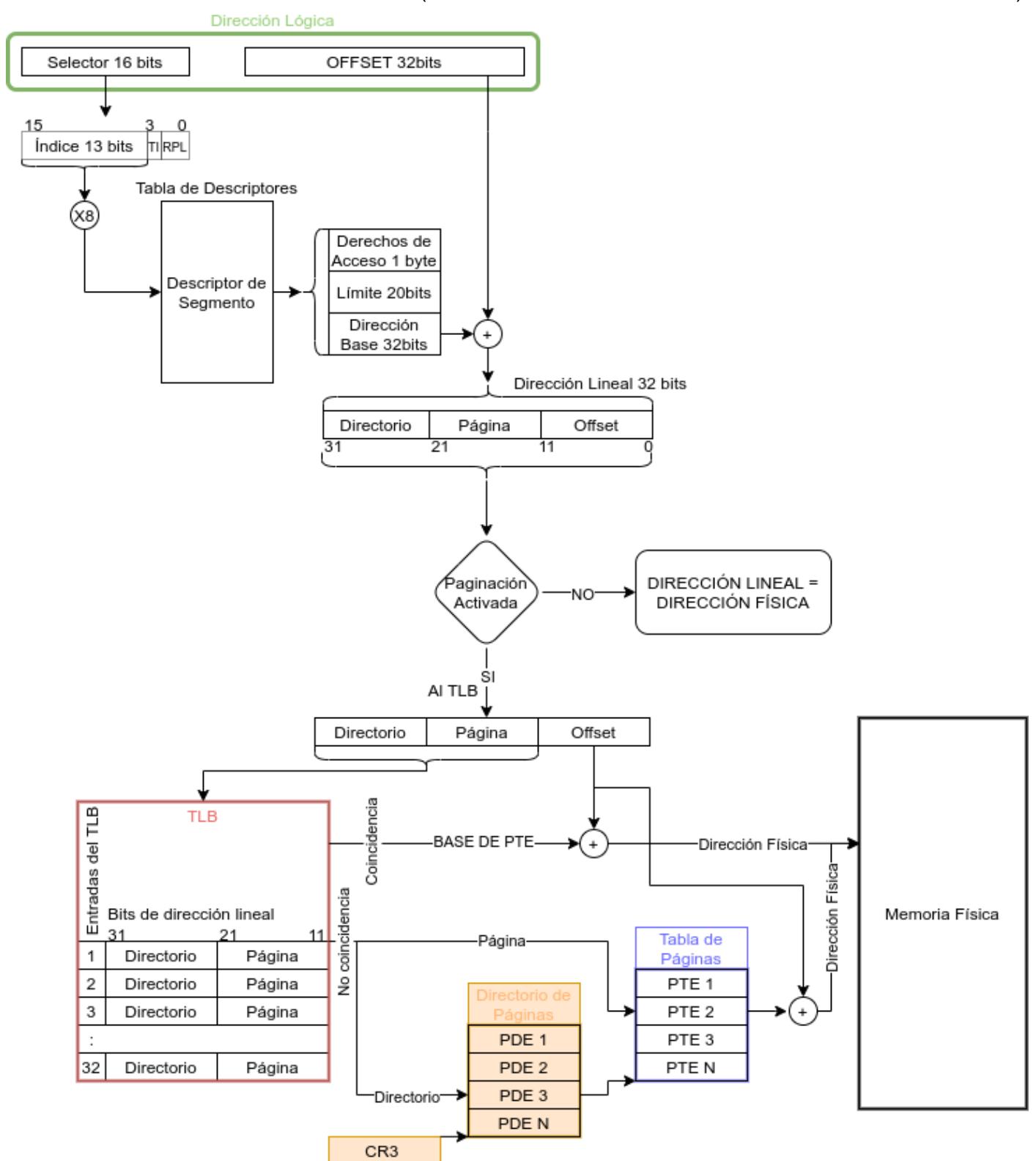
Cuando un proceso crea la dirección lineal que se mapea a un PTE ya en la caché el CPU se ahorra dos lecturas de memoria. Tiene una tasa de éxito de 98%, solo el 2% de las veces que quiera leer algo de memoria tendrá que recurrir al mecanismo de paginación completo. Pasos de funcionamiento:

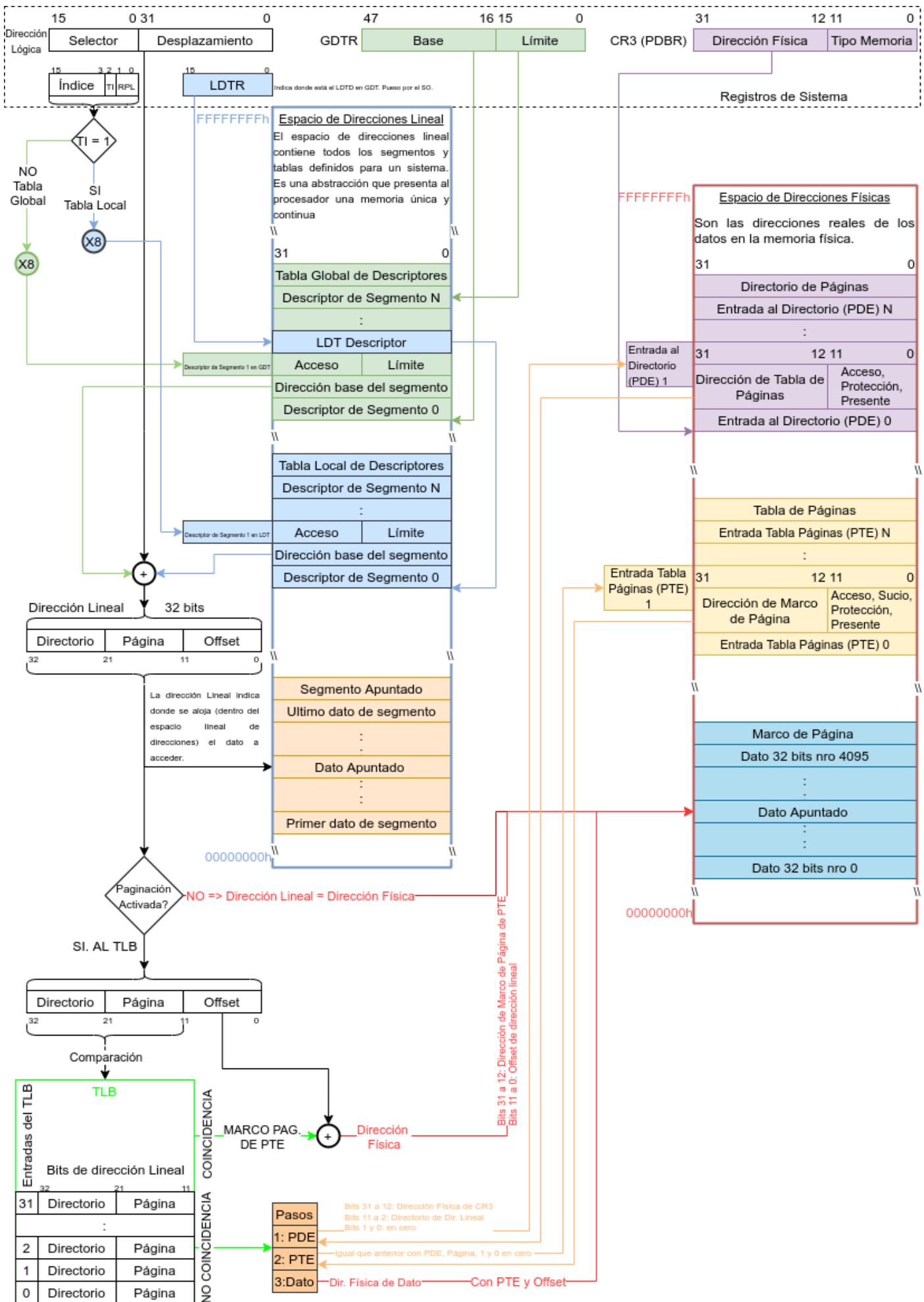
- 1) Entra dirección lineal generada por el proceso en ejecución
- 2) Si está en caché, se lee directamente de memoria física sin pasar por tablas. Ahorro 2 accesos
- 3) Si no está, busca con los bits DIRECTORY en el directorio de páginas
- 4) Luego le suma el offset dado por los bits PAGE y encuentra el PTE correspondiente
- 5) Con el offset de la dirección lineal accede al dato en la memoria física.

OPERACIÓN DE PAGINACIÓN

El mecanismo de paginación recibe la dirección lineal de 32 bits de la unidad de segmentación. Los 20 bits superiores se comparan con las 32 entradas del TLB para determinar si hay una coincidencia. Si la hay, la dirección física de 32 bits se calcula y se coloca en el bus de direcciones. Si la entrada de la tabla de páginas (PTE) no está en el TLB, el procesador lee el apropiado PDE. Si en este PDE P=1, el procesador lee el PTE

y pone en uno el bit Accedido. De la misma manera, si P=1 en el PTE, se actualizan los bits Accedido y Dirty. Luego, el procesador almacena los 20 bits superiores de la dirección lineal, leídos de la tabla de páginas, en el TLB para accesos futuros. Sin embargo, si P=0 ya sea para el PDE o el PTE, se genera una falla de página (Excepción 14). Dicha excepción también se genera si el acceso a memoria viola los atributos de protección (U/S o R/W). Si el procesador quiere acceder a espacios de memoria física cuya información no está en la caché, entonces el mismo examina las 32 entradas y descarta las PTE menos usadas recientemente. Luego pone en su lugar las nuevas PTE. Este método de actualización de caché es conocido como LRU (Menos Usado Recientemente).





Protección

Sistema multitarea, problema cuando dos procesos tratan de acceder a memoria al mismo tiempo. La sección de memoria usada por un proceso o SO debe ser protegida. Se debe prevenir también que por dirección incorrecta en un programa se corrompa secciones de otro o del SO. Pentium usa:

Protección por segmentación

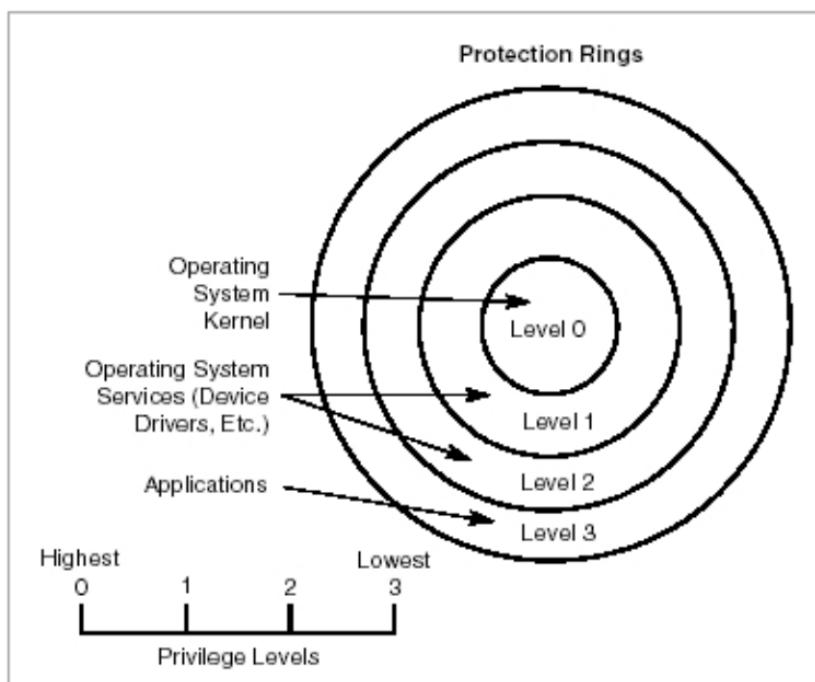
Cuando se quiere acceder a un segmento de memoria:

1. Verifica que exista el descriptor en la GDT o LDT (según el bit TI) para el selector de dicho segmento. Excepción si no existe.
2. Chequeo de TIPO: chequea que el descriptor sea del tipo correcto para el segmento. Se usa para detectar si un programa intenta usar segmentos en formas no permitidas.
3. Chequeo de LÍMITE: previene direccionar fuera del segmento elegido. También chequea los bits de dirección de crecimiento (ED) y de tamaño de bloques (B: big).

Protección por Niveles de Privilegios

Se usa para aislar y proteger programas entre sí y al SO. Cuatro niveles. PL0 el de más privilegio, usado por SO. Puede usar todas las instrucciones y acceder a todas las E/S. PL3, el de menor privilegio. Solo usa instrucciones de propósito general.

Linux solo usa PL0: kernel o núcleo y PL3: usuario.



- Current Privilege Level: **CPL**: es el nivel de priv que tiene el código en ejecución y que quiere acceder a un descriptor de segmento. Está almacenado en la porción oculta del selector de segmento que apunta a la región de memoria donde se encuentra el código assembler en ejecución.
- Descriptor Privilege Level: **DPL**: es el nivel de priv del objeto a ser accedido. Puede ser: descriptor de sistema o de no-sistema.
- Requested Privilege Level: **RPL**: valor que se escribe en los primeros 2 bits del selector. Indica al CPU con el PL que se quiere acceder al descriptor. Si es numéricamente mayor que CPL, cpu usar el RPL, si no, usa el CPL.

Acceso a Segmento de Datos Restringido por Privilegios

Cuando se intenta acceder a un segmento cargando un selector de segmento en la porción visible del data segment register, el cpu verifica los 3 tipos de niveles y se debe cumplir:

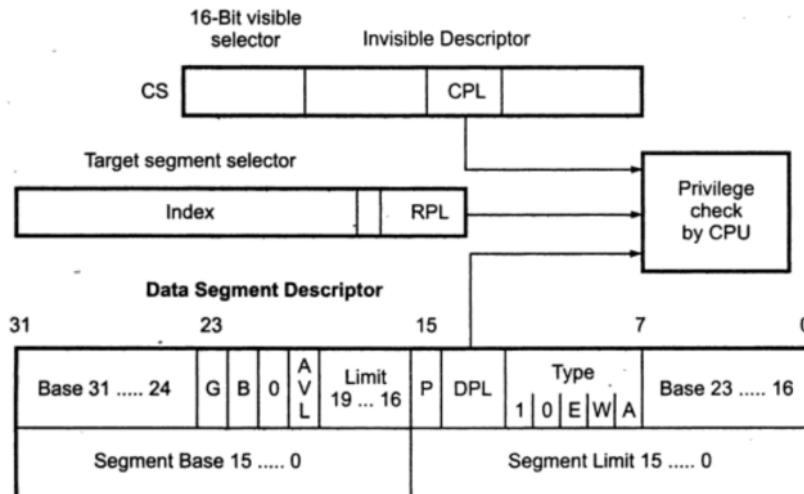
Numéricamente: DPL >= maximo CPL y RPL

Esto quiere decir que para acceder a un segmento de datos con priv 3, osea el menor de todos, lo puede hacer cualquier proceso y el SO.

En cambio, si el segmento tiene priv 0, osea el mayor nivel de priv posible, solo puede acceder el SO que corre con CPL 0.

DPL	CPL	RPL	
2	0	1	Valido
3	1	2	Valido
1	1	0	Valido
1	2	0	Invalido
2	2	3	Invalido

Gráficamente las comprobaciones de privilegios se hacen en:



1. Usuario

carga en CS el selector del segmento de datos al que desea entrar. En la porción invisible el cpu colocó el CPL del código en ejecución.

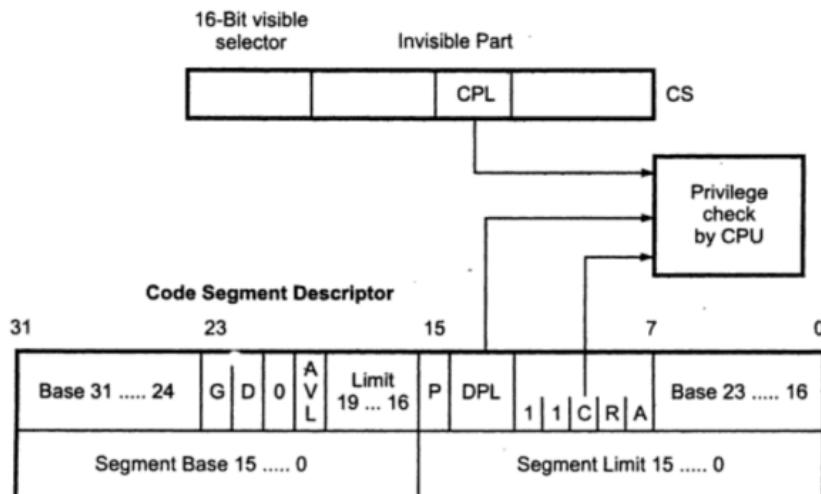
2. En los 16bits del selector están los 2 bits de RPL, tambien los puso el usuario.

3. El descriptor del segmento posee su DPL.

4. Al comprobar se debe cumplir que:

0 DPL \geq max [RPL o CPL] numéricamente.

Acceso a Datos en Segmento de Código Restringido por Privilegios



CPL - Current Privilege Level

RPL - Requestor's Privilege Level

DPL - Descriptor privilege level

Para leer datos en segmento de código hay 3 formas:

1. Cargar en el CS el selector a un segmento legible, ejecutable y no conforming.->Solo accede = PL

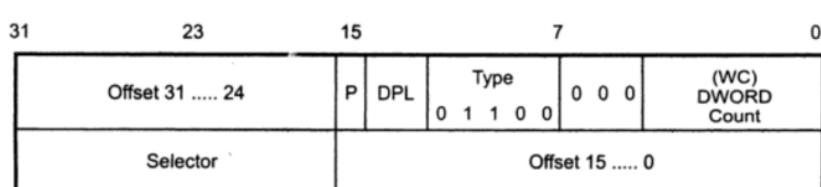
2. Cargar en el CS el selector a un segmento legible, ejecutable y conforming. ->Accede PL0

3. Sobrescribir el CS para leer datos

4 que están en el mismo segmento. En este caso CPL = DPL.

0 CallGates

Son un tipo especial de descriptor, no definen un espacio de memoria. Actúan como interfaz entre segmentos de código con distintos niveles de privilegio. Es el único mecanismo que permite llamar código de mayor privilegio. El callgate define un segmento de código y offset exactos donde se transmitirá el control.



codigo.

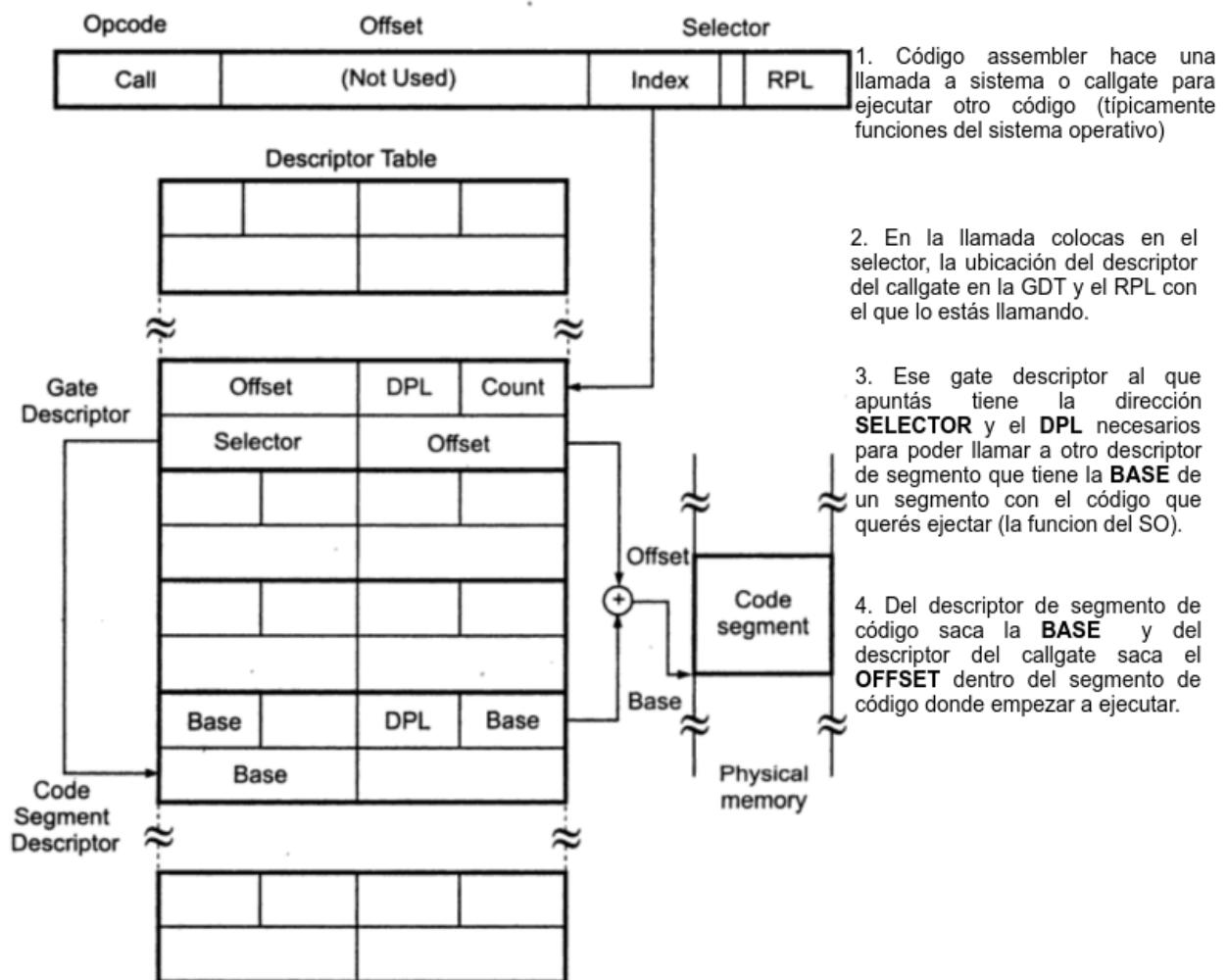
El descriptor del callgate se encuentra en la GDT o LDT. Cuando un programa hace un call a otro segmento, el mismo es ubicado en la porción visible del CS (code segment) y el descriptor del callgate en la porción invisible.

El offset se toma del CALL DESCRIPTOR y no de la LLAMADA. **EL OFFSET ES DE 32 BITS Y ESTÁ DIVIDO EN DOS PARTES DE 16 EN EL DESCRIPTOR.... PORQUE LOS DE INTEL SON UNO CLDOS**

Las puertas o gates son inicializadas por el SO y usadas por los programas de usuario, que típicamente tienen menor nivel de privilegio y las funciones a las que quieren acceder las piden por callgates al SO.

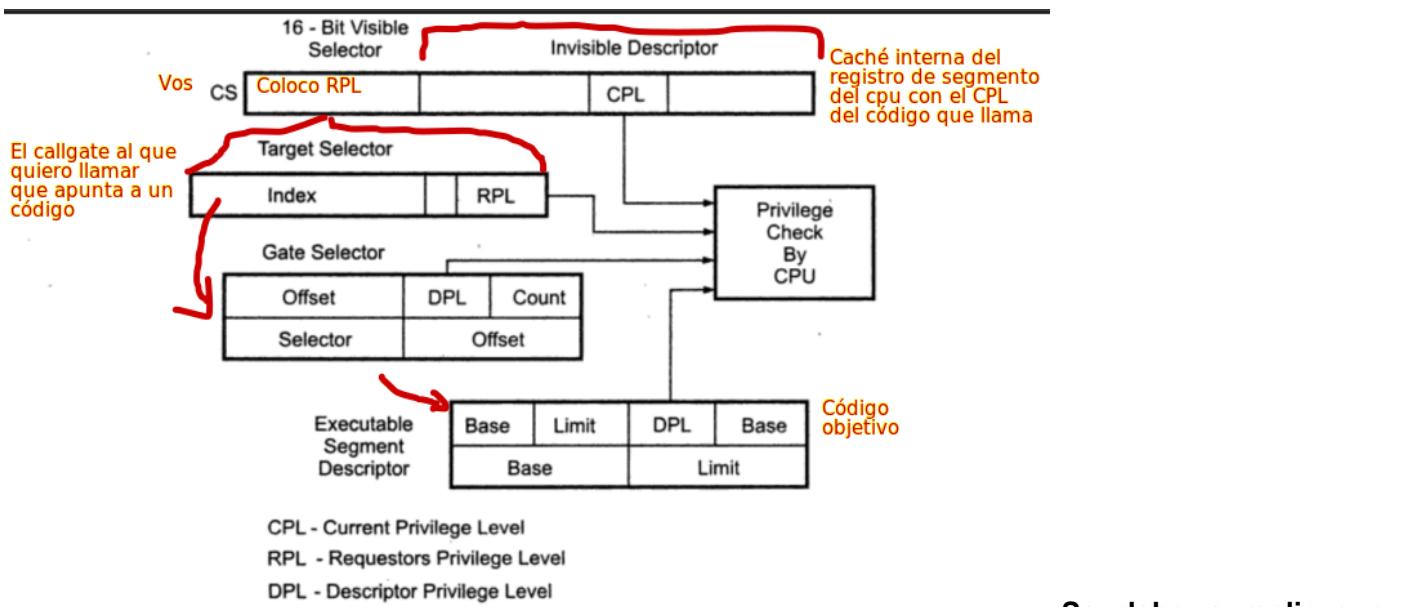
Quien llama al CALLGATE debe tener los privilegios necesarios.

Requisitos para que un callgate se ejecute con éxito:
Tener espacio en la pila. Permisos para hacer el call. CPL llamador <= DPL de callgate llamado



Descripción funcional del CallGate

DURANTE EL PROCESO DE TRANSFERENCIA INDIRECTA DE CONTROL SE CHEQUEAN 4 TIPOS DE NIVELES DE PRIVILEGIOS:



Se debe cumplir que:

Target DPL <= Max(RPL,CPL) <= Gate DPL [numéricamente]

Alternativa al CallGate

Alternativa al uso de CallGates es usar interfaces trampa. Consiste en invocar funciones del sistema operativo mediante interrupciones de soft. La interfaz trampa usa la IDT que puede contener distintos tipos de descriptores:

- Interrupt Gates: son las ISR definidas en el kernel y manejan interrupciones propiamente dichas. Estos descriptores los definimos antes.
- Trap Gates: son descriptores similares a los CallGates.
- Task Gate: apunta a un descriptor de TSS. Se usa para cambiar de tarea en ejecución.

Interrupciones y Excepciones

Son maneras especiales de transferir el control de CPU. Trabajan como calls no programadas y alteran el flujo normal del programa.

- Interrupciones: manejan eventos externos al cpu asíncronos.
 - Enmascarables: las puedo desactivar por software, controlables por usuario
 - No enmascarables: están siempre activas. [jodete]
- Excepciones: manejan condiciones internas del cpu, detectadas por este mismo durante la ejecución de un programa. Fuentes de excepciones:
 - Internas del CPU. Cuando hay algún error interno
 - Programadas por usuarios. Llamadas interrupciones de software.

En la IDT se guardan los 3 tipos de descriptores que se pueden llamar mediante interrupciones o excepciones. A la tabla entra lo que se llama un **VECTOR DE INTERRUPCIÓN** que corresponde con el descriptor correspondiente para manejar dicha interrupción o excepción.

Multitareas

Compartición de Tiempo

- Permite múltiples usuarios
- Provee uso económico de recursos
- Invisible al usuario
- Múltiples usuarios a la vez

Multiusuario vs Multitarea

- PC multiusuario: puede hacer varias tareas por usuario
- PC multitarea: puede hacer varias tareas para UN usuario

Métodos de Planificación para SO multiusuario

Planificación por Porción de Tiempo

Un componente del SO determina cuándo cambiar de tareas: scheduler, supervisor o planificador

Ventaja: todos los usuarios o tareas son atendidos a intervalos aproximadamente regulares

Desventaja: al aumentar los usuarios o tareas los tiempos asignados son más cortos y cada tarea demora más en terminar. Para estos casos se prefiere planificación por prioridad.

Planificación por Prioridad

Cada tarea tiene asignada una prioridad y pueden interrumpir tareas de menor prioridad. Permite terminar los procesos más importantes primero.

Cambio de Contexto

El contexto de una tarea son los registros, punteros, variables, stack, etc. Cuando ocurre un cambio de tarea el contexto debe ser guardado, junto con su dirección en un segmento especial. Para volver a la tarea el SO usa la dirección guarda en un registro especial para acceder a todo el contexto guardado.

Además de los cambios de tareas normales, Pentium soporta otros dos tipos de manejos de tareas: excepciones e interrupciones.

Cada tarea tiene una LDT y un DIRECTORIO DE PÁGINAS separados del resto con direccionamiento distinto. Esto permite mantener los espacios de memoria de las tareas separados y previene que se interfieran.

Un cambio de tarea puede ocasionar también que se cambien los privilegios de ejecución (si se pasa de programa usuario a SO por ejemplo), esto cambia el espacio de memoria direccionable. Por otro lado, el **NIVEL DE PRIVILEGIO DEL STACK SEGMENT DEBE SER IGUAL AL DEL CODE SEGMENT EN EJECUCIÓN**, por lo que cada vez que cambia el nivel de privilegio del código en ejecución se debe cambiar de stack segment. El intercambio de segmento se trata más adelante.

Registros y Descriptores para Multitarea en Modo Protegido

Task State Segment

Tipo especial de segmento usado para manejar tareas. Funciona como un borrador solo accesible por el CPU para almacenar el contexto de ejecución de una o más tareas. Se divide en dos secciones o sets:

Set Dinámico

Se actualiza cada vez que el CPU cambia de tarea, guardan el estado del CPU. Incluye:

- Registros generales.
- Registros de Segmentos.
- Registros de banderas: permiten ejecutar sentencias condicionales y saltos condicionales.
- Puntero de instrucción: permite reiniciar la tarea donde se la detuvo.

- Link de retorno: permite volver a la tarea anterior.

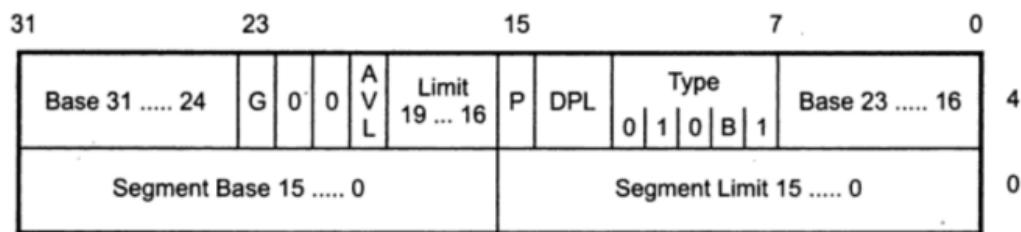
Set Estático

El CPU solo lee estos registros. Incluye:

- Selector para la LDT de la tarea
- PDBR o CR3: contiene la dirección base del directorio de páginas de la tarea.
- Punteros a stack de privilegios 0 a 2
- T-BIT: pone en modo debug al cpu
- I/O map offset.

TSS Descriptor

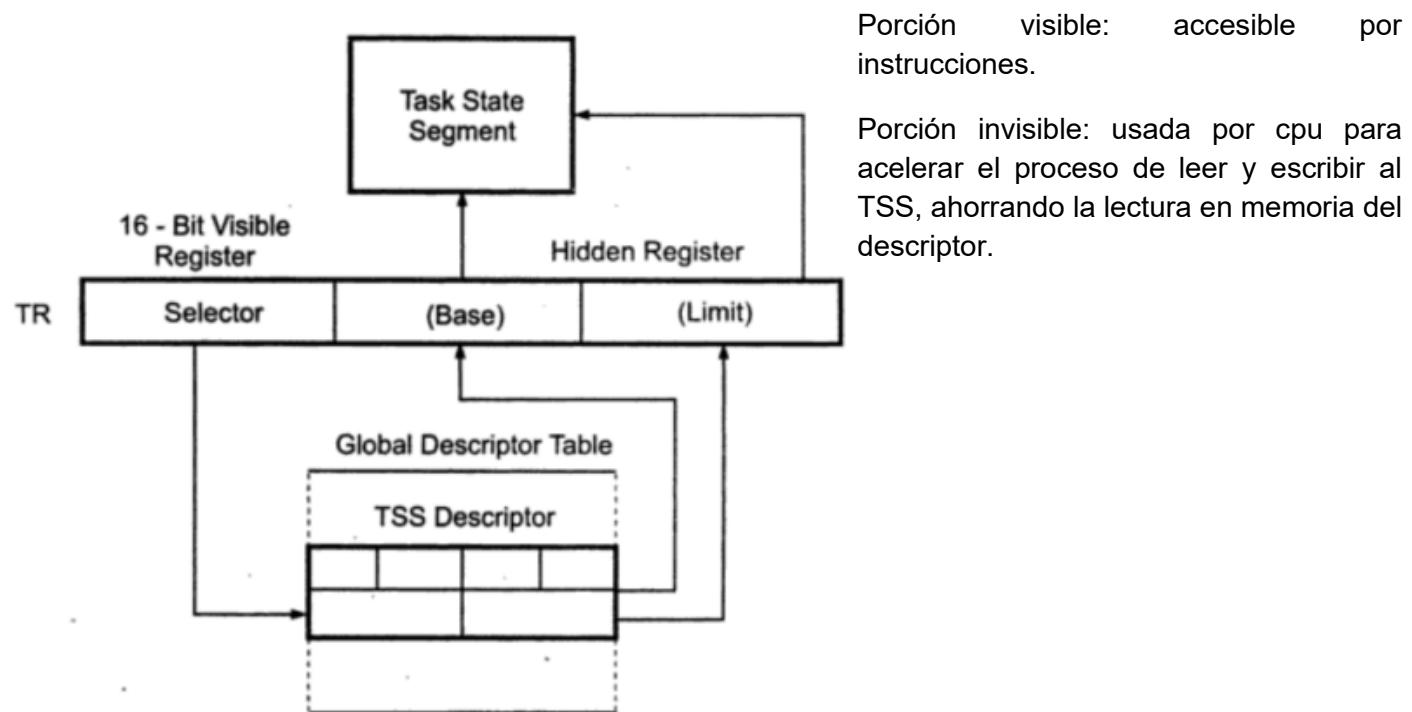
Se encuentra en la GDT.



Bit B: indica si está actualmente en ejecución
El segmento debe ser mínimo de 104 bytes -> Límite >=103

Task Register (TR)

Especifica la tarea actualmente en ejecución al apuntar al TSS correspondiente de esta tarea. El TR es un selector.



LTR: Load Task Register

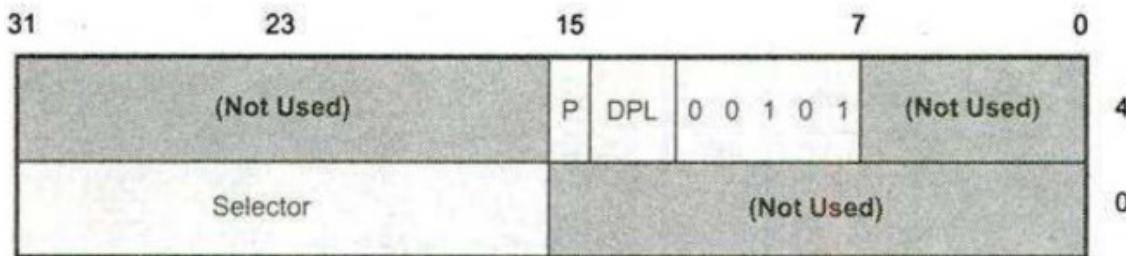
Instrucción de nivel cero. Carga en el TR el selector del TSS y su parte invisible del TSSD.

STR: Store Task Register

Instrucción de nivel 3, usuario. Carga en un registro cualquiera la porción visible del TR.

[Advertencia, me canse de hacer este resumen respetablemente, de ahora en más no me hago responsable de lo que escribo]

Task Gates y Task Gate Descriptor



El **taskgate** tiene su propio descriptor, no define un segmento de memoria sino que actúa como un punto de interfaz entre el código de usuario y un TSS. Da una referencia indirecta y protegida (por el DPL) al TSS.

El **taskgate descriptor** define un selector al descriptor de un TSS que identifica una sola tarea. [si no entendiste la respuesta está en tucorazón]

El DPL del taskgate controla el derecho de uso del descriptor para hacer un cambio de tarea. Se debe cumplir: CPL y RPL <= DPL (numéricamente, osea el RPL y CPL deben ser de mayor priv que DPL)

Cambio de Tareas (Task Switching)

Luego de cada cambio de tarea, el CPU marca la tarea como ocupada colocando el B-Bit en 1 en el descriptor de TSS de la tarea en ejecución. Los cambios de tareas no pueden ser recursivos (menos mal).

Cambio SIN Task Gate (método directo)

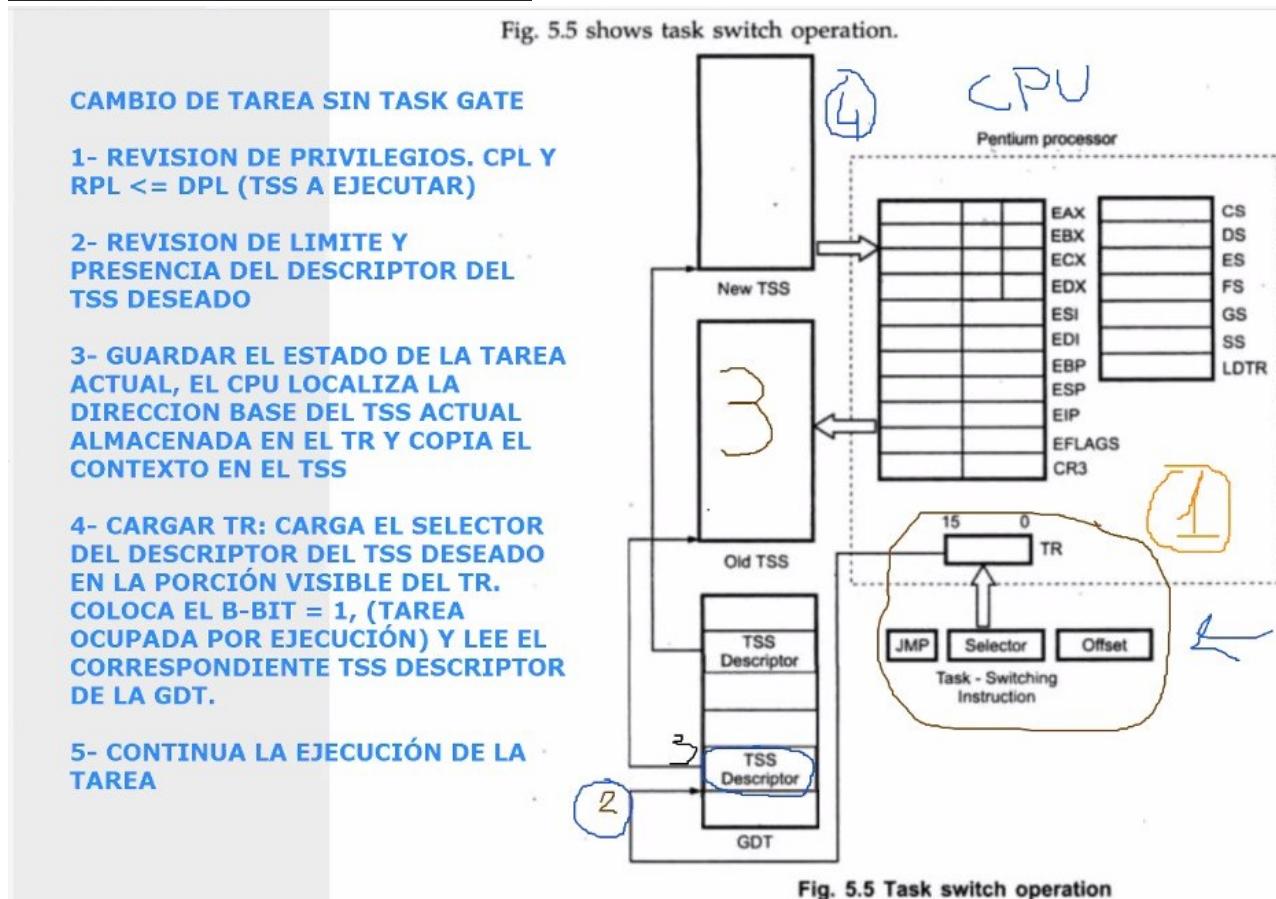


Fig. 5.5 Task switch operation

Cambio CON Task Gate (método indirecto)

Es la misma verga que lo anterior solo que compara el DPL del taskgate con el CPL y RPL del taskgate selector, osea el que uno pone en el CS para hacer la llamada a sistema. A diferencia del anterior que lo hace entre el selector que metemos y el TSS descriptor.

Tareas Anidadas

La misma poronga que las subrutinas anidadas. Teniendo en cuenta que se pone la bandera NT en 1 en el registro EFLAGS para que el cpu sepa que el backlink del tss que termina o cambia se corresponde con una tarea que lo anidó.

U2: Sistemas Operativos

¿Qué es un SO?

El SO proporciona a los programas y usuarios un modelo de pc más simple que administra todos los recursos del sistema. El usuario interactúa con un SHELL o GUI que no forman parte del sistema operativo (pero acceden a funciones del mismo).

Hay dos enfoques. El enfoque de “Máquina extendida” donde el trabajo del SO es el de proporcionar abstracciones para ocultar el hardware del programador. El de “Manejador de Recursos” donde el SO cumple la tarea de administrar el hardware y asignarlo de manera ordenada y justa a todos los programas.

En la práctica, el SO proporciona a los programadores una capa de abstracción de los recursos del hardware disponibles a la vez que los administra (tiene que hacer las dos cosas). Su tarea principal es ocultar el hardware. Administrar los recursos del sistema requiere multiplexión:

- En el tiempo: procesos o programas y usuarios se turnan para usar el sistema.
- En el espacio: los clientes de un mismo sistema obtienen una porción de los recursos. Por ej: 1GB de ram en un sistema con 4GB.

Modos de Operación del Software

- Kernel: es el modo en el que se ejecuta el SO. Tiene acceso a todo el hardware. Sería como un nivel de privilegio 0, el más alto.
- Usuario: es donde se ejecuta el resto del software. PL = 3. Tiene instrucciones limitadas. Accede a funciones del hardware mediante llamadas al SO. El nivel más bajo en modo usuario es el SHELL o GUI.

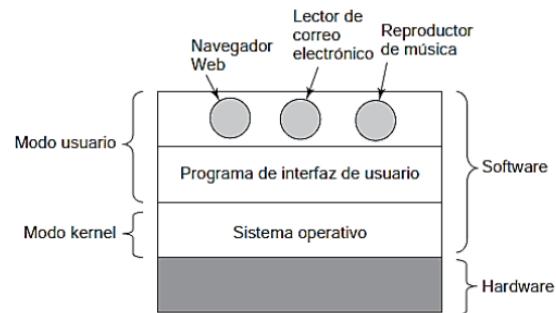


Figura 1-1. Ubicación del sistema operativo.

Revisión del Hardware

Para poder abstraer y agregar funciones al hardware, el SO lo debe conocer completamente.

Procesadores

El SO debe conocer todos sus registros. Además tiene muchos invisibles incluso para el SO. Los CPUs modernos tienen modo kernel y usuario.

Memoria

Idealmente debe ser más rápida que la velocidad de ejecución del CPU pero esto es imposible en la práctica. Por lo que separa en bancos de memoria según su velocidad y donde se encuentra.

Tiempo de acceso típico	Capacidad típica	Descripción
1 nseg	<1 KB	Los cpu modernos tienen varios tipos de caches cada vez más “alejados” en velocidad y físicamente de los núcleos de ejecución de instrucciones del CPU. L1, L2, L3.. etc.
2 nseg	4 MB	Registros y caches: dentro de cpu
10 nseg	512-2048 MB	Memoria Principal: es la RAM, fuera del cpu.
10 mseg	200-1000 GB	
100 seg	400-800 GB	

La memoria virtual posibilita ejecutar programas más grandes que la memoria RAM física disponible al usar el HDD como ram guardando las páginas de la paginación. Para controlar esto se usa la MMU.

Dispositivos de E/S

Cada dispositivo de E/S externo contiene un controlador. Se requiere un soft o DRIVER para controlar y comunicarse con cada controlador de dispositivo externo. El driver se coloca en el SO para que pueda ejecutarse en modo kernel.

Todos los registros del dispositivo forman el **espacio de puertos de E/S**. Operaciones:

1. Driver inicia una E/S y espera continuamente por la RTA: espera ACTIVA u OCUPADA porque bloquea al CPU.
2. Driver inicia una E/S y activa una interrupción cuando está listo: no bloquea al CPU.
3. DMA: es un chip que controla el flujo de bits entre la memoria y el periférico sin intervención del CPU, cuando termina avisa al SO mediante una interrupción.

Conceptos Generales

Procesos

Más adelante.

Espacio de Direcciones

Es la abstracción de la memoria física y es la que usa el programador. De cómo se traslada la memoria lógica a la física se encarga el SO.

Archivos

En linux todo es un archivo. **[un clásico]**

Protección

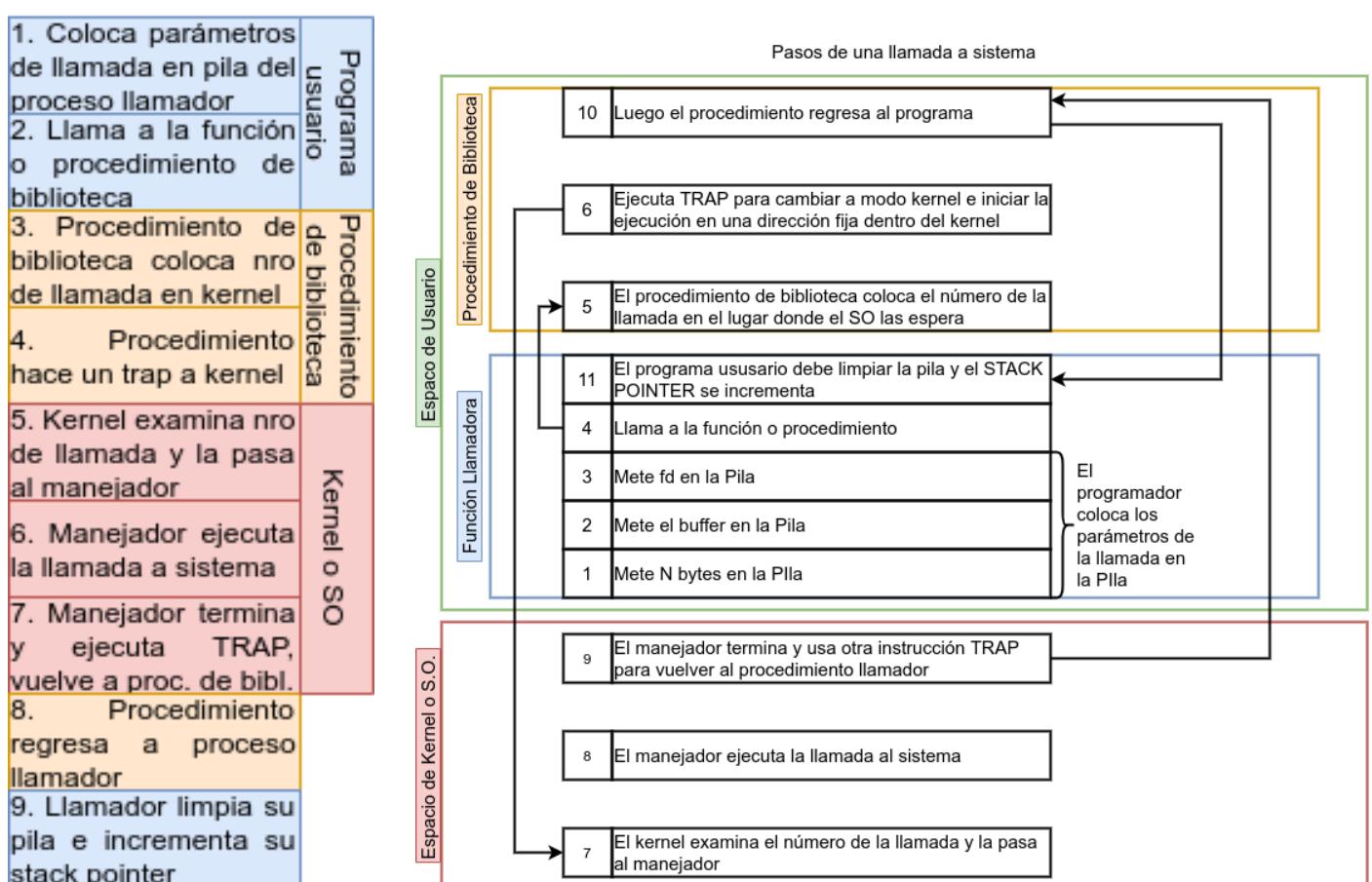
Es responsabilidad del SO. En unix cada archivo tiene un código de protección binario de 9 bits, esos 9 se dividen en grupos de 3 bits para los grupos del SO. [SO, Propietario, Otros usuarios]. Se conocen como bits rwx.

El Shell

Es el intérprete de comandos. El usuario interactúa directamente con el SO por ahí.

Llamadas a Sistema

Son la interfaz entre los programas y el usuario con el SO. El ejemplo en la imagen es para una función `read(fd,buffer, nbytes)`; por lo que queda graficado cuando el programador mete esos parámetros en la pila.



Siempre se deben verificar los resultados de una llamada a sistema. Típicamente en linux -1 es error y un número > 0 indica la cantidad de bytes o bits o lo que sea y es una señal de que no dio error pero deberías verificar la rta con tu código por si no tiene lógica.

Llamadas para Administración de Procesos FORK

Llamada FORK, única manera de crear un proceso en POSIX. Crea un duplicado exacto del proceso original. Con las mismas variables y todo. Devuelve un valor. El proceso de copia se llama “HIJO” y el original “Padre”.

<p>Llamada:</p> <pre>pid = fork(); pid = waitpid(pid, &stalock,...); exit(status);</pre>	<p>Espera a que el hijo termine Termina la ejecución de un proceso con estado</p>
--	---

Descripción:

Crea un hijo. PID=0 en hijo, Pid del hijo en padre.

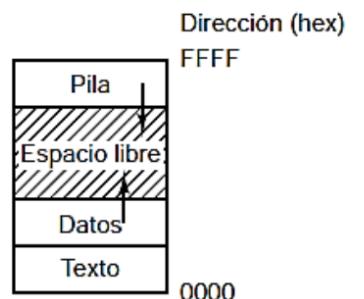
Memoria de un proceso se divide en 3 segmentos:

-Texto o código

-Datos o variables (heap): crece hacia arriba (en espacio de direcciones lineales)

-Pila: crece hacia abajo (de la máxima dirección posible a la dirección cero)

El espacio libre se usa para que los segmentos de pila y datos puedan crecer dinámicamente durante la ejecución del programa.



Llamadas para Administración de Archivos

Antes de leer o escribir un archivo se debe abrir (o crear y abrir). Esto se hace con **OPEN** y unos parámetros: O_RDONLY, O_WRONLY, O_RDWR y O_CREAT (que crea el archivo si no existe cuando se llama OPEN). La función OPEN devuelve un descriptor de archivo con el que se trabaja dentro del código.

<u>Llamada</u>	<u>Descripción</u>
<code>fd = open(archivo, como, ...)</code>	Abre archivo para lectura, escritura o ambos.
<code>s = close fd</code>	Cierra un archivo abierto (NO LO ELIMINA).
<code>n = read(fd, buffer, nbytes)</code>	Lee datos de un archivo y coloca en buffer.
<code>n = write(fd, buffer, nbytes)</code>	
<code>posicion = lseek(fd, desplazamiento, de donde)</code>	Desplaza en apuntador de archivo.
<code>s = stat(nombre, &buff)</code>	Obtiene información de estado de un archivo .

Estructuras de un SO

- Estructura monolítica: No tienen estructura. Tienen una pequeña jerarquía de procedimiento principal (invoca call systems), procedimientos de servicio (call system) y procedimientos utilitarios (ayudan a los de servicio).
- Estructura de capas: Se divide en 6. 0 “Asignación del procesador y multiprogramación”, 1 “Administración de la memoria”, 2 “Comunicación operador-proceso”, 3 “Control de entrada/salida”, 4 “Programas de usuario” y 5 “Proceso operador del sistema”.
- Estructura Microkernel: Enfocado a funciones centrales, como: de control, planifica hilos y no procesos, manipula interrupciones y excepciones, recupera fallas. La interfaz la controla el SHELL.
- Estructura de máquinas virtuales: Aparenta que cada terminal posee su máquina real y propia, pero se basa en multiprogramación, los SO de cada máquina corren sobre el SGMV sistema generador de máquinas virtuales que multiprograma las máquinas virtuales.

Procesos

Son abstracciones de los SO que permiten trabajar de manera pseudo concurrentemente. Convierten un CPU único en varios virtuales y ejecutan varias tareas “en paralelo” [mentira, pero es tan rápido que así parece].

PROGRAMA: COLECCIÓN DE PROCESOS DISTINTOS. EL PROGRAMA CONTIENE LOS PASOS PARA EJECUTAR UN PROCESO
PROCESO: ACTIVIDAD QUE TIENE UN PROGRAMA. CONTIENE ENTRADA, SALIDA Y ESTADO

Modelo

El soft se organiza en procesos secuenciales, que son una instancia de un programa en ejecución. Incluyen el valor del CP (contador de programa), registros y variables de ese programa. Cada proceso tiene una “cpu virtual”, lo que realmente pasa es que el CPU comuta el proceso en ejecución rápidamente dando la sensación de paralelismo y de múltiples procesadores.

La comutación rápida entre procesos se conoce como **MULTIPROGRAMACIÓN**. Existe un solo CP único físico, en el cual se va cargando el CP lógico de cada proceso cuando se lo ejecuta.

Creación

Los eventos que crean procesos son:

- 1) Arranque del sistema
- 2) Llamada a sistema
- 3) Usuario
- 4) Trabajo por lotes

En todos los casos, un proceso existente llama al sistema. En UNIX, **fork()** crea un clon del proceso llamador y mediante la función **execve** se independiza.

Terminación

- 1) Voluntaria: cuando el proceso hace **exit(0)**;
- 2) Voluntaria por error: **exit(-1)**;
- 3) Involuntaria: error fatal, el cpu o SO la terminan
- 4) Involuntaria: terminada por otro proceso

Jerarquías

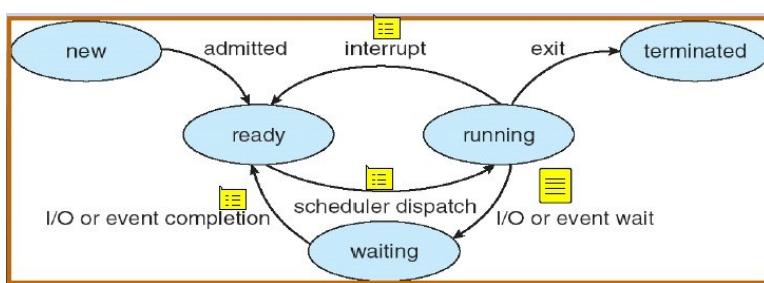
Cada proceso y sus hijos forman un grupo de procesos. Todos los procesos del sistema pertenecen a un solo árbol con INIT en la raíz. A pesar de esto, cada proceso es un ente independiente con su propio CP y estado interno. Además los procesos necesitan interactuar entre sí.

Estados

- En ejecución: cuando usan el CPU
- Listo: espera a ser ejecutado
- Bloqueado: no puede ejecutarse hasta que ocurra algo

Transiciones entre estados

- 1) Cuando un proceso no puede continuar (por la razón que sea, siempre que no sea un error), UNIX lo bloquea hasta que ocurra lo necesario para continuar.
- 2) Planificador: para ejecutar otro proceso.
- 3) Planificador: para continuar la ejecución del que pausó.
- 4) De bloqueado a listo cuando aparecen los datos necesarios.



ACLARACIONES:

Si el proceso hace un call al kernel, este lo coloca en estado bloqueado y llama al planificador para ejecutar otro proceso mientras que el original espera a que termine la llamada que hizo. El nivel más bajo del SO es el planificador, maneja las interrupciones, inicia y detiene procesos. El planificador es una ISR, se ejecuta cada X milisegundos. Hace un taskgate y directamente termina para continuar ejecutando un proceso.

Implementación

Para poder cambiar el proceso en ejecución el SO (más específico el planificador) mantiene una "tabla de procesos" que es un arreglo de estructuras. La estructura contiene, por cada proceso, CP, SP, asignación de memoria y sus archivos. Cuando se genera una interrupción por E/S:

1. El hardware mete el contador del programa a la pila, etc.
2. El hardware carga el nuevo contador de programa del vector de interrupciones.
3. Procedimiento en lenguaje ensamblador guarda los registros.
4. Procedimiento en lenguaje ensamblador establece la nueva pila.
5. El servicio de interrupciones de C se ejecuta (por lo general lee y guarda la entrada en el búfer).
6. El planificador decide qué proceso se va a ejecutar a continuación.
7. Procedimiento en C regresa al código de ensamblador.
8. Procedimiento en lenguaje ensamblador inicia el nuevo proceso actual.

Hilos

Son subprocessos que comparten un espacio en memoria y se ejecutan casi en paralelo. Se usan en aplicaciones que realizan varias tareas a la vez y que se pueden bloquear mutuamente. **UNIDAD BÁSICA DE USO DE CPU, LOS HILOS A LOS PROCESOS, SON LO QUE LOS PROCESOS SON AL CPU**

Ventajas

- Comparten espacios de direcciones y sus datos
- Más rápidos y livianos que crear procesos
- Útiles cuando el proceso hace muchas E/S
- Posibilitan procesos secuenciales que realizan llamadas a sistema con bloqueo y de todas las llamadas forman un paralelismo.

Cada hilo tiene su propio CP, variables locales y pila con el historial de ejecución (dirección de retorno y variables locales). Los hilos comparten variables globales, archivos abiertos, etc, todo lo que tenga el proceso padre y sus hilos. Se pueden romper mutuamente porque no hay protección en lo que hace cada hilo con la memoria que comparten todos. Hilos en POSIX:

Llamada de hilo	Descripción
Pthread_create	Crea un nuevo hilo
Pthread_exit	Termina el hilo llamador
Pthread_join	Espera a que un hilo específico termine
Pthread_yield	Libera la CPU para dejar que otro hilo se ejecute
Pthread_attr_init	Crea e inicializa la estructura de atributos de un hilo
Pthread_attr_destroy	Elimina la estructura de atributos de un hilo

Especificaciones varias

Los hilos se almacenan en tablas (como si fueran procesos), y estas tablas están dependiendo donde se implementen. Los hilos atraviesan estados, al igual que los procesos (son los mismos estados). El multihilamiento es muy útil mientras se tienen varias CPUs. Elementos de procesos e hilos:

Elementos por proceso	Elementos por hilo
Espacio de direcciones	Contador de programa
Variables globales	Registros
Archivos abiertos	Pila
Procesos hijos	Estado
Alarmas pendientes	
Señales y manejadores de señales	
Información contable	

Figura 2-12. La primera columna lista algunos elementos compartidos por todos los hilos en un proceso; la segunda, algunos elementos que son privados para cada hilo.

Hilos en espacio de Usuario

El kernel no sabe que existen. El proceso principal hace de planificador para sus propios hilos, por lo que requiere su propia tabla de hilos y se implementa con una biblioteca. Si se bloquea un hilo se bloquean todos. Conmutar el hilo es más veloz que con TRAP al kernel pq es interno al proceso. En realidad el cpu nunca cambia de proceso. Si un hilo produce un fallo, el SO mata al proceso entero. Los hilos deben renunciar voluntariamente el control del cpu para que otro hilo dentro del mismo proceso pueda ejecutarse.

Hilos en espacio de Kernel

El kernel sabe de los hilos de cada proceso y los administra. Cuando un hilo se bloquea el kernel sabe que puede cambiar de proceso o ejecutar otro hilo del mismo proceso. Desventaja del tiempo que demora para hacer el cambio de hilo, ya que usa la función TRAP. Es el sistema que se usa en linux.

Relación entre las implementaciones

- M:1 es M hilos implementados en usuario y 1 en kernel
- 1:1 es que el kernel conoce todos los hilos (muy costoso)
- M:N es un mix entre los dos anteriores (un poquito de esto... un poquito de aquello...) se debe especificar cuantos hilos se utilizaran.
- Modelo de 2 Niveles es todo lo anterior a la vez, M:N y 1:1 simultáneamente (si... bueno... quién tiene hambre?)

Planificador [se viene el acv]

Es la parte del SO que decide qué proceso se ejecuta. Es necesario pq los procesos alternan ráfagas de cálculo con E/S de manera dinámica y se debe decidir rápidamente quién sigue para aprovechar el hardware. Se distinguen dos tipos de procesos:

- Limitados a cálculo: mucho uso de CPU, poco de E/S en comparación
- Limitados a E/S: poco tiempo de uso de CPU en comparación al de E/S. Tienen esperas frecuentes.

A medida que los CPUs se vuelven más rápidos los procesos se tornan más limitados por E/S.

Cuándo Planificar Procesos?

1. Cuando se crea un proceso. Elegir ejecutar padre o hijo
2. Proceso se bloquea esperando E/S
3. Cuando un proceso en ejecución termina
4. Por interrupción de E/S. Proceso bloqueado pasa a estar listo
5. Luego de haber ejecutado una llamada a sistema

Tipos de Planificación

- Apropiativa: el planificador puede interrumpir procesos, pasarlos de running a ready, etc. Requiere que exista una interrupción periódica en el cpu cuya ISR sea el planificador.
- No apropiativa: no interrumpe los procesos. El proceso en ejecución se debe bloquear o ceder el control del cpu voluntariamente.

Categorías de los Algoritmos de Planificación

- Lotes: se usan cuando no hay usuarios y se pueden usar algoritmos no apropiativos. Reduce la conmutación entre procesos y mejora el rendimiento. Porque los procesos se ejecutan uno detrás de otro hasta terminar.
- Interactivos: cuando hay usuarios. Son apropiativos y de propósito general
- De tiempo real: pueden ser o no apropiativos. Los procesos no duran mucho en entornos de tpo real.

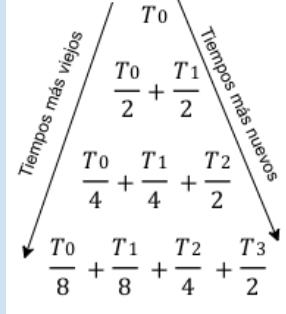
Metas de los Algoritmos (dependen del entorno)

Todos	Lotes	Interactivos	Tpo Real
<p><u>Equidad:</u> procesos comparables tienen tiempos similares.</p> <p><u>Aplicación de políticas:</u> considera los requerimientos del sistema que maneja.</p> <p><u>Balance:</u> mantener todas las partes del hardware ocupadas.</p>	<p><u>Rendimiento:</u> maximizar el throughput (tasa de transferencia).</p> <p><u>Tiempo de retorno:</u> minimizarlo.</p> <p><u>Uso de CPU:</u> mantenerla ocupada hasta terminar todo el lote.</p>	<p><u>Tiempo de Rta:</u> responder rápidamente a pedidos del usuario.</p> <p><u>Proporcionalidad:</u> cumplir con las expectativas del usuario. Tareas que el usuario percibe como cortas deberían demorar poco tiempo.</p>	<p><u>Cumplir plazos de tpo:</u> evitar la pérdida de datos.</p> <p><u>Predictibilidad:</u> evita la degradación de calidad del sistema. \Rightarrow <u>Determinístico</u></p>

Algoritmos de planificación por Lotes

1ro en entrar, 1ro en atenderse	El más corto primero	El de menor tiempo restante
No apropiativo. Asigna los procesos al cpu en el orden cronológico de aparición. Tiene una cola de listos. Los procesos se ejecutan hasta bloquearse o ceder el CPU.	No apropiativo. Requiere conocer todos los tiempos de ejecución al iniciar el lote. Cuando hay varios trabajos de igual importancia el planif elige el más corto.	Apropiativo. Requiere conocer todos los tiempos de ejecución al iniciar el lote. El planif elige el proceso con menor tiempo de ejecución restante. Los trabajos nuevos y cortos reciben mejor atención.
Problema: los procesos limitados a E/S demoran demasiado en terminar.	Problema: ¿Cómo sabe el tiempo de ejecución? Ejecutando una vez todos los procesos y midiendo. Solo es óptimo cuando todos los procesos están listos a la vez.	Problema: los procesos largos, nuevos o viejos, pueden no recibir atención del cpu. Fenómeno llamado inanición de procesos largos.

Algoritmos de Planificación Interactiva

Turno Circular (round robin)	Por prioridad	Proceso más corto siguiente
A cada proceso se le asigna un tiempo “Quantum”, si sigue en ejecución cuando se termina su turno el planif lo pasa a listo. Planif debe mantener dos listas: -Bloqueados y -Listos Todos los procesos son de igual prioridad.	Toma en cuenta factores externos. Para evitar que los procesos de más prioridad se ejecuten por siempre el planif puede bajar la prioridad de cada proceso cuando lo ejecuta con cada pulso de reloj. Prioridades se pueden asignar estáticamente o dinámicas. Se agrupan procesos de = prioridad y se usan otros algoritmos para cambiar entre niveles de prioridad y procesos dentro de cada nivel.	Trata de maximizar el throughput. Generalmente los procesos interactivos esperan comandos. Se puede minimizar el tiempo de rta total ejecutando el más corto primero. 
Problema: la conmutación entre procesos demora un tiempo, aprox 1mS. Por lo que el tiempo que dura el “Quantum” es una relación costo-beneficio. Si el Q es muy corto el SO usa mucho CPU, si es muy largo el sistema no es tan interactivo. Quantum típico de 20 a 50mS.	Problema: los procesos de menor prioridad puede que nunca tengan CPU o se ejecuten muy lentamente si son muy largos (de muchos quants).	Problema: debe saber cual es el más corto que sigue. Requiere una lista en RAM a la que debe acceder, lo hace más lento. Mide la duración haciendo una suma ponderada de los tiempos de ejecución del mismo proceso. Se conoce como técnica de “envejecimiento”.
Planificación Garantizada	Planificación por Sorteo	Por partes equitativas
Habiendo N procesos, cada uno obtiene $1/N$ ciclos de CPU. Sistema debe llevar la cuenta de cuantos ciclos tiene cada proceso individualmente. Problema: requiere saber cuanto duran todos los procesos. Surgen problemas cuando se crean o cierran procesos.	Ejecuta los procesos en forma aleatoria. Dando una etiqueta de probabilidad que se incrementa con cada “sorteo” que pierden los procesos hasta que “ganan”.	Equitativo en función de usuarios y no de tareas. A cada usuario se le asigna cierta fracción de CPU y recursos que se distribuye entre los procesos de cada usuario.

Planificación de Hilos

Existen dos niveles de paralelismo. De procesos y de hilos.

Hilos Nivel Usuario

Se ejecutan dentro del quantum del proceso, este debe elegir qué hilo ejecutar. Planifica por round-robin y por prioridad.

Problema: no posee interrupción o reloj para cambiar de hilo que se ejecuta y se puede bloquear todo el proceso con sus hilos.

Hilos nivel Kernel

Los controla el SO. Requieren conmutación de contexto completa, lo que lleva mucho tiempo.

Ventajas: si un hilo se bloquea no detiene todo el proceso

Desventaja: el proceso o programador no puede decidir el orden de ejecución de los hilos de su propio proceso, eso ahora lo hace el planificador del SO.

Planificación en Sistemas de Tiempo Real

Estos son sistemas donde la respuesta ante estímulos debe suceder en un tiempo limitado. No importa si es lento o rápido, se debe cumplir. Se dividen en dos categorías:

- Tiempo real duro: los tiempos límites se deben cumplir obligatoriamente.
- Tiempo real blando: no conviene fallar, pero es tolerable dentro de un margen.

Esta clasificación no garantiza ni mierda, eso depende de que tan manco sea el ingeniero que lo implementa.

El comportamiento en tiempo real se hace dividiendo el programa en procesos. El planificador en este caso conoce cuánto dura cada proceso. Los eventos externos pueden ser periódicos o aleatorios. si hay "m" eventos periódicos y el evento "i" ocurre con periodo "Pi", se requieren "Ci" segundos para responder a ese periodo "i", entonces la carga total de tareas del sistema solo se puede manejar cuando:

$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$ Si la carga es > 1 el sistema como está, no podrá responder correctamente. Se deberá cambiar el CPU o dividir el sistema en 2.

Es planificable si y sólo si la sumatoria de todos los tiempos, divididos en función de cada periodo de aparición es menor o igual que 1.

IPC: Comunicación entre procesos

Compartir información entre procesos relacionados o no relacionados, acelera cálculos y admite modularidad.

Clasificación

Paso de mensajes:

Usan llamadas a kernel, sencillas de implementar y usar.

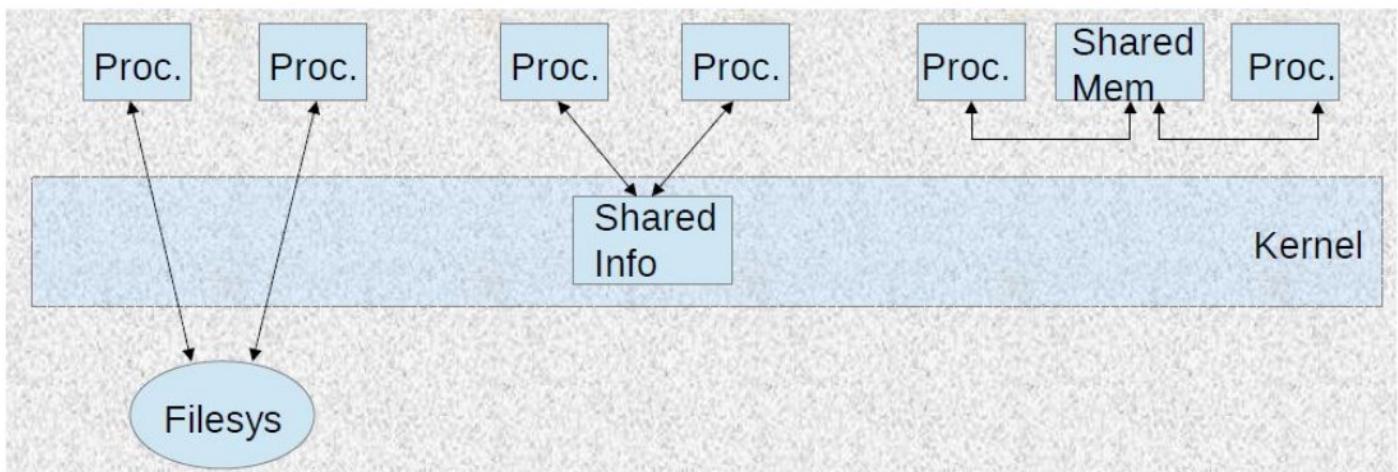
- Nivel Kernel: PIPE, FIFO y Cola de Mensajes. Son tipo Byte-Stream (flujo de bytes)
- RPC: llamada a procedimiento remoto
- MPI: interfaz de paso de mensajes
- Sockets: se ven en redes

Memoria Compartida:

Comparte un espacio de memoria, no pasa por kernel o SO, por lo que es más rápida. Debe usar sincronización para no hacer pija todo.

- Mapeo de archivos Anónimos
- Objetos de memoria
- Archivos de memoria [En 2020 no lo dieron, así que buena suerte.]

Caminos que se pueden tomar para compartir información



- 1) Por sistema de archivos: cada proceso pasa por kernel mediante read y write. Requiere sincronización.
- 2) Kernel: la info está en un buffer manejado por el SO. PIPES, FIFOs y [por la cola te di, digo] cola de mensajes.
- 3) Memoria compartida: requiere sincronización pero no pasa por kernel. el más rápido.

Persistencia de los mensajes en IPC

Cuando se pierde el mensaje.

Persistencia de Proceso

Se mantienen hasta que todos los procesos que usaban el método lo dejan de usar (cierran). Son:

- PIPE
- FIFO: se borran los datos pero el archivo queda y sobrevive reinicios
- Socket
- RPC
- Memoria compartida

Persistencia de Kernel

Se borran cuando se reinicia la pc:

- Cola de mensajes
- Semáforos
- Memoria compartida: objetos de memoria

Persistencia de Sistema de Archivos

Se tienen que eliminar explícitamente, son un archivo que se banca los reinicios...[porque es un archivo... esperen.... no era que todo es un archivo en linux?..... *se muere]

- Memoria compartida: archivos de memoria
- FIFO [de nuevo]
- Semáforos [que onda estos?.... shhh.... mejor no preguntar]

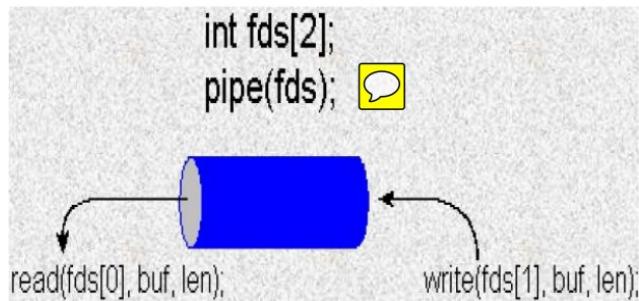
¿Cuándo uso cada una de estas vergas?

Entre procesos del mismo PC: PIPE, FIFO, MQ(Message Queue)[+olala senior ingles - y como se dice?+Cola de mensajes], Memoria compartida

Misma PC o entre PC's por red: Sockets, RPC, MPI.

Pijas, digo... PIPES

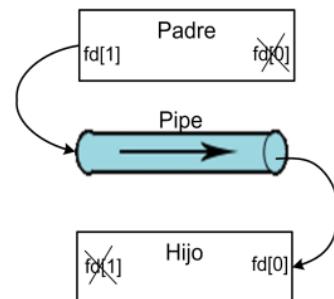
También llamadas **TUBERÍAS SIN NOMBRE**. Son una FIFO orientadas a flujos de bytes. Tamaño máximo de 8KB. Como no tienen nombre solo comunican procesos padre e hijo. Pueden ser bloqueantes o no bloqueantes.



Cuando se llama a **pipe(fds)**; el SO devuelve dos descriptores, uno en cada ubicación de ese arreglo **fds[2]**. Un selector escribe y el otro lee de la tubería. Si el buffer se llena, el proceso escritor se bloquea hasta que el otro lea y saque mensajes de la tubería. Cuando el buffer está [rancio] vacío y no hay nadie conectado al extremo de escritura el SO devuelve cero en las lecturas. **EL FLUJO ES UNIDIRECCIONAL**.

Crear tuberías:

1	int fds[2]:
2	pipe(fds);
3	pid = fork();
4	Padre cierra STDOUT que es fds[0]-> close (fds[0]);
5	Hijo cierra STDIN que es fds[1]-> close (fds[1]);



Para comunicación bi-direccional se hacen dos pipes distintas y se cierran cruzados los descriptores.

FIFO (*pijas con nombre*)

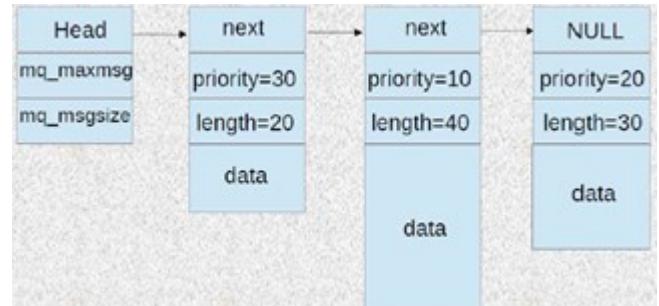
Se puede compartir información entre procesos NO relacionados, es unidireccional, tiene una ruta de acceso al sistema de archivos, está orientado a “Stream” y se puede abrir como bloqueante o NO bloqueante. Persiste con un nombre en el sistema de archivos. Se puede abrir como SOLO escritura o SOLO lectura (ya que es unidireccional). Se utilizan llamadas a sistema para crearlas. Para eliminar el nombre se debe hacer manualmente. Se crea con mkfifo (nombre, modo);, se abre con open(); y se elimina con unlink();

Problemas Stream de Datos

Los Pipe y los FIFO son IPC de “Stream” de datos, por lo que no se identifican bloques de información, sino un flujo de bytes. No discrimina quién escribió los datos y cuando hay varios lectores, si uno saca el mensaje, el otro no puede leerlo ya que se destruyó. Se puede solucionar limitando los datos “in-band” (carácter especial) limitando una nueva línea, explicitando el largo, definiendo un largo fijo o (solo en sockets) un registro por conexión, cerrando la conexión con los pares para indicar el final del registro.

Cola de Mensajes [no es la cola que estás pensando]

Cada mensaje es un registro con prioridad, envía mensajes segmentados y comparte datos entre procesos sin igual tiempo de vida. Las prioridades permiten que el receptor los reciba de acuerdo a la importancia, siempre teniendo en cuenta el más antiguo. Pueden ser abiertos como bloqueantes y NO bloqueantes. Tiene persistencia de kernel. Los atributos son: Prioridad, longitud y datos. Son como listas enlazadas. No se puede identificar quién envía el mensaje.



- ✓ `mqd_t mq_open(const char * name,int oflag);`
- ✓ `int mq_close(mqd_t mqdes);`
- ✓ `int mq_send(mqd_t mqdes,const char msg_ptr,size_t msg_len,unsigned msg_prio);`
- ✓ `ssize_t mq_receive(mqd_t mqdes,char * msg_ptr,size_t msg_len,unsigned * msg_prio);`

La cola se elimina con `mq_unlink()`; Igualmente mantiene el problema de que no identifica quién envía cada mensaje para los distintos receptores.

El sistema también le puede notificar el proceso enviador cuando está queriendo colocar un mensaje en una cola vacía y sin receptores.

Resumen de Códigos en C

PROCESOS

```
#include <sys/types.h>           //biblioteca en la que definido el tipo pid_t
#include <unistd.h>              //biblioteca en la que definida fork
#include <sys/wait.h>             //donde se define wait
```

FUNCION

`pid_t fork(void);`

fork devuelve:

- Al padre el ID de proceso hijo creado
- Al hijo 0
- -1 en caso de error

Si hacemos varios forks sin distinguir si es hijo o padre, la cantidad de final de procesos es $2^{\text{nro de forks}}$

Para ver los padres e hijos dentro de linux usamos PSTREE

PID

```
pid_t getpid(void);           // getpid() nos devuelve el pid del proceso actual
pid_t getppid(void);          // getppid() nos devuelve el pid del proceso padre (en huerto es =1)
```

EJEMPLO: `printf ("soy el pid: %d y mi papa es: %d\n",getpid(),getppid());`

TERMINAR

```
void exit(int status); //devuelve el estado en el que finalizó al padre si realiza un wait
```

ESPERAR

```
pid_t wait(int *status); //Devuelve el ID del proceso hijo terminado o -1 en caso de error
```

CAMBIAR PROGRAMA

```
execve(pathname, argv, envp) // carga un nuevo programa (nombre de ruta, con la lista de argumentos argv y lista de entornos envp) en la memoria de un proceso.
```

```
int execl(const char *pathname, const char *arg, .../* , (char *) NULL */); // No devuelve nada en caso de éxito y -1 en caso de error
```

EJEMPLO: execl("/bin/ls", "ls", "-l", NULL);

SEÑALES (kill -l)

```
#include <signal.h>
```

COMANDOS: kill –SIGXXX PID

A. Señales tratadas por defecto (SIGN_DFL)

B. Señales ignoradas (SIG_IGN)

C. Manejador

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

FUNCIÓN SIGNAL

```
void (*signal(int sig, void (*handler)(int))) (int); // sig: señal a manejar, *handler dirección de la función manejador
```

EJEMPLO: signal (SIGINT, manejador) //En vez de manejador puede ir SIGN_DFL o SIG_IGN

FUNCION MANEJADOR

```
void controlador (int); //int con el número de señal.
```

```
signal (sig, &función) //recibe una señal y define la función que la atiende
```

FUNCION DE IPC

```
int kill(pid_t pid, int sig); // Enviar una señal a un proceso. Pid: identificador del proceso, Sig: tipo de señal a enviar. Devuelve 0 si tuvo éxito, o -1 cuando hay error
```

pid > 0 señal enviada al proceso con ese pid>0

pid = 0, Se envía la señal a todos los procesos del mismo grupo que el proceso emisor

pid < -1, la señal se envía a todos los procesos en el grupo de procesos cuyo ID es igual al valor absoluto de pid.

pid = 1 la señal se envía a todos los procesos a los cuales tiene permiso de enviar (todos menos init y él mismo)

TIPOS

SIGUSR1 y SIGUSR2: Para usuarios

SIGKILL: Termina la ejecución, NO PUEDE SER IGNORADA. Es un comando de terminal que manda una señal especificada.

SIGSTOP: Detiene la ejecución, NO PUEDE SER IGNORADA en linux es CTRL+Z

HILOS

```
#include <pthread.h>
```

CREAR HILO

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void *), void *arg);
```

```
//rc = pthread_create(puntero_hilo, atributo, funcion, argumento);
```

El nuevo hilo de ejecución se inicia llamando a la función identificada por start con el argumento arg. Si los atributos son definidos como NULL, tomará los atributos por defecto. La función start retorna un valor.

“attr” es de tipo pthread_attr_t, se puede utilizar para especificar los atributos utilizados en la creación de un nuevo hilo. Incluye: localización y tamaño de la pila del hilo, prioridad de los hilos y si el hilo es unible (join) o no (PTHREAD_CREATE_JOINABLE : el hilo es “unible” y PTHREAD_CREATE_DETACHED: el hilo no es “unible”).

Devuelve 0 si tuvo éxito o un error positivo en caso de error.

Se define un argumento tipo const cuando se desea proteger a esa variable de posibles modificaciones. Que una variable sea const significa que es de solo lectura, nos aseguramos de que no se pueda modificar. attr no es definida en un principio como const cuando se la declara, sino dentro de la función.

El tercer argumento de pthread_create() indica que:

1. Es un puntero a una función start, void *(*start)(void *).
2. Su argumento de entrada es un puntero a void, void *(*start)(void *).
3. Su argumento de salida es un puntero a void, **void *(*start)(void *)**.

Que un argumento de entrada sea definido como void implica que en realidad podrá ser cualquier tipo de dato cuando se implemente la función que será asignada al hilo. Sí será **necesario** castear a puntero void esta variable arg.

LIBERACIÓN DE HILO

```
int pthread_detach(pthread_t thread); //el sistema limpia y remueve el hilo automáticamente cuando este termina, no se lo puede esperar con un join. Devuelve 0 si tiene éxito, o un error positivo en caso de error.
```

TERMINAR HILO

```
void pthread_exit(void *retval); // Termina solo ese hilo.
```

```
exit(); //es como si hiciera return(); me termina todo. retval especifica el valor de retorno para el hilo
```

```
int pthread_cancel(pthread_t thread); // Se cancela el hilo
```

IDENTIFICACIÓN DEL HILO

```
pthread_t pthread_self(void); // Devuelve el TID
```

UNIÓN DE UN HILO

```
int pthread_join(pthread_t thread, void **retval); // Espera que los demás hilos terminen.
```

Devuelve 0 si tuvo éxito o un número positivo en caso de error. Si retval es un puntero no nulo, recibe una copia del valor de retorno del hilo terminado (el valor que se especificó cuando el hilo realizó un retorno o llamado pthread_exit()). Es parecido al waitpid(); pero no hay jerarquía, cualquiera puede unirse con cualquiera.

```
char * itoa( int valor, char * cadena_destino, int base )
```

Sincronización

La sincronización se refiere a la relación entre 2 o más eventos (hilos, procesos, etc): antes, durante y luego de su ejecución.

Modelos de Ejecución

- MonoCPU - MonoTarea: no trae muchos problemas, ya que solo hace una cosa a la vez y se conoce el orden de los eventos.
- MonoCPU - Multitarea: dependen del planificador. No son deterministicos porque su salida depende del flujo de ejecución de los procesos y no siempre se ejecutarán igual.
- MultiCPU - Multitarea: un quilombo, mejor me cambio de carrera.

Definiciones

Concurrencia

Implica que dos o más procesos / hilos se pueden ejecutar en paralelo o pseudo paralelo. El objetivo es más velocidad de ejecución y aprovechar los tiempos ociosos del cpu. Características:

- No tienen restricciones de sincronización
- No modifican recursos compartidos
- Viendo el código no se puede saber la salida pq depende del orden de ejecución
- El resultado debe ser siempre el mismo.

Determinístico

Cuando la salida del proceso es predecible. Con hilos, los procesos se vuelven no determinísticos.

Los hilos y procesos son concurrentes no determinísticos. Cuando comparten recursos y el resultado puede variar entre ejecuciones, pq cambio el orden de ejecución, se deben poner restricciones de software.

Los problemas de concurrencia se dan cuando comparto variables entre procesos o hilos en un mismo espacio de memoria.

Estrategias de Sincronización

<u>Serialización</u>	<u>Punto de encuentro:</u>	<u>Exclusión mutua:</u>	<u>Barrera</u>	<u>Condición de Competencia o Carrera:</u>
Cuando un evento debe esperar a que otro termine y usar recursos modificados por el otro. Típicamente una porción de un programa es serializada. Se aplica para funciones dentro de un programa. EJ: ingresos por teclado.	Cuando eventos se esperan mutuamente y recién ambos llegan al mismo punto de ejecución el programa sigue se ejecución. Generalmente usan recursos modificados mutuamente.	Cuando dos eventos no deben producirse a la vez y ambos modifican los mismos recursos compartidos. El resultado se hace dependiente del orden de ejecución.	Punto de encuentro para más de dos procesos/hilos.	Permite dar acceso simultáneo a datos. El resultado depende del orden de ejecución. La zona de memoria con los datos se denomina "zona crítica". Se da porque el kernel es apropiado. La posible solución son las "variables cerrojo" pero tienen problemas de atomicidad.

Herramientas de Sincronización

Herramientas pedorras:

- Variables cerrojo: tienen problemas de atomicidad
- Desactivar interrupciones
- Instrucción TSL: bloquea al bus de datos mientras se lee o escribe en la memoria

Semáforos

Son construcciones de software (variables ≥ 0) que usan, internamente, instrucciones TSL. Sus incrementos y decrementos son atómicos por usar esas instrucciones. Si al decrementar el semáforo se vuelve negativo, el hilo o proceso que lo decremente se bloquea hasta que otro incremente el semáforo.

Los semáforos son llamadas a sistema y las instrucciones TSL son privilegiadas.

- Semáforos sin nombre: residen en un lugar acordado de memoria. Se comparte entre procesos e hilos. Para compartir entre procesos se debe encontrar en un espacio de memoria compartida. Tiene persistencia de memoria compartida. Cuando se cierran los proc que usan ese espacio se elimina.
- Semáforos con nombre: para compartir entre procesos no relacionados. Pueden acceder al mismo usando: `sem_open(NombreDelSemáforo);`. Lo maneja el kernel. Nombre está predefinido.

Cuando se comparte un semáforo sin nombre entre hilos/procesos en una región de memoria compartida, y el padre hace un fork, el hijo hereda el semáforo.

Operaciones

- wait: resta 1 al semáforo, si da menor que cero bloquea el proceso.
- post: suma 1 al semáforo
- init: inicializa el semáforo y se le puede dar un valor inicial

Mutex

Son variables binarias. Siempre que se crean empiezan con valor 1. Si el mutex estaba en cero y un proc lo trata de disminuir, se bloquea hasta que mutex vale 1 de nuevo.

Al acceder a secciones críticas, la ejecución del acceso a memoria tiene que ser atómica. El mutex sincroniza el uso de variables compartidas entre hilos. Pasos:

1. Restar 1 al mutex, ahora queda en cero
2. Acceder al recurso compartido
3. Sumar 1 al mutex, ahora vale 1.

Inicialización

Usando `pthead_mutex_init(...);` se inicializa de manera estática con los parámetros por defecto.

Mutex dinámico

Se inician con una función que recibe un puntero a un descriptor de mutex y otro puntero a una estructura con los atributos.

Mutex recursivo

Cuando un hilo adquiere por primera vez un mutex, pone en 1 un contador, cada lock lo incrementa y cada unlock lo decrementa, el mutex se libera cuando el contador de lock llega a cero.

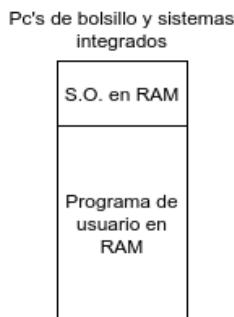
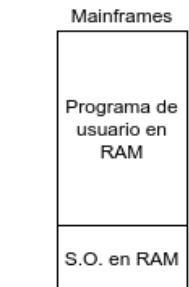
Interbloqueo:

Ocurre cuando dos mutex se bloquean mutuamente entre dos hilos distintos queriendo ingresar a una misma variable. Para evitarlo, los hilos deben estar (lock) y sumar (unlock) en el mismo orden. Como los mutex consumen ciclos de reloj pq son de kernel, hacen más lenta la ejecución.

Los mutex y semáforos son variables que debo inicializar mediante llamadas al sistema. Son únicos de cada proceso / espacio de memoria y no interfieren con otros semáforos del mismo nombre pero distinto espacio de memoria.

SIEMPRE QUE DISTINTOS HILOS O PROCESOS MODIFICAN UNA MISMA VARIABLE DEBEMOS REALIZAR EXCLUSIÓN MUTUA CON MUTEX O SEMÁFOROS.

Gestión de Memoria



Sin Abstracción

No se podían tener dos programas en memoria a la vez.

Un error en los programas podía dañar zonas de memoria con el SO y corromper el sistema.

Ejecución de múltiples programas sin abstracción

El SO debe guardar toda la memoria a un archivo en disco, para luego cargar y ejecutar otro programa desde el disco. A este proceso se le llamaba "Intercambio".

Adhiriendo un poco de hardware era posible la multiprogramación sin realizar intercambio. Se dividía la memoria en bloques con una llave de protección de 4 bits almacenada en registros especiales dentro del CPU.

Problema: TODOS los programas hacen referencia DIRECTA a la MEMORIA FÍSICA, por lo que pueden hacer pija todo.

Solución propuesta: Reubicación estática de memoria. Al cargar un programa en una dirección, se le suma un offset a la dirección base de todo el programa durante el proceso de carga en memoria. Con esto quedan todos los registros, instrucciones y variables apuntando correctamente dentro del espacio FÍSICO de memoria asignado.

ABSTRACCIÓN DE MEMORIA: ESPACIO DE DIRECCIONES [BONJOURRR MANGA DE SOQUETES]

Espacio de Direcciones

Es el conjunto de direcciones de memoria que puede usar un proceso para direccionar memoria. Cada proceso tiene su propio espacio independiente del resto.

Registros Base y Límite:

Son registros de hardware especiales.

Registro Base: guarda la dirección física donde empieza un programa

Registro Límite: se carga con la longitud total del programa.

Se usan para la reubicación dinámica de la memoria. La cual asocia el espacio de dirección de cada proceso con una parte distinta de la memoria física.

Desventaja: Cada vez que el programa referencia la memoria, se debe sumar el offset y comparar el límite (por protección), agrega retardo.

Intercambio

Se hace cuando la memoria RAM es limitada. Carga un proceso completo en memoria, se ejecuta, luego se guarda completo en disco para liberar el espacio en la RAM. Además, si se implementa memoria virtual, los programas se pueden ejecutar estando parcialmente en memoria.

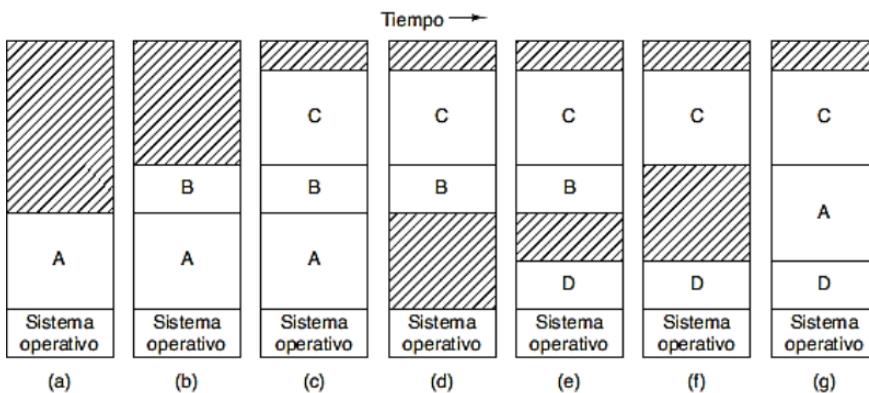


Figura 3-4. La asignación de la memoria cambia a medida que llegan procesos a la memoria y salen de ésta. Las regiones sombreadas son la memoria sin usar.

La memoria es variable porque los procesos se bloquean y al hacer los intercambios dejan espacios libres para otros nuevos.

Fragmentación Externa e interna:

Se produce cuando quedan espacios de memoria muy pequeños y no se pueden cargar procesos en estos o a un proceso el tamaño del espacio que se le asignó no le es suficiente. Esto se soluciona con:

Compactación de Memoria:

Para compactar se requiere conocer los tamaños y los tiempos de ejecución de los procesos. No hay reubicación dinámica de código. El kernel debe llevar un registro de la distribución de la memoria. Conviene asignar memoria adicional cuando se mueve un bloque de memoria porque puede crecer.

¿Cuánta memoria se asigna a proceso?

Si se asigna muy poca se puede producir fragmentación interna, si asigno mucha puedo estar ocupando espacio al pedo, se prefiere eso a que un proceso se quede sin memoria.

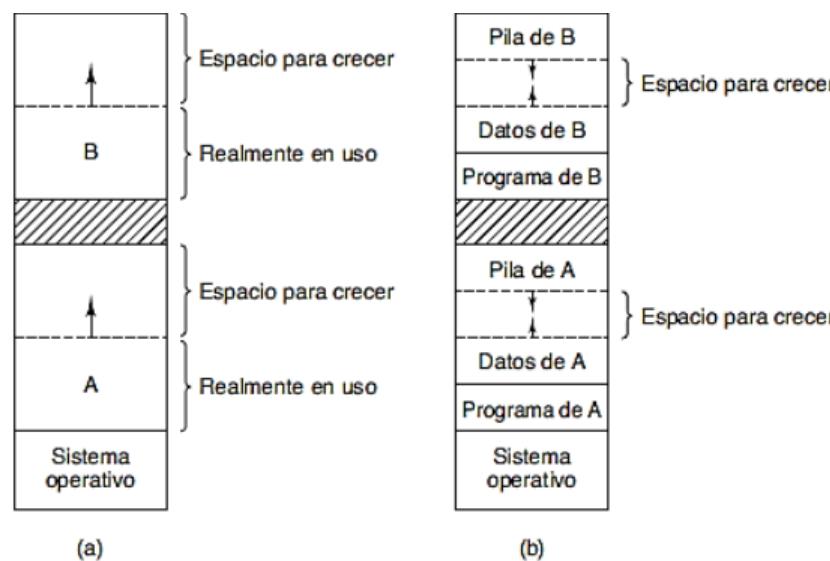


Figura 3-5. (a) Asignación de espacio para un segmento de datos en crecimiento. (b) Asignación de espacio para una pila en crecimiento y un segmento de datos en crecimiento.

Administración de Memoria Libre

Mediante Mapa de Bits

Se divide la memoria en bloques, cada

bloque se mapea con un bit. B = 0 => Bloque libre

B = 1+> Bloque ocupado

Una memoria de 32 bits necesitará n bts del mapa y este ocupa 1/33 % de la memoria.

Problemas:

-Cuánto representa cada bit? La mínima unidad de asignación.

-Proceso de búsqueda de posiciones libres consecutivas es lento.

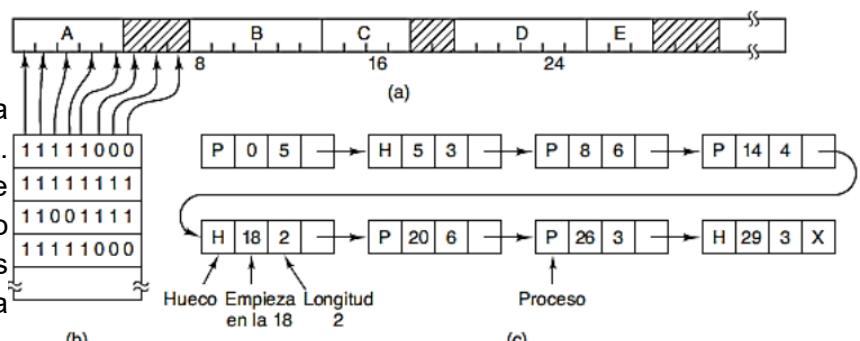


Figura 3-6. (a) Una parte de la memoria con cinco procesos y tres huecos. Las marcas de graduación muestran las unidades de asignación de memoria. Las regiones sombreadas (0 en el mapa de bits) están libres. (b) El mapa de bits correspondiente. (c) La misma información en forma de lista.

Mediante Listas Enlazadas

Cada entrada en la lista especifica si es un huevo o un proceso, la dirección de inicio, longitud y puntero al siguiente elemento de la lista. Puede mantener dos listas separadas de huecos y procesos.

Algoritmos de Asignación de Memoria por Lista Enlazada

Primer Ajuste	Siguiente Ajuste	Mejor Ajuste	Peor Ajuste	Ajuste Rápido [como tu vieja, ah re bardero]
Explora toda la lista hasta encontrar huecos consecutivos donde entre el proceso. Es el más rápido y simple y los huecos que genera son más grandes y útiles.	Guarda el último lugar que asignó y parte su búsqueda desde ahí.	Busca toda la lista y toma el hueco más pequeño posible para el tamaño del proceso. Produce excesiva fragmentación externa inutilizable.	Toma siempre el huevo más grande posible. Es una poronga. [Te fragmenta hasta los horarios de cursado]	Mantiene listas separadas para los tamaños estadísticamente más comunes de huecos.

Conclusiones

El mapa de bits es más rápido para actualizar pero lento para buscar. Las listas enlazadas son de búsqueda rápida y actualización lenta, pero puede mantener listas separadas para distintas cosas, por lo que se prefiere.

Todos los algoritmos deben realizar la fusión de espacios de memoria cuando se termina un proceso, si no se hace, la memoria se fragmenta rápidamente y no cabe nada adentro.

Memoria Virtual

Los programas se dividían en pequeñas partes conocidas como “sobrepuertos”. Al iniciar un programa solo se carga en memoria un “administrador de sobrepuertos” y éste se encarga de decidir qué porciones del programa se cargan en la memoria al inicio. Los sobrepuertos se mantienen en el disco y se llevan a memoria RAM por el método de intercambio. Esto se dejó de usar porque era el mismo programador quien debía dividir el programa y el SO solo hacía los intercambios de memoria dados por el programa en tiempo de ejecución. Se pasó esta tarea al mismo CPU y SO.

Paginación

En la paginación cada programa tiene su propio espacio de direcciones, dividido en trozos llamados páginas, cada página es un rango continuo de direcciones. Cuando el programa quiere hacer referencia a memoria que ya está en RAM, la asociación es instantánea, si no, el SO busca la página en disco y la coloca en RAM, si tiene que reemplazar alguna página pq no tiene espacio se usan algoritmos para decidir cuál dinamitar.

En paginación los procesos referencian **direcciones de memoria**, estas son virtuales y forman el **ESPACIO DE DIRECCIONES VIRTUALES**. Estas van a la MMU que las asocian las direcciones físicas reales. El espacio de direcciones virtuales se divide en unidades de tamaño fijo llamadas páginas. Las unidades correspondientes a las páginas en la memoria física se llaman marcos de páginas y son de igual tamaño. Las páginas son de 4096 direcciones y siempre empiezan en un múltiplo entero de 4096 y terminan en 4095, para dar lugar al inicio del siguiente marco de página.

Proceso de búsqueda de página

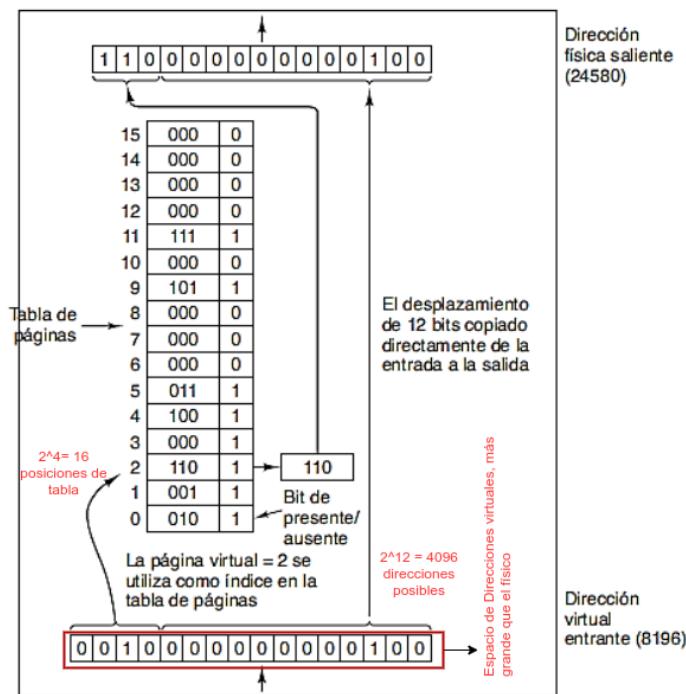
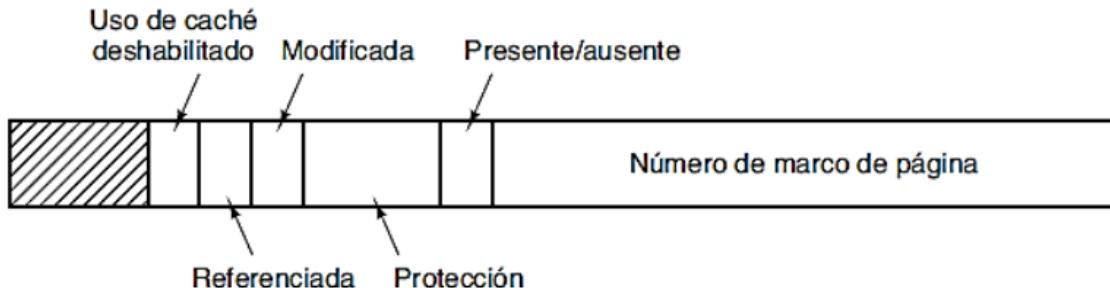


Figura 3-10. La operación Interna de la MMU con 16 páginas de 4 KB.

reemplaza por el correspondiente y vuelve a ejecutar TRAP para volver a buscar la página que ahora tiene $P=1$.

Pentium solo tiene 8 marcos de página por proceso en RAM. [por eso lo gorreaban]

Estructura de una entrada en la tabla de páginas



- Referenciada y modificada llevan el registro del uso de esa página, sirve para llevar un historial de uso de las páginas al momento de eliminarlas.
- Uso de caché deshabilitado: permite deshabilitar el uso del caché de esa página, útil cuando la página se asocia con dispositivos de E/S y no con memoria.

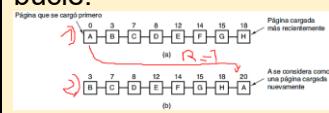
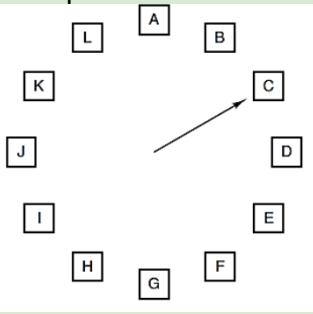
La dirección en disco de la página no está en la tabla de páginas, eso lo maneja el SO con sus propias tablas en RAM. Acá solo está lo físico para hacer el direccionamiento rápido por hardware.

- 1) Programa trata de acceder a una dirección, la envía a la MMU
- 2) En la MMU se usan los 4 bits más significativos como índice para buscar en la tabla de páginas de ese proceso en particular. Cada proceso tiene su tabla.
- 3) Si el bit de Presente P está en 1, la tabla está cargada en RAM.
- 4) MMU reemplaza los 4 MSB de la dirección virtual por los 3 bits de offset que contiene en su tabla.
- 5) La dirección física se obtiene con los 3 MSB que tiene la tabla y los 12 bits restantes de la dirección virtual.

Los números que tiene la MMU son los **MARCOS DE PÁGINAS DE ESE PROCESO**.

Si $P=0$ se hace un TRAP al sistema operativo que por "FALLO DE PÁGINA", en ese momento el SO elige un marco de página que no se use mucho y lo guarda en el disco. Luego lo reemplaza por el correspondiente y vuelve a ejecutar TRAP para volver a buscar la página que ahora tiene $P=1$.

Algoritmos de Reemplazo de Páginas

<p>Reemplazo Óptimo Es imposible de implementar porque requeriría saber que páginas necesitaré en el futuro.</p>	<p>No Usadas Recientemente Pág. contienen 2 bits: R: se hace 1 con cada referencia. Borrado periódicamente. M:con cada escritura Clases: 0. R = 0 M = 0 1. R = 0 M = 1 2. R = 1 M = 0 3. R = 1 M = 1 Cuando se produce un fallo de página el MMU elimina una al azar de la clase menor que no esté vacía.</p>	<p>FIFO Se implementa mediante una lista enlazada. La página más vieja se coloca al final de la lista y las que entran van empujando las anteriores hacia el final. Cuando se produce un fallo de página, se reemplaza la última de la lista, osea la más vieja.</p>	<p>Segunda Oportunidad Usa una lista enlazada. Es FIFO modificada porque antes de eliminar verifica el BIT R, si está en 1 no elimina la más vieja (en tiempo) porque fue accedida recientemente, busca la más vieja que tenga R = 0. Puede pasar que un programa ocupe toda la memoria y entre en bucle.</p> 
<p>Reloj Para no desplazar elementos de una lista, mantiene los marcos de páginas en una lista circular, cuya entrada apunta al elemento de lista más antiguo. Si R = 1 en ese elemento el puntero se mueve a la siguiente entrada más vieja con R = 0 hasta encontrar uno reemplazable.</p> 	<p>Menos usadas Recientemente Es la mejor aproximación a un algoritmo óptimo. Cuando se produce un fallo de página, se descarta la página que no se haya usado por más tiempo. Es el de la matriz de unos y ceros. Filas en 1 cuando se usa y columnas en cero cuando no, con cada acceso se actualiza</p>	<p>Conjunto de Trabajo Comienza sin páginas en memoria. Se le llama de “Páginas bajo demanda” pq éstas se cargan sólo cuando son necesarias. “Conjunto de trabajo” es el conjunto de todas las páginas que usa un proceso en un momento dado. “Sobrepaginado” es un proceso que produce fallos de páginas cada pocas instrucciones pq está mal paginado. “Prepaginación” es el proceso de cargar el Conjunto de Trabajo antes de ejecutar un proceso.</p>	<p>WSCLOCK Mejora el algoritmos de “Conjunto de Trabajo” mediante la lista enlazada tipo reloj. Porque el anterior debe explorar todas las páginas para ir conformando el conjunto de trabajo con cada fallo de página.</p>

Conclusiones de los Algoritmos

- El óptimo (imposible) debería eliminar la página que solo se usará en futuro lejano.
- El NRU divide las páginas en 4 clases según el R y M.
- FIFO lleva un registro del orden y la antigüedad con una lista enlazada.
- El de segunda oportunidad comprueba si la más vieja se usó recientemente antes de borrarla y la manda al inicio de la lista como si fuera nueva.
- LRU es el más óptimo pero requiere hardware especial.
- Conjunto de trabajo y WSCLOCK usan grupos de páginas. El más fácil de implementar y rápido es el WSCLOCK.

Segmentación

La memoria virtual por si sola es unidimensional, muchos procesos prefieren 2 o más espacios de direcciones lineales virtuales separados. Una solución simple y general es proporcionar espacios de direcciones diferentes e independientes llamados **SEGMENTOS**.

Segmento: secuencia lineal de direcciones de 0h hasta un máximo. Cada segmento puede crecer o reducir su dirección máxima sin afectar a los otros. Los programadores son conscientes de la existencia de los segmentos, a diferencia de las páginas, y se usan como entidades lógicas. Los segmentos pueden contener cualquier tipo de dato pero no mezclas, osea, son de datos o de código.

Ventajas: como los segmentos son independientes, modificar 1 no afecta al resto. Se facilita la compartición de procedimientos o datos entre procesos. Los segmentos pueden tener distintos tipos de protecciones.

SEGMENTACIÓN	PAGINACIÓN
Es la asignación de N espacios de direcciones lineales (segmentos) independientes entre sí y de tamaño variable dinámicamente que nunca se solapan entre sí.	Tiene un solo espacio de direcciones por proceso. Como el tamaño es limitado se pueden producir solapes entre espacios de direcciones distintos.

Segmentación y Paginación

	SEGMENTACIÓN	PAGINACIÓN
Afecta al programador	si	no
Cantidad de espacios de direcciones lineales por proceso	muchos	1
¿Puede el tamaño de direcciones que ve el proceso exceder el real?	no	si
¿Puede separar código y datos con protecciones?	si	no
¿Tamaño variable?	si	no
¿Puede compartir entre procesos?	si	no
Razón por la que se inventó	Para separar programas en segmentos (datos, código, pila) con espacios de direcciones independientes, mejorando compartición de datos y procedimientos entre procesos y agregados protecciones.	Para tener un espacio de direcciones lineales mayor que el que físico real sin tener más memoria real.



U2.1: Sistemas Operativos de Tiempo Real

Son más parecidos a una librería de funciones en C. No se protegen frente a errores de las aplicaciones. Simplifica el diseño y los requisitos de hardware.

Tareas

Bloque básico de programa en RTOS, es una función en C que no debe terminar nunca, debe ser un loop. Se inicializa mediante una llamada al SO especificando:

- Prioridad
- Memoria
- Función que la llama
- Etc.

El número máximo de tareas está limitado por la cantidad de memoria disponible. Se debe limitar el número de prioridades pq ocupan memoria.

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode, const char * const pcName, uint16_t usStackDepth,  
void *pvParameters, UBaseType_t uxPriority, TaskHandle_t *pxCreatedTask );
```

Nombre de la Tarea (puntero a la función de la tarea (o sea, el nombre nada más), “nombre del dubugueador”, tamaño del stack en número o en variable, enviar un parámetro como puntero void, prioridad de la tarea en número o variable, nombre que se utilizará como manejador)

```
vTaskPrioritySet(manejadorTarea,Prioridad);
```

Con esta función se puede cambiar la prioridad de una tarea. El manejadorTarea debe ser definido globalmente como TaskHandle_t manejadorTarea;

```
uxTaskPriorityGet(manejadorTarea);
```

Con esta función se puede obtener la prioridad de una tarea. El manejadorTarea debe ser definido globalmente como TaskHandle_t manejadorTarea;

```
vTaskDelete(manejadorTarea);
```

Con esta función se elimina la tarea utilizando el manejador. Este manejadorTarea debe ser definido globalmente como TaskHandle_t manejadorTarea;

Estados de las Tareas

- Ejecución
- Lista
- Bloqueada
- Suspendida: es un estado especial en el que una tarea no se ejecuta nunca hasta que otra la despierte mediante una llamada a sistema pasándola a estado listo.

Planificador

Controla el estado de las tareas. La que tenga mayor prioridad es la que estará siempre en running a menos que se haya bloqueado, cedido el control o terminado un loop.

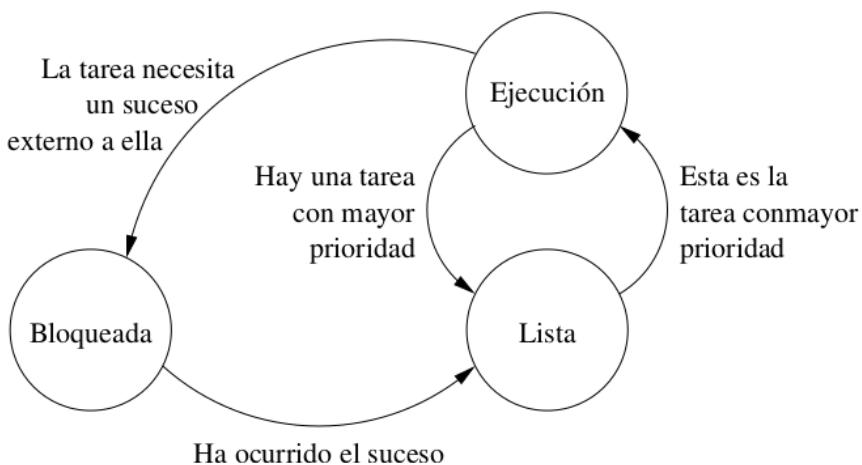


Figura 4.1: Estados de una tarea.

- Tarea sólo puede bloquearse cuando se está ejecutando
- Para qué tarea bloqueada pase a lista otra debe despertarla.
- Tarea lista solo se ejecuta si el planificador lo decide.

- ❖ Tarea se auto bloquea llamando al sistema
- ❖ Si no hay tareas el SO ejecuta bucle sin fin (idle task)
- ❖ 2 de = prioridad están listas,

según el SO ejecuta cualquiera de las dos, en serie o en paralelo o no permite que hayan tareas de igual prioridad.

- ❖ Si una de mayor prioridad se desbloquea, el SO expropiativo, detiene la que se estaba ejecutando de menor prioridad y ejecuta la de mayor. Si el sistema no es expropiativo esperará que la tarea ceda el CPU.

Tareas y Datos

Cada tarea tiene contexto, formado por:

- Registros internos del CPU
- Contador de programa
- PILA: se usa para llamadas a funciones y variables locales

Variables globales son compartidas por todas las tareas y no se guardan al cambiar de tarea. Los cambios de tareas son transparentes para el programador.

Funciones Reentrantes

Pueden ser llamadas desde distintas tareas. Debe cumplir:

- a. Solo puede usar variables de forma atómica, salvo que las variables que usa estén en la pila de la tarea que llama la función, en ese caso manejaría los punteros a esos datos.
- b. Solo puede llamar, la función reentrante a otras funciones reentrantes.
- c. Debe usar el hardware de manera atómica.

Semáforos

Los recursos compartidos se pueden gobernar mediante semáforos.

La llamada `xSemaphoreTake` bloquea la tarea que la llama hasta que el semáforo se libere con `xSemaphoreGive`. En un sistema de tiempo real pueden haber declarados varios semáforos, por lo que le tenemos que pasar el descriptor del que queremos usar.

Los semáforos se definen como variables globales. Lo último que se llama dentro del main es la función del planificador: `vTaskStartScheduler ()`;

Las tareas se crean con: `xTaskCreate()`;

`xSemaphoreTake();` recibe como entrada el descriptor del semáforo y además una variable de temporizador sobre cuánto tiempo espera bloqueada la tarea antes de dar error pq no pudo tomar el semáforo.

Múltiples Semáforos

CADA SEMÁFORO SOLO PUEDE PROTEGER UN SOLO RECURSO COMPARTIDO

Para evitar confusión se suele usar una sola función para el manejo de cada recurso compartido en lugar de usar 500 semáforos.

Semáforos para sincronizar tareas

Además de proteger recursos compartidos, también pueden sincronizar dos tareas entre sí, o una tarea y una ISR. Básicamente hacemos que una tarea se bloquee porque otra tarea está usando el semáforo y recién cuando esta termine se lo dá. Por más que no usen variables compartidas. En el caso de una ISR puede ser cuando se obtiene un dato por un ADC o cualquier poronga. PARA QUE ESTO FUNCIONE SE PIDE EL SEMÁFORO EN EL MAIN APENAS LO CREAMOS PARA QUE LA TAREA QUE QUEREMOS SINCRONIZAR SE BLOQUEE NI BIEN ARRANQUE EL SISTEMA.

Ventajas

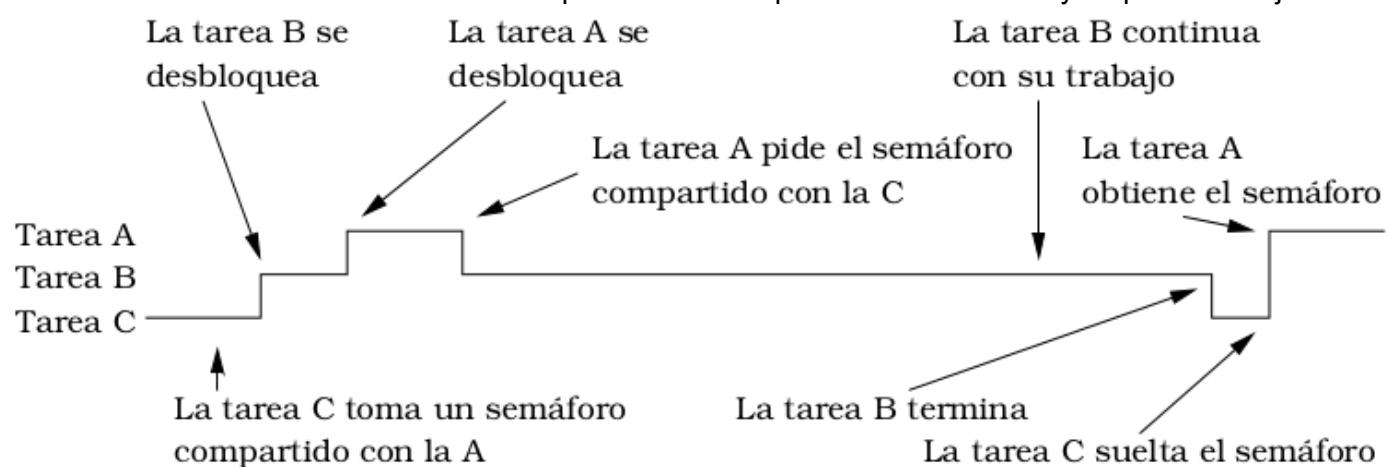
- Frente a bucles SCAN, la latencia es mucho menor.
- No se pierden ciclos del CPU comprobando banderas chotas ni otras vergas. Si la tarea se bloquea esperando una ISR que desbloquee el semáforo, no ocupa CPU hasta que esto pase.

Problemas con los semáforos [la mayoría son por boludo]

- Olvidar tomar el semáforo
- Tomar el equivocado
- No soltar el semáforo
- Tomarlo demasiado tiempo, aumenta la latencia de las demás tareas que lo usan
- Inversión de prioridad.

Inversión de Prioridad

Ocurre cuando una tarea de menor prioridad le impide a una de mayor prioridad ejecutarse.



Esto pasa cuando una tarea de menor prioridad se ve interrumpida por otra de mayor prioridad antes de soltar el semáforo, esto causa que una tercera tarea de mayor prioridad que la bloqueada se bloquee esperando que la tarea pete suelte el semáforo. Esto se arregla con mutex. Que son semáforos que heredan la prioridad de las tareas con las son compartidos. De esta forma no se quedan a medio camino si otra tarea quiere cagarla.

Abrazo mortal

Se da cuando dos tareas se bloquean mutuamente porque los mutexes o semáforos se piden en el orden incorrecto. La forma de evitarlo es pidiendo siempre los semáforos y mutexes en el mismo orden en ambas tareas.

Resumen de métodos para proteger recursos compartidos

Deshabilitar interrupciones	Inhabilitar Comutación de tareas	Semáforos	Mutex
Afecta tiempos de respuesta de todas las tareas e ISR. Es el método más rápido y el único que protege datos compartidos con ISR. Solo se recomienda usar cuando la zona crítica es pequeña.	Solo afecta tiempos de respuesta de tareas en primer plano. Se realiza con llamada al sistema vTaskSuspendAll(); luego se llama a xTaskResumeAll(); Las ISR siguen funcionando.	Solo afectan a las tareas que comparten el recurso. Son un mecanismo más complejo que presenta más carga al sistema.	Son lo mismo que semáforo pero heredan la prioridad de la tarea de mayor prioridad que usa el mismo semáforo. Con esto evitan la inversión de prioridad.

Colas para comunicar tareas

Se usan cuando:

- Se necesita un almacenamiento temporal o buffer para soportar ráfagas muy rápidas de datos
- Cuando existen varios generadores de datos y un solo consumidor, se pueden bloquear los generadores hasta que el consumidor puede continuar recibiendo.

Características de las Colas (cola):

- Se crean mediante llamadas a sistema: crea el manejador y reserva la memoria necesaria
- El manejo de la cola se hace solo por llamadas al sistema
- En freeRTOS se pueden enviar cualquier tipo de elementos.
- Las funciones que envían y reciben se pueden bloquear porque está llena o vacía.

Creación de colas en FreeRTOS (queue) [sorry es que me la deja picando el título]

Se hace en el main antes de comenzar el planificador mediante:

```
xQueueHandle cola_err ; // declaro el manejador de la cola como variable global  
cola_err = xQueueCreate (TAM_COLA , TAM_MSG ); // en el main la inicio
```

Le debo pasar el tamaño de la cola en cantidad de mensajes que guardará el buffer y el tamaño de cada mensaje.

Envío de datos a la cola [no se lo que estás pensando y no, no es eso]

Se usa la función:

```
xQueueSend (cola_err , (void *) cad_err ,( portTickType ) 100);
```

Se le pasa:

- El manejador de la cola con la que lo creamos.
- Un puntero al dato que se quiere enviar. Se debe castear a void para que la función sea genérica a cualquier tipo de dato.
- El timeout para dar error por si la cola está llena y no entra más nada. Se suele poner en cero para que la tarea no se bloquee si la cola está llena. [posta deja los chistes picando]
- La función devuelve pdTRUE o pdFALSE si se pudo o no enviar el dato al terminar el timeout.

Recepción de datos

```
xQueueReceive (cola_err , (void *) cad_rec ,( portTickType ) 0xFFFFFFFF );
```

Se le pasa:

- El manejador de la cola
- Un puntero a un buffer, de nuevo casteado void por generalización,
- Un tiempo de espera. Devuelve pdTRUE si hay datos o pdFALSE si no hay nada al terminar el timeout. También devuelve pdTRUE si durante el tiempo de espera aparece un dato.

Rutinas de atención a interrupciones

Precauciones que se deben tener al escribir ISR's:

- No llamar a funciones de sistema que se puedan bloquear. Para ello existen funciones especiales atómicas.
- No llamar funciones de sistema que puedan producir conmutaciones de tareas. Salvo que el sistema sepa que se está ejecutando una interrupción, para esto hay funciones especiales.

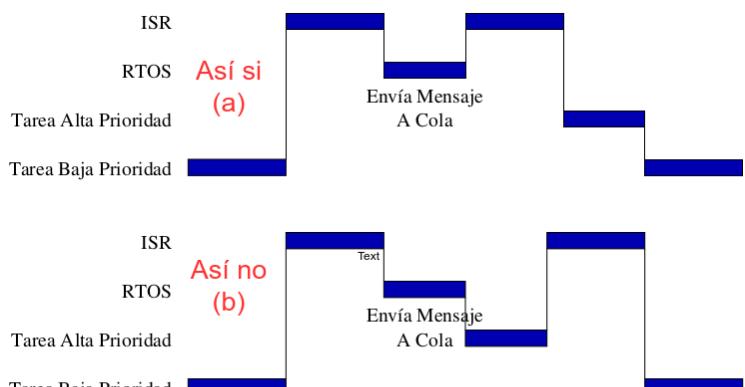


Figura 4.5: Interrupciones y sistemas operativos en tiempo real

Lo que debe pasar cuando una ISR desbloquea una tarea, por cualquier motivo, es que el SO se entera de lo que tiene que hacer pero en lugar de irse a la tarea, lo hace atómicamente en (a). Y recién luego de terminar la ISR y haber hecho el envío a cola atómico, va a la tarea de alta prioridad.

Lo que no tiene que pasar es que el sistema commute a tareas de alta o baja prioridad sin antes haber salido de la rutina de atención a la interrupción(b).

Para evitar esta cagada, FreeRTOS implementa funciones atómicas para llamar desde ISR's.

Rutinas de atención a interrupciones en FreeRTOS

En freeRTOS hay dos llamadas a sistema distintas:

- Las normales [duh]
- Las diseñadas para ser llamadas desde ISR's

Estas últimas tienen la característica de que:

- No pueden bloquearse
- No producen cambios de contextos. Lo debo hacer a mano al finalizar la ISR. La idea es que apenas termine la ISR se vaya a la tarea que ésta debería haber despertado.

Funciones

```
xQueueSendFromISR ( xQueueHandle xQueue ,const void * pvItemToQueue ,portBASE_TYPE xTaskPreviouslyWoken );
```

Lo mismo que la otra verga, solo que ahora agrega: xTaskPreviouslyWoken que permite desde la ISR hacer varias llamadas a xQueueSendFromISR u otras funciones atómicas.

La función devuelve pdTRUE si ha despertado alguna tarea o pdFALSE si no.

```
portBASE_TYPE xQueueReceiveFromISR ( xQueueHandle xQueue ,void *pvBuffer , portBASE_TYPE *pxTaskWoken );
```

*pxTaskWoken permite que desde la ISR se puedan hacer varias llamadas a xQueueReceiveFromISR o similares. Este puntero ha de contener la dirección de una variable que en un principio se iniciará a pdFALSE . Si alguna de las llamadas a xQueueReceiveFromISR hace que alguna tarea tenga que esperar por estar esperando a que se liberara sitio en la cola, la variable *pxTaskWoken pasará a valer pdTRUE ,pero si no mantendrá su valor.

Gestión de tiempo

Es necesario gestionar el tiempo en rtos para poder realizar timeouts en las funciones que se bloquean esperando mensajes o semáforos.

La gestión de tiempo se basa en usar una interrupción periódica que incrementa un contador con el que se lleva la cuenta del tiempo transcurrido desde el inicio del sistema. Cada incremento del contador se denomina “tick” de reloj. Normalmente el periodo del tick es configurable.

Funciones

```
void vTaskDelay(portTickType xTicksToDelay);//Suspende la tarea durante el número de ticks especificado.
```

Produce una demora de Xticks cediendo el control, queda bloqueada.

```
void vTaskDelayUntil ( portTickType * pxPreviousWakeTime ,portTickType xTimeIncrement );
```

Asegura un tiempo constante entre los sucesos llamados a esta función

Suspende la tarea hasta el instante especificado por la expresión:

```
*pxPreviousWakeTime + xTimeIncrement .
```

FreeRTOS define una constante denominada configTICK_RATE_MS con el número de milisegundos que dura un tick.

Nota sobre EDU-CIAA

La placa 4 anda mal. Los leds se numeran como: 0, 1 y 2 (RGB); 3 (Amarillo), 4 (Rojo) y 5 (Verde).

```
Board_LED_Toggle(0); /* Red (RGB) */  
Board_LED_Toggle(1); /* Green (RGB) */  
Board_LED_Toggle(2); /* Blue (RGB) */  
Board_LED_Toggle(3); /* Yellow */  
Board_LED_Toggle(4); /* Red */  
Board_LED_Toggle(5); /* Green */
```

IDLE_HOOK dentro de FreeRTOSConfig.h debe ser 1 si se va a utilizar la función Idle o 0 si no se va a utilizar ya que sino genera error al compilar

Para usar el hook aparte de poner el 1 en el archivo de configuración se tiene que poner esta función sin cambiar el nombre: `void vApplicationIdleHook(void){CÓDIGO DE LA FUNCIÓN}`

No es necesario colocar nada más.

Para la función periódica con interrupción se ponen estas funciones antes de las tareas:

```
static void initHardware(void) {  
    SystemCoreClockUpdate();  
    Board_Init();  
}  
  
static void InitTimer(void) {  
    Chip_RIT_Init(LPC_RITIMER);  
    Chip_RIT_SetTimerInterval(LPC_RITIMER, 3000);  
    // modificando el segundo argumento cambio los  
    // ms entre cada interrupcion, en este caso 3000ms para lograr una  
    // interrupcion cada 3s  
}  
  
void RIT_IRQHandler(void) {
```

ACÁ PONES TU CÓDIGO PARA QUE CORRA EN LA INTERRUPCIÓN

/*Si cualquier función que llamo desde la interrupción despierta alguna tarea de alta prioridad, cambia el valor de la variable declarada “ xHigherPriorityTaskWoken a pdTRUE, con esa variable hacemos un if y si es verdadero, significa que tenemos que salir de la interrupción haciendo un cambio de contexto a la tarea de mayor prioridad que se despertó por la interrupción. Para esto llama a la función yiel con el argumento true de la variable. Finalmente se debe borrar la bandera de la interrupción que originó la ISR. */

```
 BaseType_t xHigherPriorityTaskWoken = pdFALSE;  
/* Despierta las tareas */  
xSemaphoreGiveFromISR(Semaforo, &xHigherPriorityTaskWoken);  
if (xHigherPriorityTaskWoken == pdTRUE) {  
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);/* Si el semáforo ha despertado una tarea , se fuerza  
    un cambio de contexto */  
}  
/* Borra el flag de interrupción */  
Chip_RIT_ClearInt(LPC_RITIMER);  
}
```

Dentro de main() debo colocar:

```
/*---- Codigo para iniciar la interrupcion -----*/  
initHardware(); /* Inicializa el Hardware del microcontrolador */  
InitTimer();  
/*-----Mas cosas para la interrupcion -----*/
```

```
NVIC_EnableIRQ(RITIMER IRQn);
/* Set lowest priority for RIT */
NVIC_SetPriority(RITIMER IRQn, (1 << __NVIC_PRIO_BITS) - 1);
```

Pool de librerías que nunca fallan:

```
#include "FreeRTOS.h"
#include "board.h"
#include "task.h"
#include "supporting_functions.h"
#include "stdlib.h"
#include "stdio.h"
#include "math.h"
#include "string.h"
#include "queue.h"
#include "semphr.h"
#include "chip.h"
```

U4: Numerología [we] [dieron la U4 primero que la 3, cualquier queja diríjase al pela]

Cuando uno quiere representar números reales en una computadora, hay infinitos números y para representar estos infinitos números necesitaríamos infinitos bits, es imposible. Hablamos de una representación finita de números. Lo que hace el programador es ajustar esta representación al problema a resolver. Todas las representaciones tienen sus ventajas y desventajas.

Representación de enteros

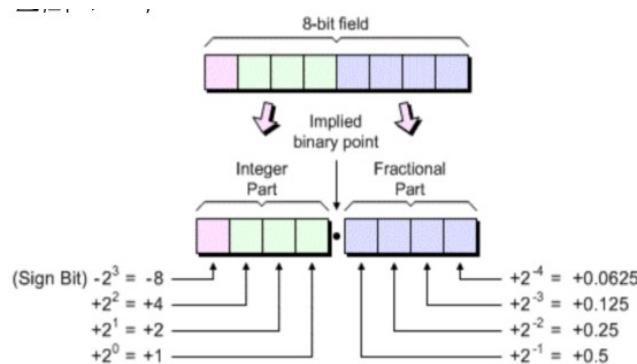
Bit Pattern	Unsigned	2's Complement
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
•	•	•
•	•	•
0111 1110	126	126
0111 1111	127	127
1000 0000	128	-128
1000 0001	129	-127
•	•	•
•	•	•
1111 1110	254	-2
1111 1111	255	-1

Una palabra de N bits puede representar de 0 a $2^N - 1$ enteros.
Le restamos 1 pq una palabra representa el cero.

En complemento a 2 el rango es de: -2^{N-1} a $2^N - 1 - 1$. Se usa por eficiencia en hardware. Permite sumar y restar con el mismo sumador.

Acá se muestra como se pasa de complemento base 2 a entero decimal. Si b_{N-1} es 1 los números serán negativos, si no son positivos.

Representación en PUNTO FIJO



Representamos una palabra de N bits:

- m para la parte entera
- n para la parte fraccional
- un bit extra para el signo (positivo o negativo)

El punto fijo es una generalización para representar un entero. Utilizamos las potencias negativas de 2 para representar la parte fraccional.

Tenemos la ecuación que va de binario a decimal, el primer término son los números negativos, el segundo término compone los números positivos y su parte fraccional con el exponente negativo que habíamos fijado como la parte fraccional.

$$n_{10} = -b_m 2^m + \left(\sum_{i=0}^{m-1} b_i 2^i + \sum_{i=1}^n b_i 2^{-i} \right).$$

"n" es la cantidad de bits, nos dice la precisión , ya que va a saltar de 2^{-n} en 2^{-n} . Si n=1 entonces salta de 0,5 en 0,5. El rango esta determinado por m y n, va de -2^m a $2^m - 2^{-n}$ ya que es desde el menor número al mayor posible. Si n=0, la precisión es 1, son los números enteros, es un caso especial, ya que represento 1,2,3... La diferencia entre estos es 1

El nombre de punto fijo representa lo contrario de punto flotante, es porque en punto flotante, el punto cambia de lugar a medida que cambia el número

Notación: Qm.n donde m es la cantidad de bits enteros y n la cantidad de la parte decimal. El rango es:

-2^m a $2^m \cdot 2^{-n}$

Conversión de Real a Punto Fijo

Unit: $z = 1 << n = 1 \cdot 2^n$.

Example: $n = 4 \Rightarrow z = 1.0000_2$.

One half (1/2): $z = 1 << (n - 1) = 1 \cdot 2^{(n-1)}$.

Example: $n = 4 \Rightarrow z = 0.1000_2$.

- ¿Cómo se define un 1 para pasarlo en punto fijo? Tengo que tomar el número 1 y agregarle bits en 0 a la derecha tantos bits como parte fraccionaria tenga, esto es, tantos n como tenga, el " $<<$ " realiza este desplazamiento a la izquierda, que es equivalente a multiplicar el 1 por 2^n .

-Para representar el 1/2 o 0,5 tengo que desplazar el 1 $n-1$ lugares a la izquierda.

Punto Flotante a Punto Fijo

$X := (\text{int})(x \cdot (1 << n))$

$X := (\text{int})(x \cdot 2^n)$

-Para pasar de punto flotante a punto fijo multiplicamos el número con un 1 corrido tantos lugares a la izquierda como n tengamos. Esto es equivalente a multiplicar por 2^n . RECORDEMOS QUE SE DEBE CASTEAR, QUE OBLIGUE A REPRESENTAR EL NÚMERO EN EL FORMATO QUE QUEREMOS.

Punto Fijo a Punto Flotante

$x := (\text{float})(X)/(1 << n)$

$x := (\text{float})(X) \cdot 2^{-n}$

-Para pasar de punto fijo a punto flotante tenemos que dividir el valor por un 1 corrido tantos n como tengamos en la parte fraccional. Es equivalente a multiplicarlo por 2^{-n} . Recordar castear

Factor de Escala

La ALU no distingue si está utilizando entero o punto fijo, ambas variables se definen como un entero, el programador no puede avisarle eso a la CPU, lo tiene que interpretar el programador. Para diferenciar, se toma el factor de escala, esto está en la cabeza del programador. El punto fijo es un entero multiplicado por un factor de escala, esto es la precisión, 2^{-n} . Puede ser un número arbitrario el factor de escala, no necesariamente un factor de base 2. Pasos:

- 1) Se toman los enteros con signo
- 2) Se multiplican estos enteros por la precisión, dando un nuevo rango.
- 3) Se multiplica por un nuevo número, siendo el factor de escala total:

x (nro cualquiera) $\cdot 2^{-n}$ (nro de bits de la parte decimal)

Rango Dinámico

$$DR_{dB} = 20 \log_{10} \left(\frac{\text{largest possible word value}}{\text{smallest possible word value}} \right) \quad [\text{dB}]$$

Nos da una idea de cuántos números podemos representar con una cantidad definida de bits, este caso, n . Sirve para comparar los rangos dinámicos de los números representados con PF y Float. La expresión final nos da el cálculo directo con N bits.

Para enteros representados con N bits: =====>

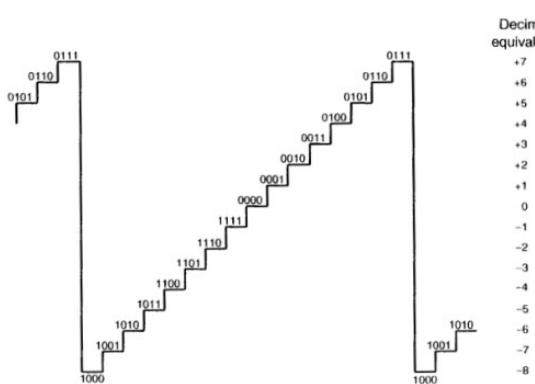
$$DR_{dB} = 20 \log_{10} \left[\frac{2^{(N-1)} - 1}{1} \right] \quad [\text{dB}]$$

$$DR_{dB} \approx 20 [(N - 1) \log_{10}(2)]$$

$$DR_{dB} \approx 20 \log_{10}(2) \cdot (N - 1)$$

$$DR_{dB} \approx 6.02 \cdot (N - 1) \quad [\text{dB}]$$

Overflow



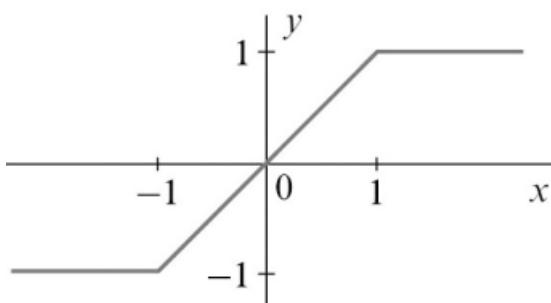
El overflow se produce cuando luego de sumar o restar salga de rango, ya sea cuando sumamos dos positivos o restamos dos negativos. Esto también se lo conoce como roll-over, es como que se desfaza de las cuentas

También puede cambiar el bit de signo si estamos trabajando en complemento a dos y se pasa de rango.

Para evitarlo se guardan los resultados de operaciones matemáticas en acumuladores con $N+1$ bits.

Regla general: acumulador = $m + \log_2(s)$ m: largo de palabras en bits, s: cantidad de sumas a realizar.

Saturación

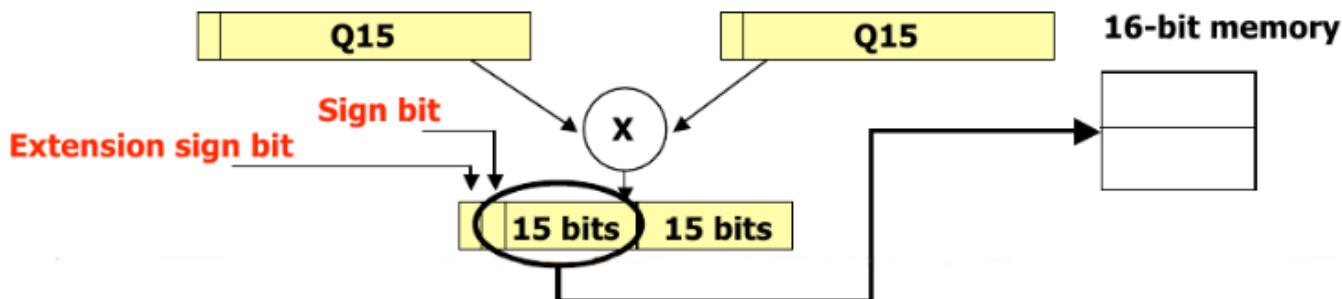


Las sumas no producen roll-over ya que se limita el valor máximo al que se le permite llegar a una operación aritmética.

Transforma todo en un sistema alineal. Esto nos produce una alinealidad importante en los filtros por lo que tenemos que tener en cuenta donde vamos a trabajar.

Puede haber saturación por hardware o software.

Multiplicación en complemento a 2

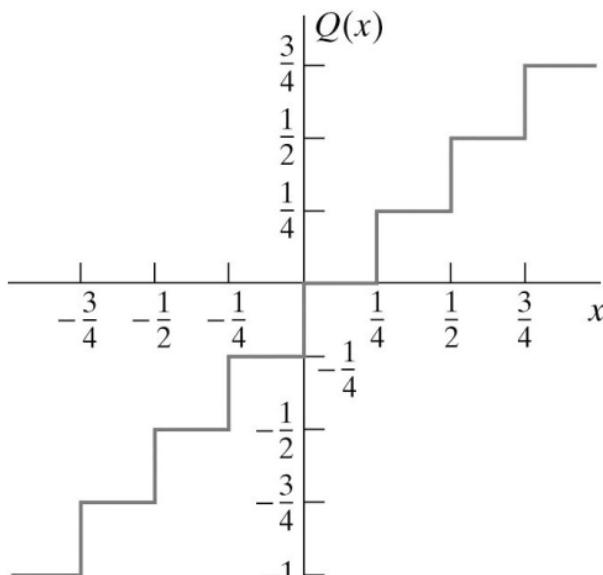


Si multiplicamos dos números con N bits, obtenemos un número de $2N$ bits como máximo. Pero esto está compuesto por 2 bits de signo y los otros bits de los números multiplicados. Por lo que se genera un desplazamiento a la izquierda para eliminar ese bit de signo y con solo $2N-1$ bits, podemos tener el resultado. También nuestra precisión aumenta en el doble, el resultado es un número más exacto. La multiplicación de este ejemplo es con formato Q15 (15 bits para parte decimal), por eso aumentó la precisión, ya que tenemos 30 bits de precisión. Cuando el nuevo número pasa a la memoria, tiene que plantearse la situación de que se hace con los nuevos bits de precisión, se quedan o se van?

Underflow

El fenómeno de recortar los bits sobrantes o disminuir la precisión, se lo reconoce como un underflow. Pasamos de una precisión de 2^{-n} a 2^{-2n} pero tenemos que volvernos al valor que corresponde a nuestro formato, es decir, 2^{-n} . Hay que decidir qué valor va a tomar. En la recta se ven las representaciones posibles para Q0.3. El valor se ubica entre dos fracciones de Q0.3, hay que aproximarse hacia algún lado.

Esquemas de Redondeo



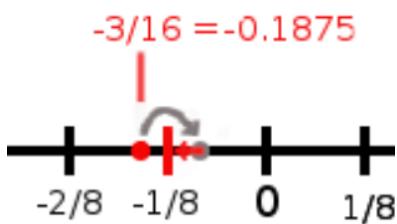
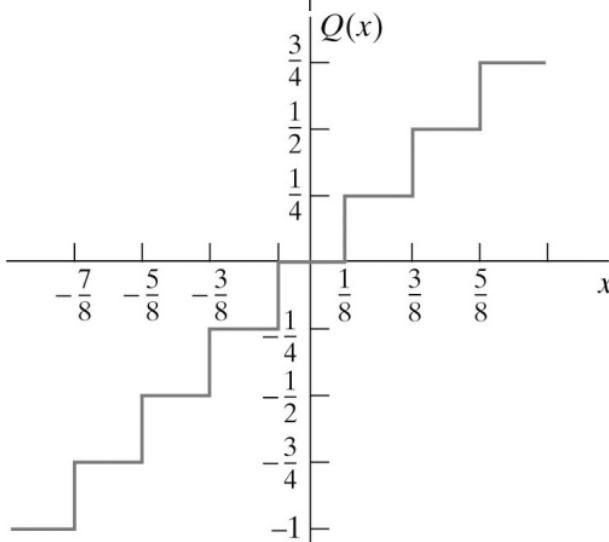
Redondeo hacia -infinito

Esquemas de redondeo, es la forma en que vamos a decidir qué número toma el nuevo número truncado. En truncación se eliminan los bits extras luego de operar, se recorta, de modo que redondea hacia menos infinito, siempre redondea hacia abajo. Para hacer esto lo que hacemos es desplazar el número obtenido n lugares de modo que se "trunquen" esos bits hacia la derecha, debemos castearlo al nuevo formato.

Si vemos la gráfica, nos damos cuenta que los valores entre un punto y los que se encuentren a la izquierda, se quedan como el menor punto, redondea hacia el menos infinito

Redondeo hacia el más cercano

Este redondea hacia al más cercano. Esto es sumando al número el valor $2^{-(n+1)}$ que es igual a la mitad de la precisión. Esto se hace porque al sumar el medio valor de precisión, salta medio lugar hacia adelante y al truncar se acerca más al valor más cercano.



Errores en redondeo

Round-off tiene menor error. Los modelos se modelan con una distribución uniforme, con la variancia y la media se estudia el error. Al ser la media =0 tenemos un sesgo 0 y es igual a decir que tiene menor error. Tenemos un sesgo igual a cero en round-off.

Operación MAC [Multiplicar y Acumular]

C code

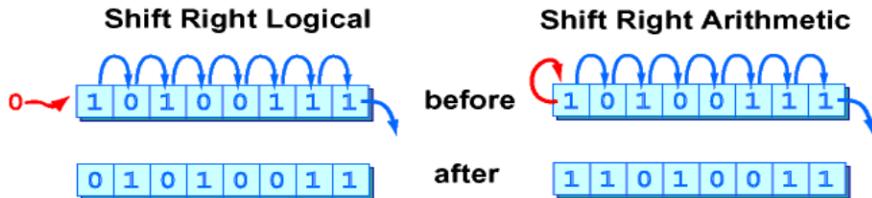
```

1 int32_t a[K] ;
2 int32_t b[K] ;
3 int64_t c = 0 ;
4 for (i=0; i<K; i++)
5 { c = c + ( (int64_t) a[i] * (int64_t) b[i] ) ; }
```

Se utiliza para

convolucionar y hacer la FFT. MAC multiplica el contenido de dos vectores elemento a elemento y lo va acumulando en una variable resultado de esa multiplicación. Al comprender dos operaciones, es más susceptible a tener problemas con los bits, obtenemos tanto overflow como underflow. Los procesadores DSP cuentan con bits extras para evitar problemas.

Desplazamientos Aritméticos



Desplazar un número binario un lugar a la izquierda es multiplicarlo por 2 y hacia la derecha es dividirlo por 2. Desde el punto de vista de cálculo es muy fácil multiplicar o dividir por 2.

En un desplazamiento aritmético, debemos guardar el signo. En el desplazamiento lógico, no guardamos el signo, por lo que cambia de un número negativo a un número positivo. MANTIENE EL 1 O 0 QUE TENÍA EL NRO EN EL msb.

Desplazamiento Aritmético BUENO Desplazamiento Lógico MALO

Representación de Punto Flotante

La característica más importante es que puede representarse números muy chicos y números muy grandes.

Expresión del Flotante donde:

$$(-1)^S \times F \times r^E$$

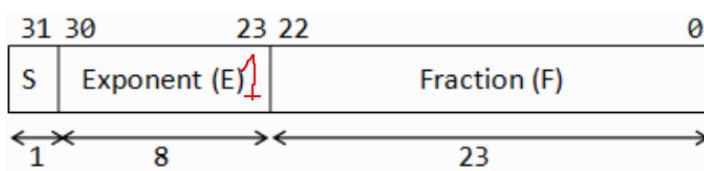
S bit de signo, F la mantisa o parte fraccional(tiene un 1 implícito adelante), r base del número, E es el exponente

El estándar IEEE 754 consta de la norma de representación actual

Parameter	Binary formats ($B = 2$)				Decimal formats ($B = 10$)			que se
	Binary 16	Binary 32	Binary 64	Binary 128	Decimal 132	Decimal 164	Decimal 128	
p , digits	$10 + 1$	$23 + 1$	$52 + 1$	$112 + 1$	7	16	34	
e_{max}	+15	+127	+1023	+16383	+96	+384	+16,383	
e_{min}	-14	-126	-1022	-16382	-95	-383	-16,382	
Common name	Half precision	Single precision	Double precision	Quadruple precision				

utilizan en las FPU. Están las reglas de redondeo, operaciones y excepciones. 16 bits es half precisión, 32 es single precision y 64 es double precision.

Precisión Simple 32 bits



Comienza con 1 bit de signo, 8 para el exponente y 23 para el fraccionario. El bit de signo vale 0 para números positivos y 1 para números negativos. La parte fraccionaria de 23 bits responden a potencias negativas de 2, esto es $2^{-1} \dots 2^{-23}$. El exponente no tiene signo, puede valer entre 1 y 254. El sesgo vale 127; los valores 0 y 255 están reservados. Si resto el exponente y el sesgo nos da, de qué número a qué número puedo representar; esto sirve para evitar ponerle signo al mismo.

Al mover el punto, voy aumentando o disminuyendo el exponente, representan todos el mismo número. Para evitar estas representaciones múltiples, buscamos que el número quede representado siempre como $1.zxcvb\dots$ de modo que tiene un 1 implícito que no es parte de los 32 bits. Esta manera se llama forma normalizada de representación de números.

Multiplicar y dividir se hace aumentando o disminuyendo el exponente, se suele perder precisión y también son auto-rango. Hemos visto que no podemos representar todos los números con punto flotante, siempre va a pasar, sin importar el formato que utilicemos, es una limitación computacional.

2^E	MIN F 1..0000000000000000000000000000000 (b2) 1 (b10)	MAX F 1..1111111111111111111111111111111 (b2) 1.999999881 (b10)
2^-126	1.1755E-38	2.3510E-38
2^-30	9.3132E-10	1.8626E-09
2^-20	9.5367E-07	1.9073E-06
2^-10	9.7656E-04	1.9531E-03
2^-3	0.12500000	0.24999999
2^-2	0.25000000	0.49999997
2^-1	0.50000000	0.99999994
2^0	1.00000000	1.99999988
2^1	2.00000000	3.99999976
2^2	4.00000000	7.99999952
2^3	8.00000000	15.99999905
2^10	1.0240E+03	2.0480E+03
2^20	1.0486E+06	2.0972E+06
2^30	1.0737E+09	2.1475E+09
2^127	1.7014E+38	3.4028E+38

Son autorango porque al modificar el exponente, lo que hacemos es movernos por diferentes tramos en la recta de los reales. En la primera columna tenemos el exponente, la segunda la mantisa con el mínimo valor posible (con el 1 implícito) y en la tercera columna vemos la mantisa con el valor máximo posible (con el 1 implícito). Vemos que al comparar los rangos que tomamos con cada exponente, estos disminuyen fuertemente en los exponentes negativos y aumentan fuertemente en los exponentes positivos, determinando distintos tamaños.

Valores especiales de Exponente:

- El cero se representa como exponente cero y parte de mantisa en cero. Tenemos 2 ceros, un 0+ y un 0-
- Se puede representar el infinito, con el exponente todo 1 o 255, mantisa 0 y tenemos 2 valores, -inf con s=1 y +inf con s=0
- "no es un número" el exponente es todo 1, la mantisa es diferente de cero, puede tener cualquier cosa. Es para operaciones que no tienen representación, como dividir 0/0.
- Vemos que estas operaciones dan el resultado que tendríamos si lo hacemos con papel y lápiz

Métodos de Redondeo

La distancia entre los números que sí son representables se le llama ulp (unit of least precision), esto es la precisión que teníamos en punto fijo.

- Truncación o redondeo hacia cero. Si f es + toma el valor f' y si f es - va al f'' (LO ACERCA LO MÁS QUE PUEDE AL CENTRO DE LA RECTA, ESTO ES EL 0)
- Redondeo hacia el +inf, redondea hacia la derecha (sin importar si es + o -)
- Redondeo hacia el -inf, redondea la izquierda (sin importar si es + o -)
- Redondeo al valor más cercano, por defecto lo utiliza la FPU cuando arranca la computadora. El redondeo de toda la vida. Lo que hace es tomar f y lo compara con el f' (el menor) sumado a la mitad de la precisión, si f meda que es más chico, quiere decir que se debe redondear al más bajo y si me da mayor, se debe redondear al más grande. Esto es porque al redondear, trunca y se queda con el valor más cercano al fin y al cabo.

Rango Dinámico

$$DR_{db} = 20 \log_{10} \left(\frac{\text{largest possible word value}}{\text{smallest possible word value}} \right)$$

El rango dinámico nos dice la cantidad de números que podemos representar en una computadora con un formato determinado. Se representa como $20\log(>N^{\circ}/<N^{\circ})$.

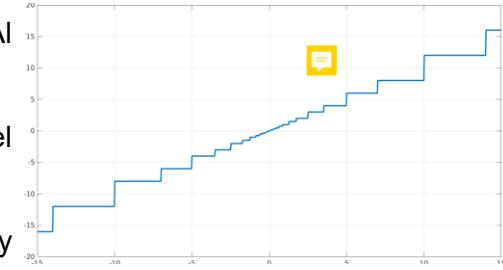
$$DR_{dB} \approx 6.02 \cdot 2^{b_E}$$

Para punto flotante se define como en la foto, donde bE es la cantidad de bits del exponente.

Precisión

Es la distancia entre dos posibles números, y lo definimos en punto fijo como 2^{-n} , con n la cantidad de bits que tenemos en la parte fraccional del formato punto fijo. En punto flotante se define como $2^E \cdot 2^{-23}$

La precisión en punto flotante no es constante en todo el rango. Al agrandarse el número representado crece su precisión.



Cerca del cero se hace casi una línea, a medida que me alejo del centro el escalón crece, por lo que la precisión también crece.

No se recomienda trabajar con valores de números que sean muy diferentes entre sí porque la diferencia de precisión entre ambos hace que de cualquiera la cuenta.

Suma de Flotantes

Representamos los valores en punto fijo binario. Normalizamos los valores a operar, esto es dejar un 1 al frente y lo colocamos con los exponentes. Igualamos los exponentes, el mayor exponente se toma como referencia y los demás se fuerzan a este. El mayor exponente es el más positivo

Para forzar el exponente, se restan ambos exponente guardandolos en una variable "n" y se hacen en desplazamientos hacia la derecha, luego se le suman al exponente. De manera que el número que representa es el mismo que antes.

Luego se suman las mantisas, y se normaliza nuevamente, ya que debemos cumplir con la norma. Si el exponente es menor a -126 hay underflow y si es mayor a 127 es un overflow.

Por último se toma en cuenta si hace falta redondeo.

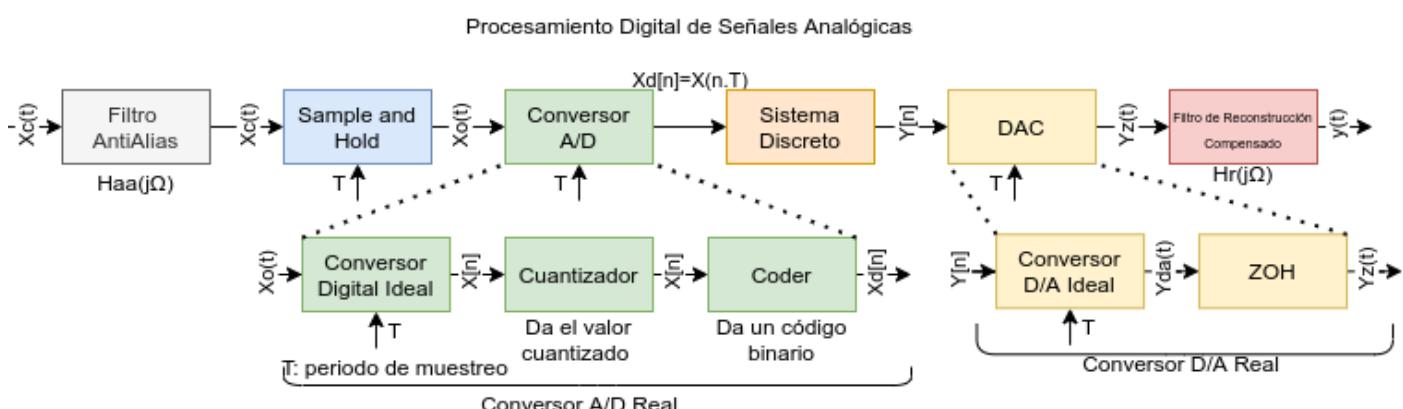
Conclusiones

Tenemos ventajas y desventajas en cada sistema.

Punto flotante tiene gran rango dinámico y no tiene que verificar tanto el underflow y el overflow.

Punto fijo presenta la ventaja de que la precisión es constante en toda la escala, los procesadores suelen ser más baratos (ya que los procesadores con FPU salen el doble casi).

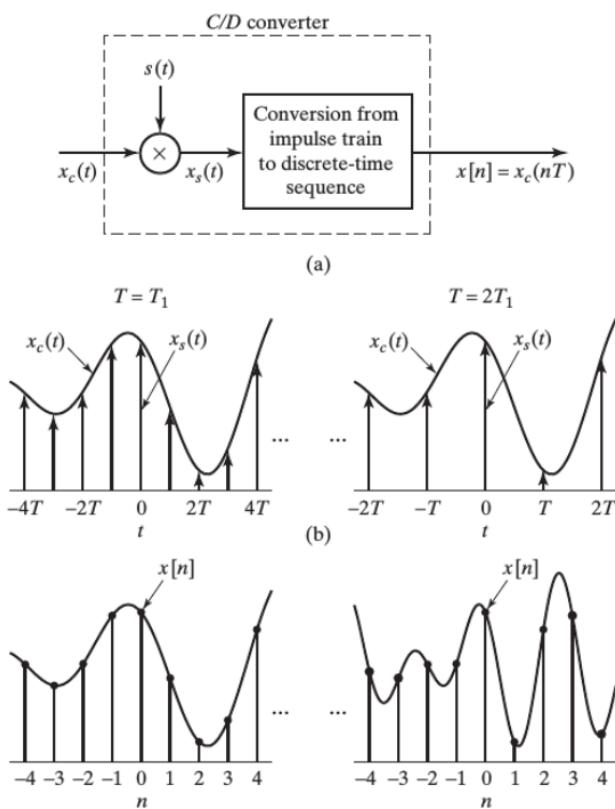
Etapas de Procesamiento de Señales



Muestreo Periódico

Muestramos la señal con un periodo T. Vemos que la $x[n]$ se puede representar como una señal $Xc(nT)$ donde n varía de menos infinito a más infinito y T es el periodo de muestreo, donde $x[n]$ va a poder tomar valores que dependen de n y el periodo T. Vemos que $1/T$ es la frecuencia de muestreo y Omega-s es la frecuencia digital

Proceso de Muestreo



$$X_c(j\Omega) = 0 \text{ para } |\Omega| \geq \Omega_N$$

El proceso de muestreo se puede representar de la siguiente forma, una señal analoga que entra es muestreada por un tren de impulso, siendo su salida convertida a valores binarios de modo que obtenemos finalmente la $x[n]$. Vemos que en:

a) se comporta como una señal de impulsos escalados separados por T y en

b) ya está digitalizada en binarios y esta depende de la variable n directamente. El cero ($n=0$) tomamos como el valor de inicio del sistema. Cuando en el segundo ejemplo tomamos $T=2T_1$ de modo que se toman muestras más espaciadamente, es mal tiempo de muestreo, no es suficiente.

Vemos que por Nyquist-Shannn se determina la mínima frecuencia de muestreo para que se pueda recuperar correctamente. Vemos que la señal debe ser de banda limitada. Omega-N es la máxima frecuencia de X_c y la mínima frecuencia de muestreo será $2\Omega_N$ de modo que sea el doble de la frecuencia máxima de X_c .

$$\Omega_s = \frac{2\pi}{T} \geq 2\Omega_N$$

Representación en el Dominio de la Frec

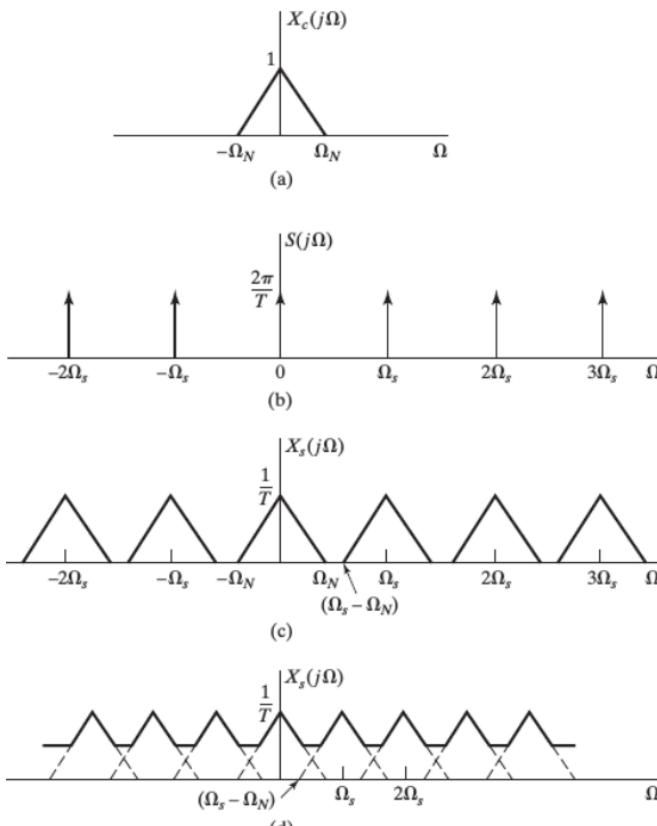
Vemos el espectro original, el tren de impulsos, la respuesta que muestra cómo se han multiplicado los espectros en X_s .

Cuando la frecuencia de Nyquist no se cumple los espectros se interfieren y se produce aliasing.

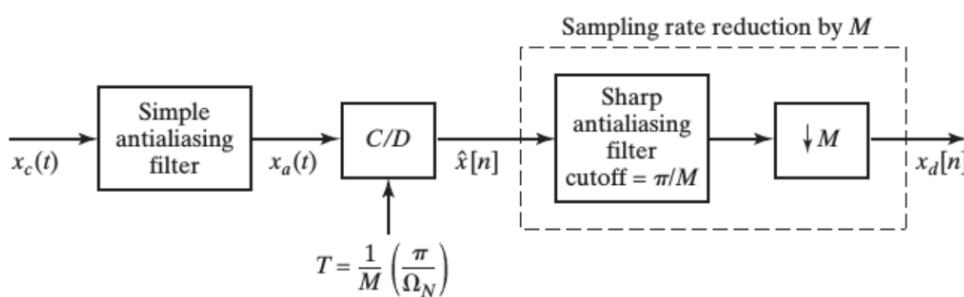
Pre Filtrado

El filtro antialiasing siempre está antes del ADC ya que tiene que filtrar la señal para garantizar que la frecuencia máxima de la señal que entrará al sistema es suficientemente chica como para que se cumpla el teorema de Nyquist, que sea Ω_N . Normalmente es un filtro pasa bajos o pasa banda. Siempre está en el dominio del tiempo.

A pesar de que nuestra señal está limitada a una máxima frecuencia, debemos colocar un filtro para eliminar el ruido, sino igualmente se produce aliasing por no limitar todo lo que entra al sistema, debemos colocar una ventana que limita lo que entra, esto a se limita a la máxima frecuencia.



Oversampling



Podemos utilizar oversampling. Cuando veíamos el filtro que teníamos al principio, la ventana que cortaba las frecuencias de ruido, era muy exigente. El oversampling es utilizar un filtro simple de anti aliasing combinado con un circuito de muestreo que tiene una

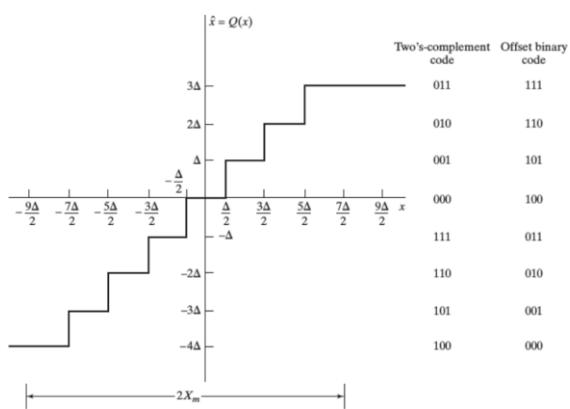
frecuencia ALTA de muestreo y en el dominio digital utilizamos un filtro de corte abrupto y luego disminuimos la frecuencia de muestreo para trabajar la señal de forma normal.

Conversor ADC

El ADC se divide en dos partes. El Sample and hold lo que hace es tomar una muestra y mantenerla para que sea convertida a binario por el ADC en sí, esto lo hace porque toma un tiempo la conversión.

El conversor se divide en 3 etapas, la conversión en sí, toma la muestra de tensión, luego esta muestra la pasa al cuantizador que determina en qué valor binario (dependiendo la cantidad de bits) se convertirá el valor muestreado y por último el Coder toma el string de bits y lo convierte a un sistema o formato binario que utilicemos (int, float, etc).

Cuantizador

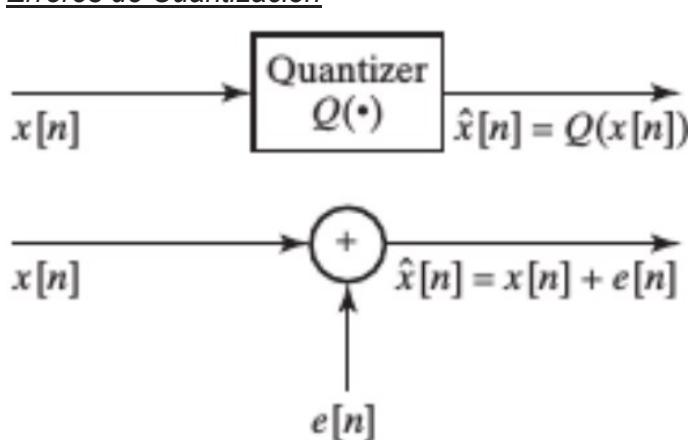


El cuantizador, en este caso, vemos que está espaciado uniformemente, la llamamos $Q(x)$ espaciado delta. Delta se define como el rango máximo de tensión dividido la cantidad posible de números representables con los bits, se expresa en mV normalmente.

$$\Delta = \frac{\text{full voltage range}}{2^{\text{word length}}} = \frac{2V_p}{2^B} . [mV]$$

Vemos que para varios valores de tensión, todo lo que caiga en un intervalo toma el mismo valor. El trabajo del Coder es tomar los bits que genera el quantizer o cuantizador y los transforma a un formato específico. La tabla pasa de binario común a, por ejemplo, complemento a 2.

Errores de Cuantización



La señal cuando sale del cuantizador está determinada por valores fijos que no son exactamente los que tiene la señal. Tenemos que hay un delta entre valor y valor posible. Si hacemos la diferencia entre el valor posible y el valor real tenemos el error de cuantización.

Podemos concluir que al cuantizar generamos ruido, ya que este error de cuantización genera errores en la señal obtenida.

Modelizamos el error de cuantizador esto es como sumarle a la señal original una señal de ruido.

Relación SNR en ADC

Vemos que en vez de utilizar el nombre delta para el salto de cuantizador lo llamamos q . Es lo mismo que definimos antes, rango dividido cantidad de valores posibles a representar.

$$\Delta = q = \frac{\text{full voltage range}}{2^{\text{word length}}} = \frac{2V_p}{2^B}$$

Buscamos definir una relación señal a ruido.

Es como una caja que al entrar una señal limpia sale una señal con ruido y buscamos cual es la relación entre ambas. Como la generación de ruido es aleatoria, buscamos definir estadísticamente. Esto lo hacemos con varianzas, hacemos $10\log$ de la varianza de la señal / la varianza del ruido de la salida.

$$SNR_{ADC} = 10 \cdot \log_{10} \left(\frac{\text{input signal variance}}{\text{A/D quantization noise variance}} \right), \quad [\text{dB}]$$

$$= 10 \cdot \log_{10} \left(\frac{\sigma_{signal}^2}{\sigma_{ADC}^2} \right).$$

Definimos el valor de la varianza del error. Lo hacemos con un modelo, decimos que es uniforme el ruido. Obtenemos finalmente una fórmula que determina la varianza del ruido de cuantización en función del valor de tensión al cuadrado y la cantidad de bits que utilizamos. Esta es la varianza del ruido que se inyecta a la señal.

$$\sigma_{ADC}^2 = \left(\frac{2V_p}{2^B} \right)^2 \cdot \frac{1}{12} = \boxed{\frac{V_p^2}{3 \cdot 2^{2B}}}$$

El factor de carga es el valor eficaz de potencia de la señal dividido el valor pico de la señal. De acá podemos despejar la varianza de la señal que queda en función del factor de carga al cuadrado y el valor pico al cuadrado

$$\text{Load Factor, } LF = \frac{rms_{signal}}{V_p} = \frac{\sigma_{signal}}{V_p} \implies \sigma_{signal}^2 = \boxed{LF^2 \cdot V_p^2}$$

Por último obtenemos una función que permite calcular SNR en función del factor de carga y la cantidad de bits del cuantificador:

$$SNR_{ADC} = 10 \cdot \log_{10} \left(\frac{\sigma_{signal}^2}{\sigma_{ADC}^2} \right) = 20 \cdot \log_{10}(LF) + 4.77 + 6.02 \cdot B. \quad [\text{dB}]$$

Consideraciones

Para mejorar la digitalización buscamos tener una SNR muy grande de modo que el ruido sea bajo a la par de la señal. Lo que vemos en la expresión es que depende del factor de carga y los bits que se pueden utilizar.

$$SNR_{ADC} = 20 \cdot \log_{10}(LF) + 4.77 + 6.02 \cdot B, \quad [\text{dB}]$$

$$= 20 \cdot \log_{10} \left(\frac{rms_{signal}}{V_n} \right) + 4.77 + 6.02 \cdot B. \quad [\text{dB}]$$

Si queremos modificar el factor de carga no es posible siempre ya que no podemos modificar mucho la potencia con la que alimentamos el conversor. Recordemos que si aumentamos la potencia, en cierta manera aumentamos la tensión y el valor tiende a ser mayor, manteniendo una LF constante. Igualmente

nos damos cuenta que mientras más chica es la potencia con la que alimentamos el conversor, peor es ya que la SNR disminuye, siempre tenemos que alimentarlo con la mayor tensión que permita para evitar que el SNR decaiga.

Vemos que el SNR depende también de la cantidad de bits que utilizamos, de modo que este tiene multiplicando 6, por cada bit que aumentemos, aumentamos en 6dB nuestro SNR.

Hay que tener en cuenta que no siempre estamos trabajando con los picos de tensiones, por lo que tenemos que intentar hacer que sea la mayor posible. Como el modelo de densidad de error no es tan uniforme como lo modelizamos, se toma un margen de error disminuyendo la relación señal a ruido entre 3 y 6 dB para que sea más real.

SNR para Senoidal Pura

$$\begin{aligned} SNR_{ADC} &= 20 \cdot \log_{10} \left(\frac{rms_{signal}}{V_p} \right) + 4.77 + 6.02 \cdot B \\ &= 20 \cdot \log_{10} \left(\frac{V_p/\sqrt{2}}{V_p} \right) + 4.77 + 6.02 \cdot B. \end{aligned}$$

$$\begin{aligned} SNR_{ADC} &= 20 \cdot \log_{10} \left(1/\sqrt{2} \right) + 4.77 + 6.02 \cdot B \\ &= -3.01 + 4.77 + 6.02 \cdot B, \\ &= 1.76 + 6.02 \cdot B. \quad [\text{dB}] \end{aligned}$$

Reemplazando rmssignal por el valor de tensión correspondiente. Observando detenidamente nos damos cuenta que el aporte de LF es negativo porque nos queda un valor menor a 1 dentro del logaritmo y finalmente obtenemos una ecuación fácil de utilizar en donde se simplifican los valores y todo depende de la cantidad de bits que utilicemos. ESTO ES VÁLIDO SOLO PARA SEÑALES SENOIALES.

SNR para señal genérica

En este ejemplo tomamos un conversor de audio de 24 bits, y buscamos obtener al menos una salida de audio con un SNR de 110dB. Vemos que al reemplazar los colores en SNRadc le restamos 3dB para seguridad.

Si tomamos el valor obtenido, le restamos 110dB y lo dividimos por 6 (la constante que acompaña la variable "cantidad de bits") obtenemos que la cantidad de bits necesaria es 5,5 o mejor dicho, 6 bits, nos damos cuenta que esto es lo que dedicamos a ver ruido. Esto lo utilizamos para darnos cuenta que debemos descartar esa cantidad de bits ya que en realidad no nos interesa ver ruido. La forma de calcular esto es $(SNR_{señal} - SNR_{adc})/6,02=B$ (bits)

Por otro lado, vemos que si utilizamos 10 bits en vez de 24, cuando calculamos la cantidad de bits vemos que nos faltan al menos 8 bits para que funcione.

Lo que obtenemos como resultado es que siempre debemos tener un SNRadc mayor que el SNR de la señal, esto lo hacemos modificando la cantidad de bits.

Una regla es que siempre dejemos 6 dB de más para "salvaguardar" que lo que hemos muestreado cumpla con el SNR mínimo.

Cómo podemos saber cuántos bits utilizamos para ver ruido, los debemos eliminar, esto o hacemos con el código en c que hace un desplazamiento a la derecha de nuestro valor leído, esto es eliminar los bits que están por debajo de los bits significativos de nuestra señal, tiramos tantos bits como sea necesario ya que no queremos tener ruido.

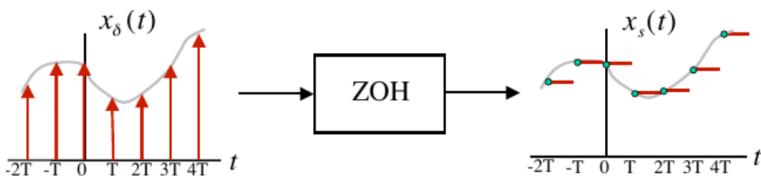
Conversor DAC

El DAC hace lo contrario del ADC, pasa de binario a un valor de voltaje. Lo que define a un DAC es:

- Resolución: se especifica a través de los números de bits, a mayor cantidad de bits, mayor resolución y puede representar la señal de mejor forma con menores saltos.
- Frecuencia de muestreo: la salida de este está definida por un tren de pulsos, puede ser igual a la frecuencia del conversor ADC o puede tener otro valor, a veces conviene que sea mayor que el ADC.

Conversor IDEAL

Convierte binario en un tren de impulsos, este tren contiene también el espectro de salida repetido, osea tiene aliasing, por lo que se pone un filtro anti alias. Hacer impulsos es imposible, por lo que se agrega un ZOH (Zero Order Holder). Este recibe el tren de impulsos y lo transforma en un tren de escalones. El tiempo de sostenimiento depende de la frecuencia de muestreo. Básicamente interpola.



Respuesta en Frecuencia del ZOH

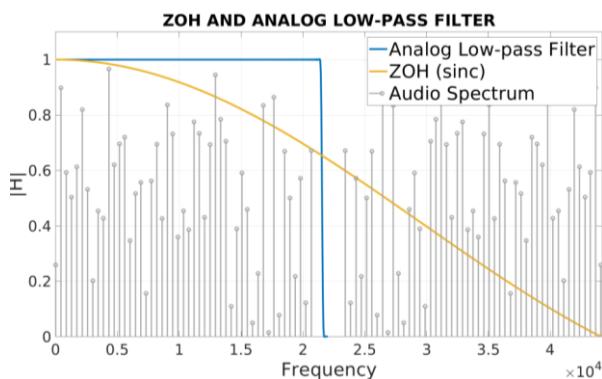
$$H_{ZOH}(f) = \frac{\sin(\pi f/f_s)}{(\pi f/f_s)} = \text{sinc}(\pi f/f_s)$$

Vemos que el H es la transformada de fourier del pulso. A la salida obtenemos un seno cardinal que depende de la frecuencia de muestreo.

El ZOH hace, matemáticamente, la convolución del tren de impulsos con un pulso rectangular cuya frecuencia depende de la frecuencia de muestreo. El ZOH, que funciona como si fuera un MAL filtro pasabajos, no corta en la frecuencia máxima que tendría que cortar para que solo quede el espectro útil. Porque a la salida se siguen teniendo los espectros aliados.

Esto implica que a la salida debemos agregar un filtro analógico estándar. Recordemos que a la salida tenemos una señal ANALÓGICA escalonada (ya no es un tren de impulsos), por lo que al agregar un filtro vemos que corta luego de la frecuencia máxima y antes del espectro espejado, tomando únicamente el espectro útil. Vemos que este filtro pasabajos elimina altas frecuencias, recordemos que los escalones tienen alta frecuencia, esto quiere decir que este filtro suaviza la señal, le saca las altas frecuencias.

Como resultado, la cadena de bloques nos queda que el DAC nos da un tren de impulsos, el segundo bloque nos da una señal escalera y el tercero nos da un filtro pasabajos; la convolución de estos 3 nos da la multiplicación en frecuencia. Entonces al multiplicar los 3 obtengo una respuesta efectiva.



El ZOH produce una atenuación de -3dB en la frecuencia de corte, que la compensamos ecualizando la señal de salida de 3 formas:

1. Ignorar el problema cuando no nos afecta
2. Pre-ecualizar en software
3. Post-ecualizar en el filtro pasabajos (ahora tiene que ser activo y con rta en frecuencia contraria al ZOH)
4. Oversampling: aumentar la frecuencia de muestreo del DAC para que la caída del ZOH sea constante dentro del BW del pasabajos de salida. Típicamente se hace 8 veces.

Filtros

Cuando filtramos en tiempo queremos eliminar el ruido, suavizamos la señal; si queremos eliminar o separar frecuencias, filtramos en frecuencia. Depende de donde tenemos la información, es donde filtraremos. Los filtros en el tiempo solo se pueden hacer digitalmente

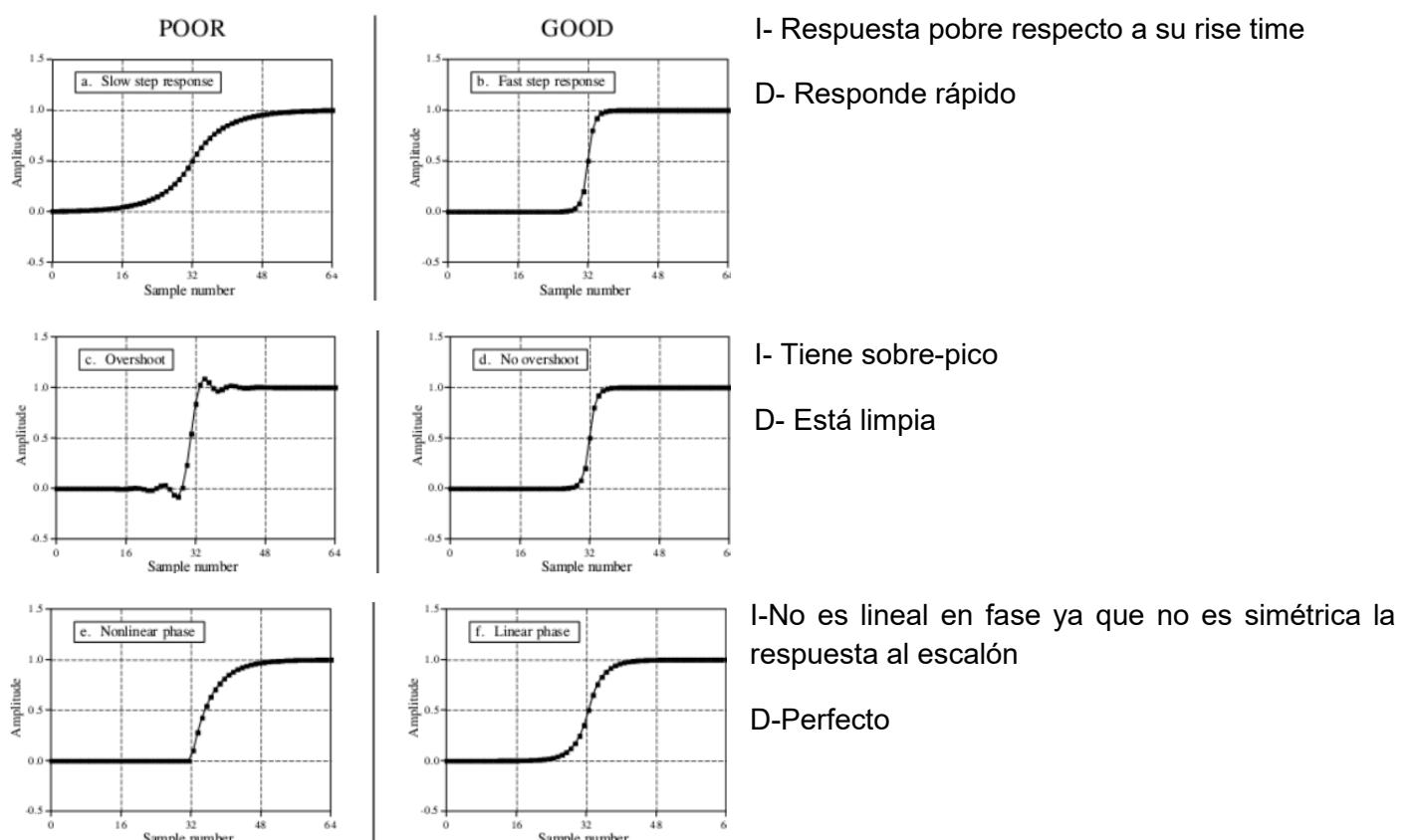
	Finite impulse response (FIR)	Infinite impulse response (IIR)
Filtering in time domain	Moving average	Leaky Integrator
Filtering in frequency domain	Windowed Filters Equiripple Minimax	Bilinear z-transform

FIR: Ventana Móvil

Filtran en el tiempo. La información está en el tiempo, no en frecuencia. Cada muestra tiene una amplitud y tiempo asignados. Suavizan una señal con ruido superpuesto en el tiempo.

Rta al Escalón

Un buen filtro en el dominio de tiempo es un pobre filtro en frecuencia y viceversa. Es decir que con un solo filtro no se puede resolver todo; se debe filtrar en ambos.



Definición

Se define como: Convolución entre una señal de entrada y un pulso rectangular de área 1. También se lo conoce como filtro de promedio local. Produce un retraso de N/2 muestras entre la entrada y la salida. Es de respuesta FINITA porque no realimenta la salida. La salida solo depende de valores pasado y actual de la entrada.

$$h[n] = \frac{1}{N} \sum_{k=0}^{N-1} \delta[n - k]$$

h[n] es el kernel del filtro.

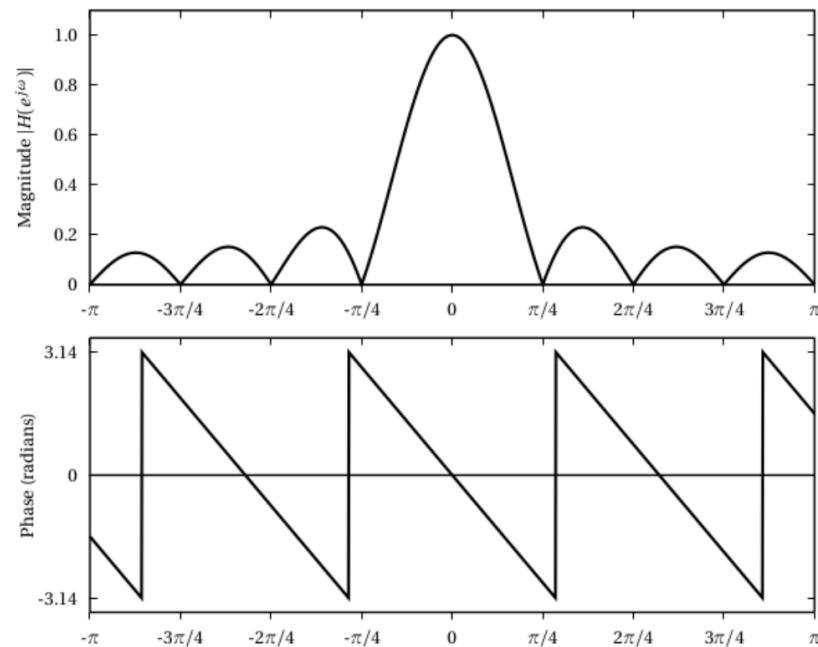
$$y[n] = x[n] * h[n] = \frac{1}{N} \sum_{k=0}^{N-1} x[n-k]$$

El kernel es un vector de N elementos que contiene los coeficientes del filtro.

Este filtro promedia los valores dentro de la ventana, permite reducir niveles de ruido pq las componentes en amplitud se cancelan entre sí.

Si es de orden muy grande puede eliminar información de la señal.

Respuesta en Frecuencia

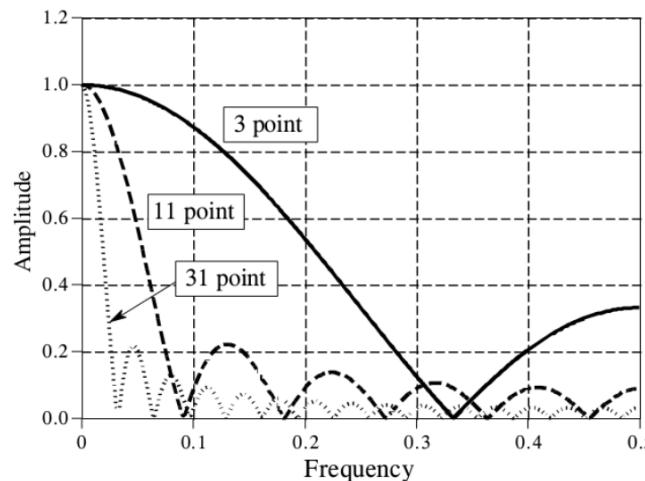


Como el filtro moving average, en la frecuencia, es un cuadrado en el dominio del tiempo, en la frecuencia, tiene el formato de un seno cardinal. Y pasa viceversa cuando lo hacemos en frecuencia y miramos el tiempo.

Este filtro sería como un mal filtro pasabajo ya que en la banda pasante no es plano. Esto tiene sentido que ya que el ruido es de alta frecuencia, esto es, eliminamos el ruido con un pasabajos.

La respuesta en FASE es lineal, característica típica del tipo FIR (en banda pasante)

Influencia del Orden del Filtro en la RTA en FREC.



El eje de frecuencia está normalizado según la frecuencia de muestreo, llega a 0,5 ya que tiene que ser la mitad al representar la respuesta en frecuencia por el desdoblamiento de la respuesta original. A medida que aumenta la frecuencia del filtro, se hace más selectiva. A medida que aumentamos el orden, tiende a quedarse con las frecuencias continuas. Debemos buscar el N máximo que filtre solo ruido no frecuencias que necesitamos en la información o señal de interés

$$|H[f]| = \frac{1}{N} \left| \frac{\sin(\pi \cdot f \cdot N)}{\sin(\pi \cdot f)} \right|$$

Cálculo del Orden del Filtro

$$0.707 = \frac{1}{N} \left| \frac{\sin(\pi \cdot f_{co} \cdot N)}{\sin(\pi \cdot f_{co})} \right|$$

Buscamos la respuesta en frecuencia de nuestro filtro. Reemplazamos el valor de corte y la frecuencia de corte, a -3dB. La frecuencia normalizada es la frecuencia de corte dividida la frecuencia de muestreo. Encontramos la frecuencia de corte y despejamos N.

Este N genera una respuesta en frecuencia que filtrará frecuencias superiores a la frecuencia de corte seleccionada. De esta manera nos aseguramos que no nos elimine información

$$N_{max} = \text{round} \left(\sqrt{\frac{0.885894^2 \cdot f_s^2}{f_{co}^2} - 1} \right)$$

FIR: Ventaneo

Filtran en la frecuencia. La información en la frecuencia tiene:

- Banda Pasante.
- Banda Atenuada.
- Banda de Transición.
- Frecuencia de corte: donde cae -3dB.
- Ripple en banda pasante.

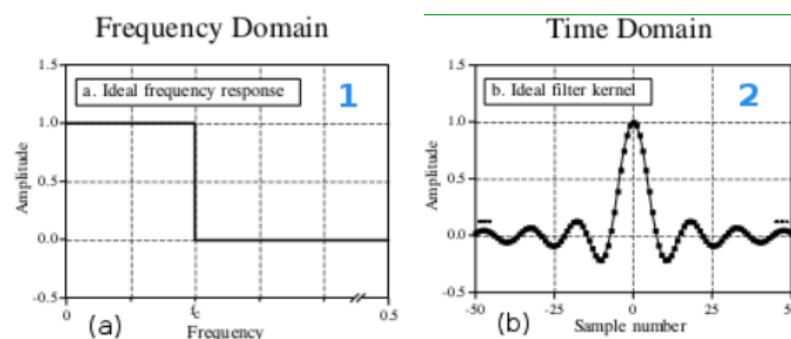
Una buena respuesta en freq se define por:

- Banda de Transición Corta.
- Bajo Ripple: idealmente cero.
- Alta atenuación en banda atenuada.
- Bajo ripple en banda de transición.

Se requieren varias muestras para hacer el análisis en freq.

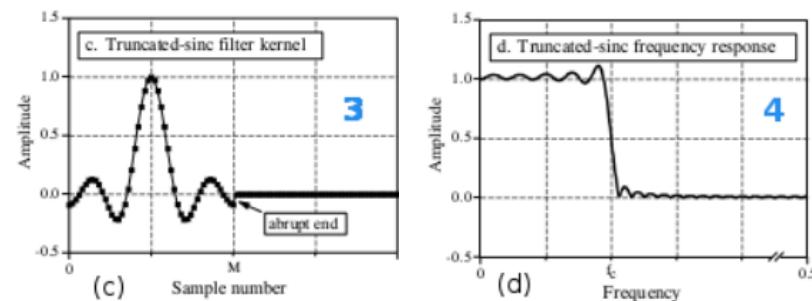
Estrategia de Filtrado por Seno Cardinal Ventaneando

1. La respuesta de un pasabajos ideal en la freq es un seno cardinal infinito en el tiempo. No se puede implementar. Produce un kernel de infinitos términos.



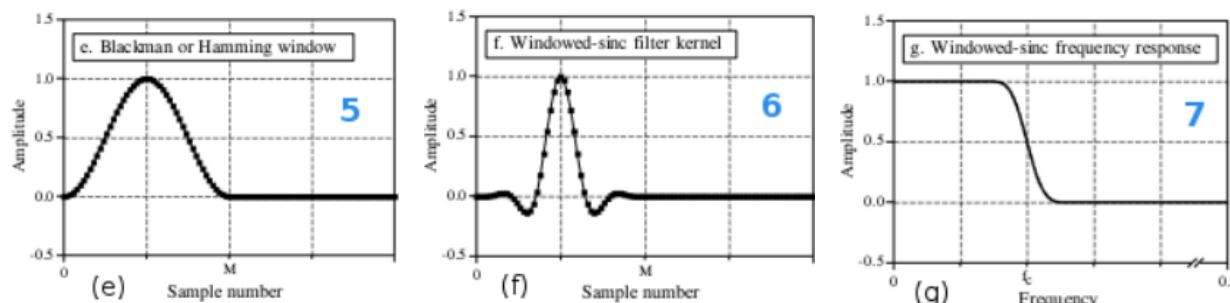
$$h_s[n] = \frac{\sin(\pi f[n]/f_s)}{(\pi f[n]/f_s)}$$

2. Se truncan los términos del seno cardinal, eligiendo M elementos de este. M tiene que ser PAR. Al truncar en el tiempo, deformamos la respuesta en frecuencia, aparece el fenómeno de GIBBS.



3. Para mejorar la respuesta en frecuencia del filtro truncado, se multiplica cada elemento del kernel del filtro por una ventana de elementos en el tiempo (FIGURA 5). Es un vector con coeficientes que al multiplicar elemento a elemento por la respuesta en el tiempo obtenemos la FIGURA 6 en el tiempo.

Dando como resultado EN FRECUENCIA LA FIGURA 7.



Al eliminar el corte abrupto del seno cardinal, tenemos que la respuesta en frecuencia perdió un poco la calidad en la banda de transición, pero no hay ripple y tiene elementos finitos, por lo que se puede aplicar en una computadora. Comparado con la figura 4 hay una gran mejora.

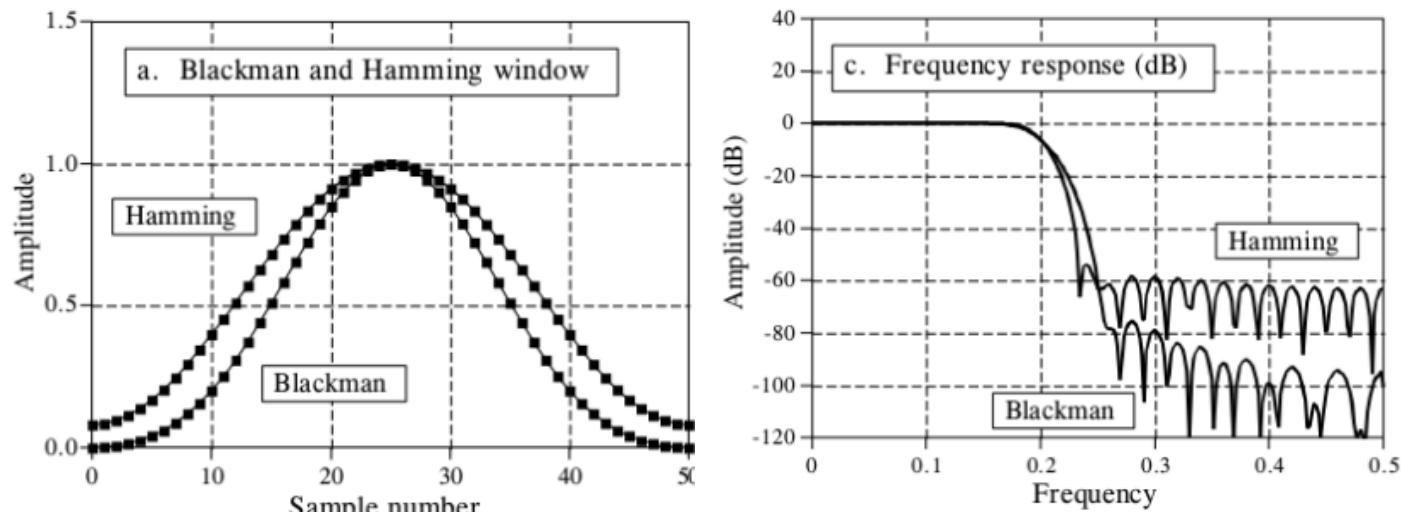
Matemáticamente: Tenemos que se multiplica la respuesta del filtro truncado (kernel del filtro) por la ventana, nos da una nueva respuesta ventaneada. Luego con los coeficientes de este filtro, se realiza la convolución con los elementos de la señal de entrada.

$$h_w[n] = h_{st}[n] \cdot w[n]$$

$$y[n] = h_w[n] * x[n]$$

Diferencias entre las ventanas de HAMMING y BLACKMAN

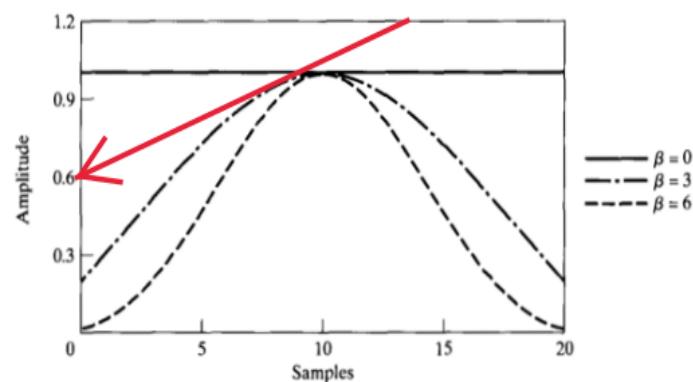
[blackman la tiene más grande.. por nekro]



Cada ventana tiene 51 elementos. La ventana de Hamming tiene una transición más angosta (20%) que la blackman. Para la atenuación de Hamming, tenemos una atenuación de -53 dB para el primer lóbulo y para blackman -73 dB del primer lóbulo de transición.

La relación de compromiso es: Hamming tiene una banda de transición más estrecha pero ofrece menos atenuación en la banda atenuada y en el caso de Blackman tiene banda de transición más ancha pero en la banda de atenuación está más atenuada.

En general se parte eligiendo Blackman, ya que el ancho de la banda de transición se achica aumentando el orden del filtro. Obteniendo como resultado un filtro con gran atenuación y banda de transición estrecha.



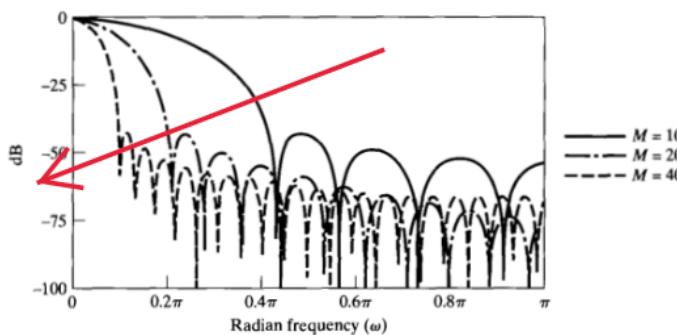
Ventana de Kaiser

La ventana de kaiser es una ventana de ventanas. Esta se configura eligiendo el orden de la ventana y a demás el valor de un parámetro beta. Modificando el beta, pasamos por varios tipos de ventanas estándar, beta:

- $\beta=0$ (rectángulo): el seno cardinal infinito se multiplica por una ventana rectangular para truncar (lo

que vimos al principio de truncar la cantidad de elementos)

- $b=3$ pasa por ventana de Hamming



- $b=6$ ventana de Blackman
- Podemos elegir una ventana que está entre las dos
- Si aumenta beta, aumenta la atenuación del filtro.

Al variar M , varía la frecuencia de corte del filtro en relación con la frecuencia de muestreo.

Comparación

Name of window function $w[n]$	Transition width ΔF in (Hz), (normalised)	Pass-band ripple A_p in (dB)	Ripple δ_p, δ_s	Side-lobe level in (dB)	Stop-band attenuation A_s in (dB)
Rectangular	$0.9/N$	0.741	0.089	-13	21
Hanning	$3.1/N$	0.0546	0.063	-31	44
Hamming	$3.3/N$	0.0194	0.0022	-41	53
Blackman	$5.5/N$	0.0017	0.000196	-57	74
Kaiser $\beta=4.54$	$2.93/N$	0.0274			50
$\beta=5.65$	$3.63/N$	0.00867			60
$\beta=6.76$	$4.32/N$	0.00275			70
$\beta=8.96$	$5.71/N$	0.000275			90

Ventana rectangular tiene la banda de transición más angosta, a medida que cambiamos de ventana, esta va aumentando. Blackman es la peor.

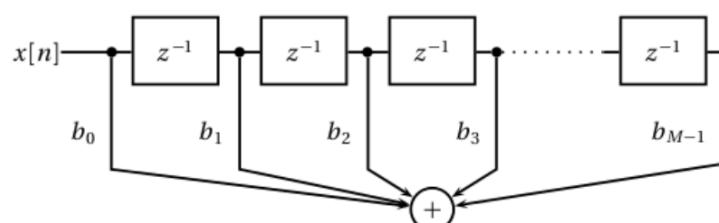
La rectangular es la que mayor ripple tiene, a medida que avanzamos en la tabla, disminuyen los ripple. ESTA DADO EN dB. Blackman es la mejor.

La rectangular es la que tiene el lóbulo más alto.

Estructuras de Implementación de FIR

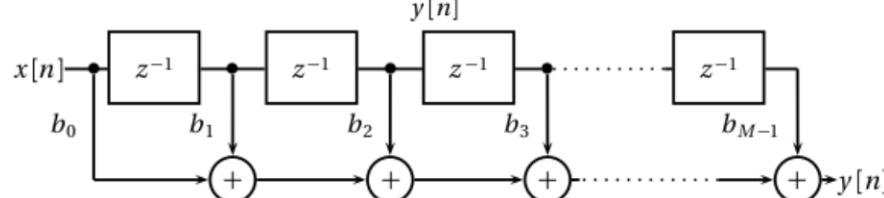
Respuesta al impulso de un filtro FIR. Está expresado con la transformada Z. Los coeficientes representan los coeficientes del filtro FIR y los Z^{-k} representan los retrasos pertinentes. Este filtro no tiene denominador, esto es, no tiene realimentación; esto quiere decir que no tiene polos. Al no tener polos, el filtro es incondicionalmente estable.

$$H(z) = b_0 + b_1 z^{-1} + \dots + b_{M-1} z^{M-1}$$



Cada bloque aplica un rechazo, multiplica por las constantes y se suma. Vemos que no hay realimentación desde la salida a la entrada. Esta es la forma directa del filtro FIR.

Es la implementación transversal, es lo que se hace con la operación MAC. Este tiene una diferencia, se hacen multiplicaciones y sumas parciales.



⇒ ESTA ES LA REAL QUE SE HACE CON UNA OPERACIÓN MAC.

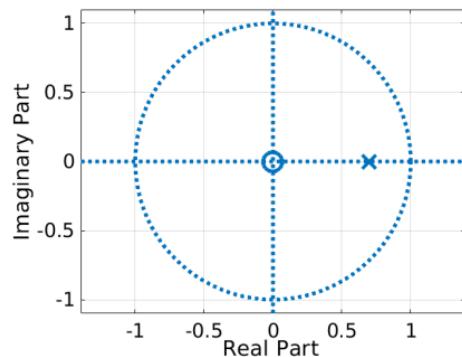
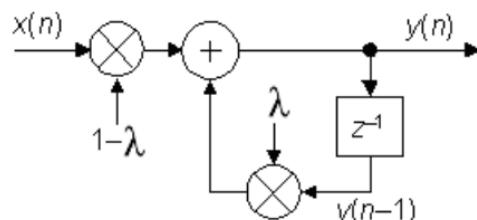
IIR Leaky Integrator

Se obtiene a partir del filtro FIR MOVING AVERAGE.

$$y[n] = \lambda y[n-1] + (1 - \lambda) x[n] \quad \lambda = \frac{M-1}{M}$$

Ya no es una convolución, es una ecuación diferencial de coeficientes constantes. Hay que fijar la condición inicial (el valor del $y[n-1]$). Normalmente el valor del pasado se coloca igual a cero. Se lo conoce como un filtro recursivo de polo simple. Es un sistema lineal invariante en el tiempo. Si lambda es menor que 1 es estable (En principio no tendría problema de estabilidad ya que es <1). El valor de lambda da que tanto se va a filtrar la señal, cuanto se va a suavizar.

$$\frac{Y(z)}{X(z)} = \frac{1 - \lambda}{1 - \lambda z^{-1}}$$

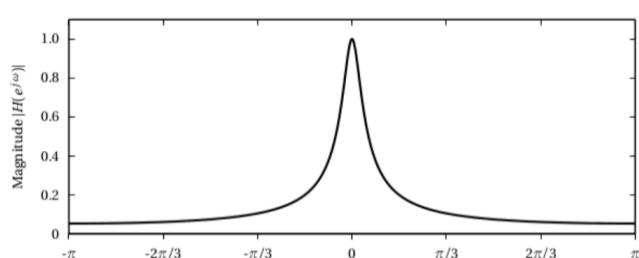


Aplicando transformada z, podemos despejar la señal de salida. En la representación del sistema muestra una realimentación.

El cero siempre estará cercano a cero y el polo siempre estará dentro de la circunferencia unitaria. Cuando nos acercamos a la circunferencia de radio 1 puede haber inestabilidad y por un error de redondeo puede quedar fuera de la circunferencia este polo y generar problemas. Esta inestabilidad sería esporádica e intermitente. Los problemas al acercar lambda a 1 se dan por error de redondeo o ruido ya que puede hacer que ocurra que se vuelva inestable al valer 1 o más.

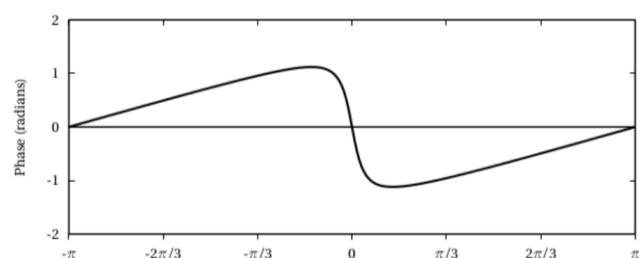
Aumentar el lambda mejora el suavizado pero produce más atraso.

Respuesta en Frecuencia



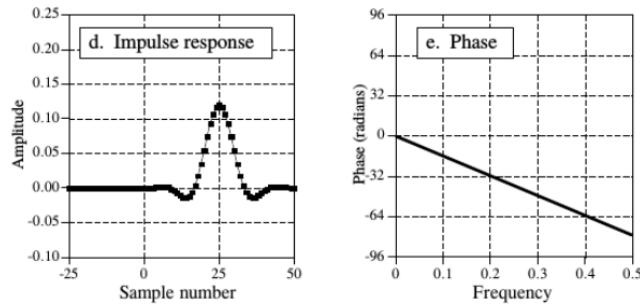
A medida que aumenta el valor de lambda se estrecha más y más la forma, este filtro pasa bajos es malo y se parece más a un selector de frecuencia.

No tiene respuesta lineal en fase en la banda pasante, no es una línea como veíamos en fir (parecía una triangular).



Comparación con el FIR

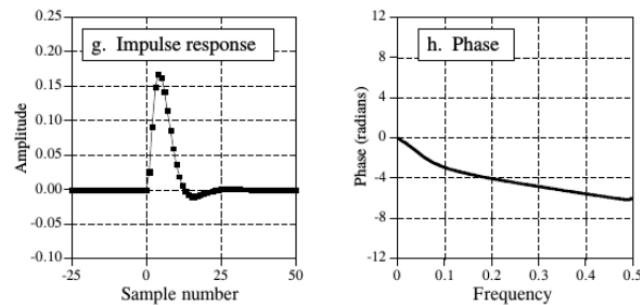
Linear Phase Filter



Si aumenta la frecuencia, aumenta el desfase LINEALMENTE. Vemos que es lineal, no tiene curvas.

Tenemos que el pulso que ha pasado por el filtro, se filtraron algunas frecuencias y lo que observamos es que es SIMÉTRICO.

Nonlinear Phase Filter



Ya no es un seno cardinal como antes, es otra cosa.

En fase ya no es lineal. Por esto, no es simétrico, el flanco de subida es diferente al flanco de bajada.

[CONCLUSIÓN ES UNA PORONGA]

IIR: Transformada Bilineal (Método de Tustin) [un mostro el tustin este]

La idea es tomar un filtro del dominio analógico y llevarlo al dominio digital, pasar de la variable "s" a la "z". Digitalizamos los filtros conocidos. Tienen diferentes características dependiendo el ripple y la banda de atenuación.

$$s \approx \frac{2}{T} \left(\frac{1 - z^{-1}}{1 + z^{-1}} \right)$$

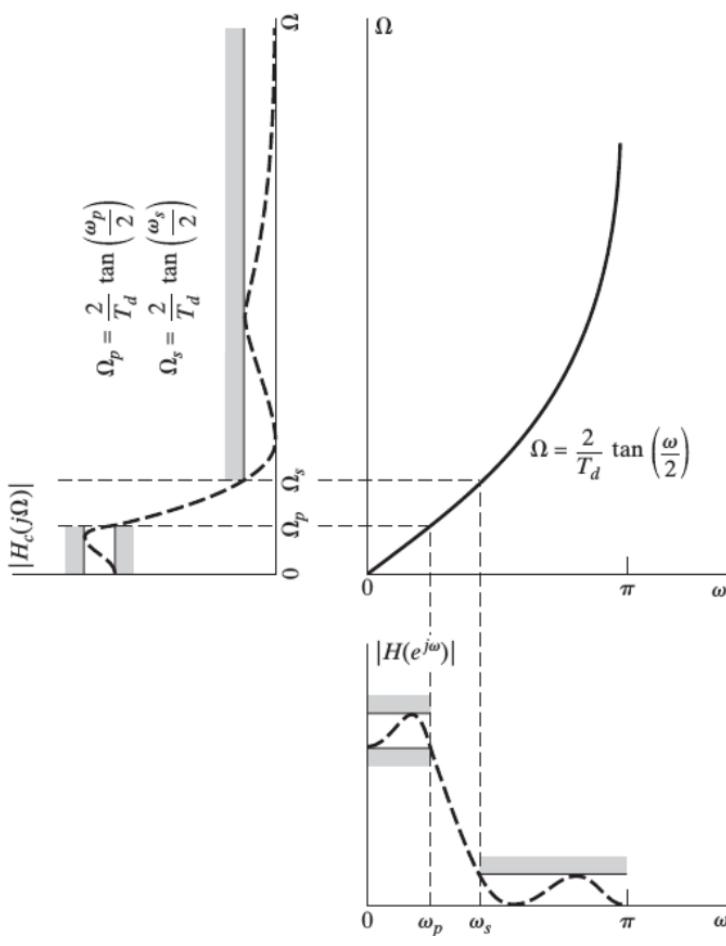
Mediante la transformada bilineal convertimos la función de transferencia de un filtro analógico al dominio digital Z. Reemplazando la S en la H(s) y haciendo matemagias algebraicas para llegar a una ecuación en diferencias con los coeficientes del filtro.

Relación entre las frecuencias analógicas y digitales

$\Omega = \frac{2}{T} \tan(\omega/2)$, $\omega = \arctan(\Omega T/2)$

Omega mayúscula es la f analógica, va de un inf al otro. La Omega minúscula va de -pi a pi, esto se define así porque cuando estamos en el dominio digital, podemos ver una frecuencia máxima, que es la mitad de la frecuencia mínima de muestreo (Nyquist), por eso no va de -inf a +inf. La relación no es lineal ya que tenemos la tangente como la función que determina la relación entre las frecuencias.

Precomulado



Para obtener la misma frecuencia del principio se le hace una operación llamada prewarping o precomulado, lo que se hace es elegir una frecuencia de corte para un filtro analógico, tal que cuando este filtro sea digitalizado con la transformada bilineal, de modo que obtengamos en el dominio discreto la frecuencia que necesitamos para este dominio en una aplicación DIGITAL.

Plantilla de filtro digital, luego la proyectamos hacia arriba con la relación que mantienen las variables, en el dominio analógico se proyectan estas frecuencias y vemos que las proyecciones se corren de los valores que teníamos en el principio.

Pasos de Diseño

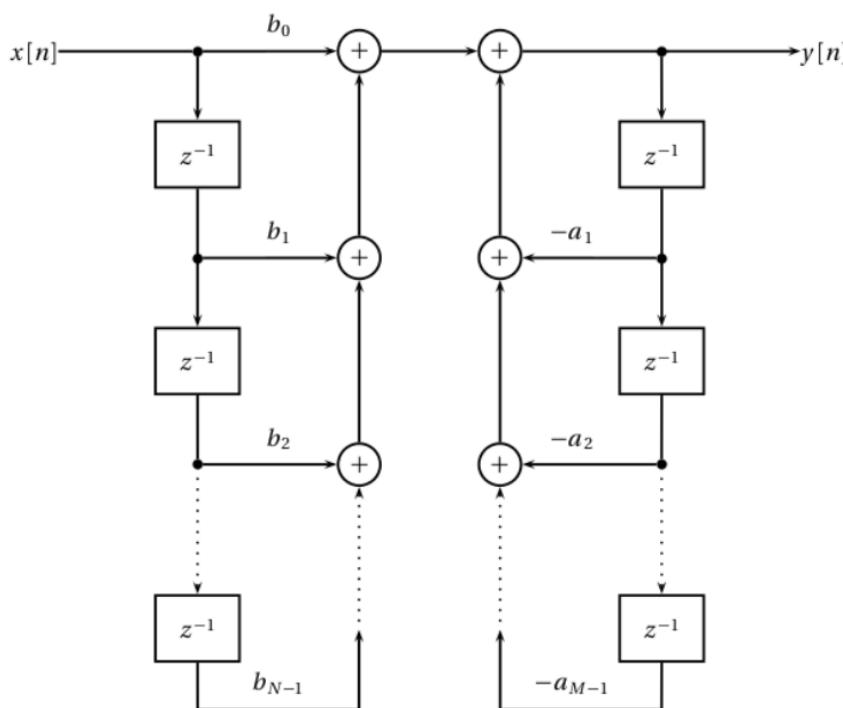
1. Elegir el filtro analógico
2. Normalizar frecuencia de corte
3. Precombar frecuencias analógicas normalizadas
4. Reemplazar S con la transformada lineal
5. Morir algebraicamente
6. Encontrar la ecuación en diferencias final y despejar $y[n]$.

7.

Formas de implementar [dudo que pregunte esto, es una poronga]

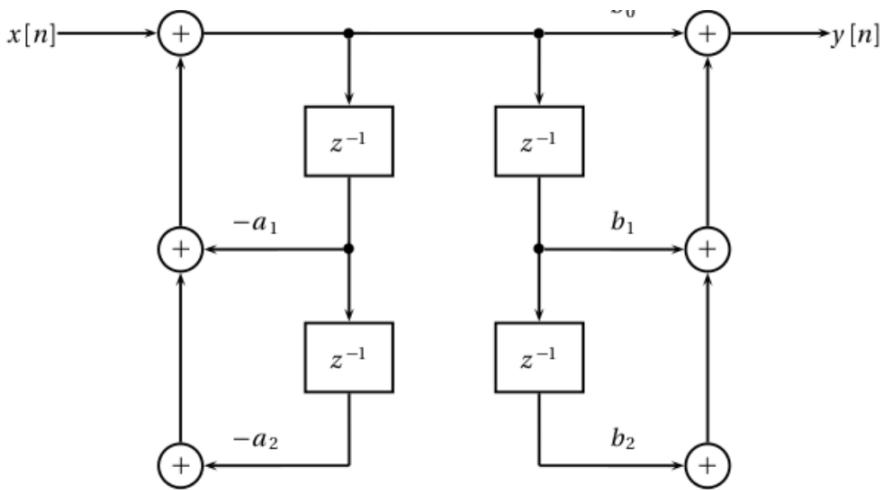
Forma directa 1

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_{N-1} z^{N-1}}{1 + a_1 z^{-1} + \dots + a_{M-1} z^{M-1}}$$



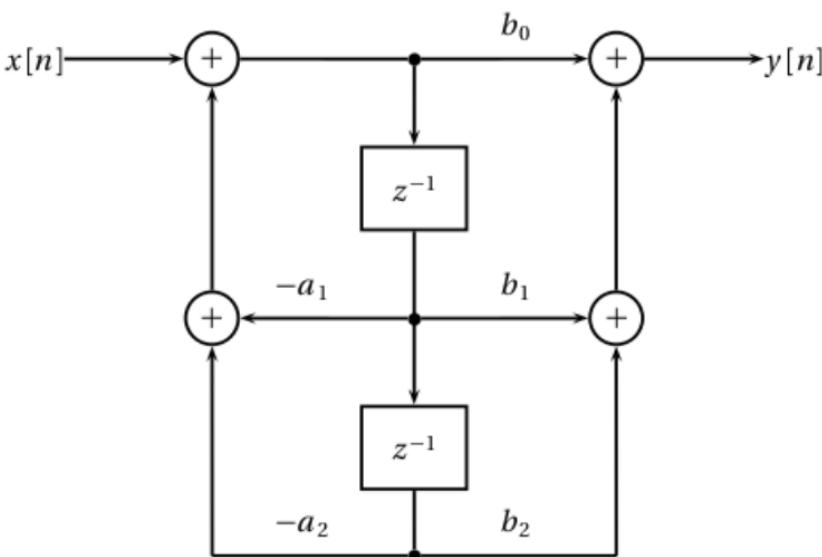
Forma directa 1: z^{-1} representa un retardo y también una variable, ya que este valor tiene que guardarse para computarse en próximas operaciones; esto es válido tanto para la entrada como para la salida, serían los $x[n-j]$ y $y[n-k]$ que pueden aparecer.

Forma directa 1 (implementación invertida)



Por la propiedad conmutativa de la transformada z , se pasaron los bloques de la entrada a la salida y los de la salida a la entrada. Matemáticamente no se alteró el sistema

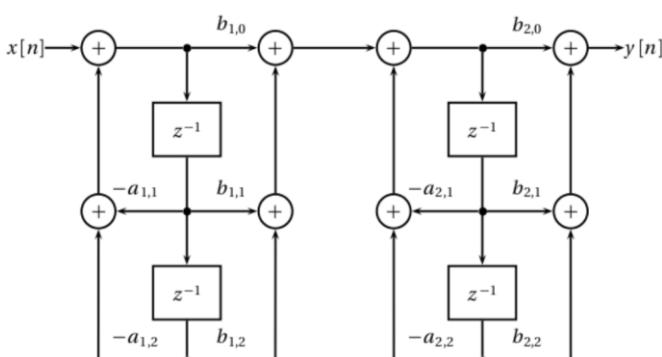
Forma directa 2



Podemos unir los bloques enfrentados de z^{-1} , teniendo así solo una variable cada dos bloques. Esto es importante porque siempre es más conveniente representar el filtro IRR con diferentes bloques.

Implementación en Cascada

$$H(z) = \prod_{k=1}^N G_k \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$



Esta representación de bloques permite hacer la cascada de bloques, se expresan con una productoria. G_k es la ganancia que está afectando a cada bloque doble. Esto se hace para disminuir la sensibilidad de cuantificación.

[coso]	FIR	IIR
PROS	<ul style="list-style-type: none"> • Es siempre estable (no tiene polos). • Tiene un control preciso de la fase, es lineal. • Es robusto para expresarlo con precisión finita. 	<ul style="list-style-type: none"> • Tiene menor costo computacional para una misma aplicación respecto del FIR. • Menor delay entre la salida y la entrada. • Es más compacto.
CONS	<ul style="list-style-type: none"> • Tiene mucho delay entre la salida y la entrada. • Tiene costos altos para una misma solución respecto del IRR. 	<ul style="list-style-type: none"> • No tiene una estabilidad garantizada. • La fase es difícil de controlar. • Es difícil de diseñar. • Es sensible a la precisión numérica.

U3: Introducción a los sistemas de Comunicaciones de Datos [prepare el ocotel]

[todo lo que sigue lo escribió el fede, para él, es la definición en vida de “mucho texto”, mejor leer del otro resumen... bah, es la misma poronga por donde lo quieras ver]

Introducción, Modelos de Referencia de Redes, Capa Física

[la capa física no la dieron en el 2020, asíque no estará en este resumen]

Hardware de Red

Se divide por tecnología de transmisión y por escala

Tecnología de transmisión:

Punto a Punto (conecta pares individuales de máquinas para intercambiar paquetes, 1 a 1 es **unicasting**) y red de difusión (llamada **broadcast**) donde todos comparten el mismo canal, pero con la dirección a la que se dirige el paquete se discrimina si toma o no el mismo, se puede hacer broadcasting para enviar a todos y **multicasting** para un grupo.

Escala:

Existen las redes de área personal, área local, metropolitana, amplia y la conexión de más redes son las interredes (un ejemplo es la Internet)

Distancia entre procesadores	Procesadores ubicados en el (la) mismo(a)	Ejemplo
1 m	Metro cuadrado	Red de área personal
10 m	Cuarto	
100 m	Edificio	Red de área local
1 km	Campus	
10 km	Ciudad	Red de área metropolitana
100 km	País	
1000 km	Continente	Red de área amplia
10 000 km	Planeta	Internet

Redes de área personal

Área Personal:

Llamadas **PAN** (*Personal Area Network*). Ejemplo: periféricos de una PC; es la comunicación dentro del rango de una persona, RFID, bluetooth, etc.

Área Local:

Llamadas LAN (*Local Area Networks*). Son redes de propiedad privada que operan dentro de un solo edificio, para conectar computadoras personales, electrodomésticos, redes empresariales. Cada computadora se comunica con un **AP** (*Access Point*), enrutador inalámbrico o estación base; hay un estándar para las redes LAN inalámbricas llamado IEEE 802.11, **WiFi**. Las redes **LAN** alámbricas utilizan distintas tecnologías de transmisión, como cables y fibra óptica. Tienen restricciones en cuanto a su tamaño, significa que el tiempo de transmisión en el peor de los casos es limitado y se sabe de antemano, cometan muy pocos errores. Está basada en los enlaces de punto a punto. Utilizan el estándar IEEE 802.3, comúnmente conocido como **Ethernet**. Un **switch** tiene varios puertos, cada uno de los cuales se puede conectar a una computadora. El trabajo del switch es transmitir paquetes entre las computadoras conectadas a él, y utiliza la dirección en cada paquete para determinar a qué computadora se lo debe enviar. Varias LAN se pueden conectar con switches entre sí. Es posible dividir una gran LAN física en dos redes LAN lógicas más pequeñas, denominada LAN virtual o **VLAN**. El diseño original de Ethernet es el que se difundían todos los paquetes a través de un solo cable lineal. Sólo una máquina podía transmitir con éxito en un instante dado, siempre que el cable estuviera inactivo. Si ocurría una colisión entre dos o más paquetes, cada computadora esperaba un tiempo aleatorio y volvía a intentar. Las redes inalámbricas y las alámbricas se pueden dividir en **diseños estáticos y dinámicos**, la estática típica sería dividir el tiempo en intervalos discretos, esta desperdicia la capacidad del canal cuando una máquina no tiene nada que decir, por lo que la mayoría de los sistemas tratan de asignar el canal en forma dinámica. Los métodos de asignación **dinámica** pueden ser **centralizados o descentralizados**; centralizado hay una sola entidad que determina el turno, podría aceptar varios paquetes y asignarles prioridades. En la asignación del canal descentralizado no hay una entidad central, cada máquina debe decidir por su cuenta si va a transmitir o no. Ejemplo: redes por el cableado eléctrico.

Redes de área metropolitana

Red de Área Metropolitana:

MAN (*Metropolitan Area Network*), cubre toda una ciudad, como las redes de televisión por cable. Ahora desarrollos en el acceso inalámbrico a Internet de alta velocidad, se estandarizó como IEEE 802.16 **WiMAX**.

Redes de área amplia

Red de Área Amplia:

WAN (*Wide Area Network*), implica un país o continente, ejemplo: una empresa con sucursales en distintas ciudades. Es una red que conecta las computadoras o hosts. Al resto de la red que conecta estos hosts se le denomina **subred**. La tarea de la subred es transportar los mensajes de host a host, cuenta con líneas de transmisión que mueven bits entre máquinas a partir de enlaces. Estas no poseen líneas de transmisión, tienen que rentarlas a una compañía. Los elementos de conmutación o switches son computadoras especializadas que conectan dos o más líneas de transmisión. Cuando los datos llegan por una línea entrante, se debe elegir una línea saliente hacia la cual reenviarlos, actualmente estos dispositivos se conocen como **enrutadores**. En una WAN, los hosts y la subred pertenecen a distintas personas, el proveedor de red opera la subred. Los enrutadores por lo general conectan distintos tipos de tecnología de red. Ejemplo Ethernet y SONET. De hecho, muchas redes WAN serán de hecho **interredes**. Una última diferencia está en lo que se conecta a la subred. Podrían ser computadoras individuales o podrían ser redes LAN completas. Hay dos variedades de redes WAN. En primer lugar, en vez de rentar líneas de transmisión dedicadas, una empresa podría conectar sus oficinas a Internet. Esto le permite hacer conexiones como enlaces virtuales, se le denomina **VPN** (*Red Privada Virtual*, del inglés *Virtual Private Network*), provee flexibilidad en la reutilización

de un recurso (Internet); pero la desventaja, es que carece de control sobre los recursos subyacentes y la capacidad puede variar según el servicio de Internet contratado. La segunda variación es que una empresa distinta puede operar la subred. Al operador de la subred se le conoce como **proveedor** de servicios de red. El operador de la subred también puede conectarse con otras redes que formen parte de Internet. A dicho operador de subred se le conoce como **ISP** (Proveedor de Servicios de Internet, del inglés *Internet Service Provider*) y la subred es una red ISP. Si dos enruteadores no comparten una línea, deben hacerlo en forma indirecta a través de otros enruteadores. Al proceso por el cual la red decide qué ruta tomar se le conoce como **algoritmo de enruteamiento**, es la decisión de hacia dónde debe enviar el paquete es el algoritmo de reenvío. Las WAN utilizan mucho las tecnologías inalámbricas como satélites y antenas. La red de telefonía celular es otro ejemplo de una WAN. Cada estación base cubre una distancia mucho mayor que una LAN inalámbrica. Las estaciones base se conectan entre sí mediante una red troncal.

Interredes

Hay personas conectadas a una red se quieren comunicar con las personas conectadas a una red distinta, estas son redes interconectadas y se le conoce como **interred** o **internet**. Internet usa redes de ISP para conectar muchos tipos. Una “subred” tiene más sentido en el contexto de una red WAN, se refiere a la colección de enruteadores y líneas de comunicación que pertenecen al operador de red. Los hosts no forman parte de la subred. Una **red** se forma al combinar una subred y sus hosts. Podríamos describir una subred como una red y una interred como una red. **Red: una colección de computadoras interconectadas mediante una sola tecnología. LA REGLA: Si varias organizaciones han pagado para construir distintas partes de la red y cada una se encarga de dar mantenimiento a la parte que le corresponde, entonces tenemos una interred en vez de una sola red; y si la tecnología subyacente es distinta en diferentes partes, es probable que sea una interred.** Una máquina que realiza una conexión entre dos o más redes y provee la traducción necesaria, tanto de hardware como software, es una **puerta de enlace (gateway)**. Se distinguen por la capa en la que operan. Formar una internet es para conectar computadoras entre distintas redes, no es conveniente usar una puerta de enlace de una capa demasiado baja, ya que no podremos realizar conexiones entre distintos tipos de redes. Tampoco demasiado alta, la conexión sólo funcionará para ciertas aplicaciones. La capa “ideal” se le denomina comúnmente capa de red; un enruteador es una puerta de enlace que commuta paquetes en la capa de red.

SOFTWARE DE RED

Jerarquías de protocolos

Para reducir la complejidad las redes se organizan como **capas** construida a partir de la que está abajo. El propósito de cada **capa** es ofrecer ciertos servicios a las capas superiores, mientras les oculta los detalles relacionados con la forma en que se implementan. Un **protocolo** es un acuerdo entre las partes que se comunican para establecer la forma en que se llevará a cabo esa comunicación. Las entidades que conforman las correspondientes capas en diferentes máquinas se llaman “iguales” (**peers**), pueden ser hardware o humanos. Los **peers** son los que se comunican a través del protocolo. Cada capa pasa los datos y la información de control a la capa inmediatamente inferior. Debajo de la **capa 1** se encuentra el medio físico a través del cual ocurre la comunicación real. Entre cada par de capas hay una interfaz. Define las operaciones y servicios primitivos que pone la capa más baja a disposición de la capa superior inmediata. Es necesario que desempeñe un conjunto específico de funciones bien entendidas. Simplifican el reemplazo de una capa con un protocolo. A un conjunto de capas y protocolos se le conoce como **arquitectura de red**, debe contener suficiente información como para permitir que un programador escriba el programa o construya el hardware para cada capa, de manera que se cumpla correctamente el protocolo apropiado. Ni los detalles de la implementación ni la especificación de las interfaces forman parte de la arquitectura. Ni siquiera es necesario que las interfaces en todas las máquinas de una red sean iguales, siempre y cuando cada máquina pueda utilizar todos los protocolos correctamente. La lista de los protocolos utilizados se le conoce como **pila de protocolos**. Un proceso de aplicación de la **capa 5** produce un mensaje M, y lo pasa a la **capa 4** para que lo transmita. La **capa 4** coloca un encabezado al frente del mensaje para identificarlo y pasa el resultado a la **capa 3**. El encabezado incluye información de control, como direcciones, para permitir que la **capa 4** en la máquina de destino entregue el mensaje; los tamaños y los tiempos, casi siempre tienen un límite impuesto por el protocolo de la **capa 3**. La **capa 3** debe descomponer los mensajes entrantes

en unidades más pequeñas llamadas **paquetes** y colocar un encabezado al frente de cada uno. M se divide en dos partes: M1 y M2, los cuales se transmitirán por separado. La **capa 3** decide cuál de las líneas salientes usar y pasa los paquetes a la **capa 2**; esta última agrega a cada pieza no sólo un encabezado, sino también un terminador (trailer), y pasa la unidad restante a la **capa 1** para su transmisión física. En la máquina receptora el mensaje pasa hacia arriba, de capa en capa, y los encabezados se van eliminando a medida que progresan. Los procesos de peers en la **capa 4** piensan conceptualmente en su comunicación como si fuera “horizontal”. La **abstracción** de los procesos de peers es imprescindible para todo diseño de red, se llama “**Software de red**”, se implementan con frecuencia en el hardware o firmware.

Aspectos de diseño para las capas

Los aspectos clave de diseño son:

Confiabilidad:

Enfocado en verificar que una red opere correctamente, utiliza códigos de detección de errores. Así, la información que se recibe de manera incorrecta puede re-transmitirse, corrección de errores en donde el mensaje correcto se recupera a partir de los bits posiblemente incorrectos. Se utilizan en capas bajas para proteger los **paquetes**, y en capas altas para verificar el contenido. También consiste en encontrar una ruta funcional a través de una red, se le conoce como **enrutamiento**.

Evolución de la red:

Emergen nuevos diseños que necesitan conectarse a la red existente. Se busca que se oculten los detalles de la implementación: distribución de protocolos en capas. El mecanismo para identificar los emisores y receptores, se conoce como **direcccionamiento**. Las distintas tecnologías de red a menudo tienen diferentes limitaciones, como orden de los mensajes que se envían, tamaño máximo, mecanismos para desensamblar, transmitir y después volver a ensamblar. Este tema se le conoce como **interconexión de redes (internetworking)**. Se busca que cuando las redes crecen, sigan funcionando bien, que sean **escalables y adaptables a salida de servicio de partes de la red**.

Asignación de recursos:

Como la capacidad de las líneas de transmisión eran limitadas, se dividían sus recursos de manera que un host no interfiera demasiado con otro host. Comparten el ancho de banda de una red en forma dinámica, con multiplexado estadístico y se comparten los recursos con base en la demanda. Se puede aplicar en capas bajas para un solo enlace o en capas altas para una red. Evitar que un emisor rápido inunde de datos a un receptor lento, se le denomina **control de flujo**. Si la red sufre un exceso de solicitudes se le conoce como **congestión**. La red puede ofrecer un ancho de banda fijo; si la puntualidad es importante, es de tiempo real, necesita alto rendimiento. La **calidad del servicio** es el nombre que se da a los mecanismos que reconcilian estas demandas competitivas.

Asegurar la red y defenderla contra amenazas:

Los mecanismos que proveen **confidencialidad** nos defienden contra esta amenaza, los de **autenticación** evitan que alguien se haga pasar por otra persona. Los de **integridad**, evitan cambios clandestinos, se basan en la criptografía.

Comparación entre servicio orientado a conexión y servicio sin conexión.

Las capas pueden ofrecer dos tipos distintos de servicio, orientado a conexión y sin conexión.

En el **orientado a conexión** el usuario del servicio establece primero una conexión, la utiliza y después la libera. El emisor mete bits en un extremo y el receptor los toma en el otro, conserva el orden. Al establecer una conexión, el emisor, el receptor y la subred llevan a cabo una **negociación**, como el tamaño máximo del mensaje, la calidad requerida del servicio y demás cuestiones.

En el **servicio sin conexión** cada mensaje lleva la dirección de destino completa, y cada uno es enrulado en forma independiente. Un **paquete** es un mensaje en la **capa de red**. Cuando los nodos intermedios

reciben un mensaje completo, antes de enviarlo al siguiente nodo, se le llama **comutación de almacenamiento** y luego se produce el envío.

Cada tipo de servicio se puede caracterizar con base en su confiabilidad, para un **servicio confiable**, el receptor tiene que confirmar la recepción de cada mensaje, de manera que el emisor esté seguro de que hayan llegado. El proceso introduce sobrecarga y retardos. Un servicio confiable orientado a la conexión es la transferencia de archivos, tiene dos variaciones menores: **secuencias de mensajes y flujos de bytes**. En la primera variante se conservan los límites de los mensajes. En la segunda variante, la conexión es simplemente un flujo de bytes sin límites en los mensajes. En algunas aplicaciones, las confirmaciones de recepción son inaceptables. Una aplicación es el tráfico de voz digitalizada o voz sobre IP. Es preferible escuchar un poco de ruido que experimentar un retardo, de manera similar, al transmitir una conferencia de video.

Al **servicio sin conexión no confiable** (que significa sin confirmación de recepción) se le denomina **servicio de datagramas**, es la forma más dominante en la mayoría de las redes. En otros casos es conveniente no tener que establecer una conexión para enviar un mensaje, pero la confiabilidad es esencial. En estas aplicaciones se puede utilizar el **servicio de datagramas con confirmación de recepción (servicio sin conexión confiable)**. Es como enviar una carta certificada y solicitar una confirmación de recepción.

Hay otro servicio de solicitud-respuesta. El emisor transmite un solo datagrama que contiene una solicitud y el receptor envía la respuesta. En el modelo cliente-servidor; el cliente emite una petición y el servidor le responde.

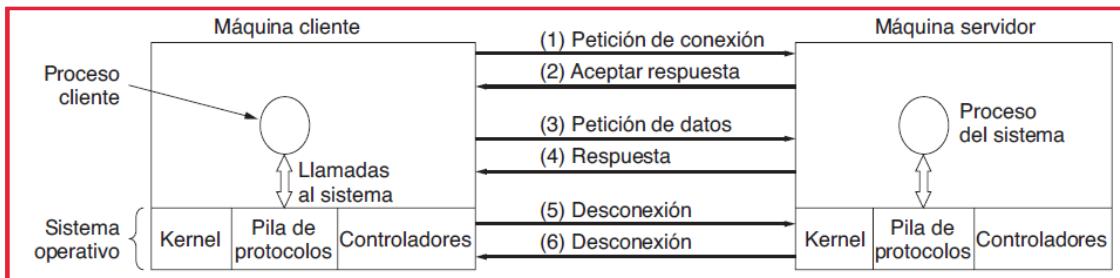
	Servicio	Ejemplo
Orientado a conexión	Flujo de mensajes confiable.	Secuencia de páginas.
	Flujo de bytes confiable.	Descarga de películas.
	Conexión no confiable.	Voz sobre IP.
Sin conexión	Datagrama no confiable.	Correo electrónico basura.
	Datagrama confirmación de recepción.	Mensajería de texto.
	Solicitud-respuesta.	Consulta en una base de datos.

Ethernet no provee una comunicación confiable. Los paquetes se pueden dañar ocasionalmente durante el tránsito. Las capas de protocolos más altas deben tener la capacidad de recuperarse de este problema. En particular, muchos servicios confiables se basan en un servicio de datagramas no confiables.

Primitivas de servicios

Un **servicio** se puede especificar de manera formal como un **conjunto de primitivas (operaciones)** disponibles a los procesos de usuario para que accedan al servicio. Indican al servicio que desarrollen alguna acción. Si la pila de protocolos se encuentra en el sistema operativo, las primitivas son llamadas al sistema, provocan un salto al modo de kernel, para que envíe los paquetes necesarios. El conjunto de primitivas depende de la naturaleza del servicio para el servicio, las orientadas a conexión son distintas a las sin conexión.

Primitiva	Significado
LISTEN	Bloquea en espera de una conexión entrante.
CONNECT	Establece una conexión con un igual en espera.
ACCEPT	Acepta una conexión entrante de un igual.
RECEIVE	Bloquea en espera de un mensaje entrante.
SEND	Envía un mensaje al igual.
DISCONNECT	Termina una conexión.



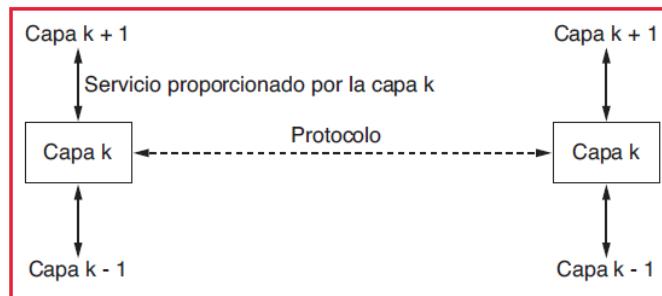
La vida no es tan simple, pueden *malir sal* muchas cosas. Respecto a la **sincronización**, se pueden perder paquetes, se resume la forma en que podría trabajar la comunicación cliente-servidor mediante datagramas con confirmación. Se requieren seis paquetes para completar este protocolo, un protocolo sin conexión se necesitarían dos paquetes. Cuando hay mensajes extensos, errores de transmisión y paquetes perdidos, algunas veces es conveniente tener un flujo de bytes ordenado y confiable entre procesos.

La relación entre servicios y protocolos

Los servicios y los protocolos son conceptos distintos.

Un servicio es un conjunto de primitivas (operaciones) que una capa proporciona a la capa que está encima de ella. El servicio define qué operaciones puede realizar la capa en beneficio de sus usuarios, pero no dice nada sobre cómo se implementan estas operaciones. Un servicio se relaciona con una interfaz entre dos capas.

Un protocolo es un conjunto de reglas que rigen el formato y el significado de los paquetes o mensajes que intercambian las entidades iguales en una capa. Las entidades utilizan protocolos para implementar sus definiciones de servicios. Pueden cambiar sus protocolos a voluntad, siempre y cuando no cambien el servicio visible para sus usuarios. De esta manera, el servicio y el protocolo no dependen uno del otro.



Un servicio define las operaciones que se pueden realizar en un objeto, pero no especifica cómo se implementan. Un protocolo se relaciona con la implementación del servicio, y como tal, no es visible al usuario de este.

MODELOS DE REFERENCIA

En lo abstracto las redes basadas en capas, hay dos arquitecturas de redes: el modelo de referencia **OSI** y el modelo de referencia **TCP/IP**. Aunque ya casi no se utilizan los protocolos asociados con el modelo OSI, el modelo en sí es bastante general y sigue siendo válido; y el modelo TCP/IP tiene las propiedades opuestas: el modelo en sí no se utiliza mucho, pero los protocolos sí son usados ampliamente.

El modelo de referencia OSI

Desarrollada por ISO, **OSI** (Interconexión de Sistemas Abiertos, del inglés Open Systems Interconnection). El modelo OSI tiene siete capas. Los **principios** que se aplicaron:

1. Se debe crear una capa en donde se requiera un nivel diferente de **abstracción**.
2. Cada capa debe realizar una función bien definida.
3. La función de cada capa se debe elegir teniendo en cuenta la definición de protocolos estandarizados internacionalmente.
4. Es necesario elegir los límites de las capas de modo que se minimice el flujo de información a través de las interfaces.
5. La cantidad de capas debe ser suficiente como para no tener que agrupar funciones distintas en la misma capa; además, debe ser lo bastante pequeña como para que la arquitectura no se vuelva inmanejable.

El modelo OSI en sí no es una arquitectura de red, ya que no especifica los servicios y protocolos exactos, sólo indica lo que una debe hacer.

La capa física: se relaciona con la transmisión de bits puros a través de un canal de transmisión. Comprende la acción de asegurarse que cuando uno de los lados envíe un bit 1 el otro lado lo reciba como un bit 1, no como un bit 0; interfaces mecánicas, eléctrica y de temporización, el medio de transmisión físico, etc.

La capa de enlace de datos: La tarea es transformar un medio de transmisión puro en una línea que esté libre de errores de transmisión. El emisor divide los datos de entrada en tramas de datos y transmite las tramas en forma secuencial. Si el servicio es confiable, para confirmar la recepción correcta, el receptor devuelve una **trama de confirmación** de recepción. Evita que un transmisor rápido inunde de datos a un receptor lento realizando regulación de tráfico, controla el acceso al canal compartido. Tiene una subcapa especial llamada **subcapa de control de acceso al medio**, es la que se encarga de este problema.

La capa de red: controla la operación de la subred, cómo se encaminan los paquetes desde el origen hasta el destino. Las rutas se pueden basar en **tablas estáticas**, aunque es más común que se actualicen de manera automática; otra tarea es determinar el inicio de cada conversación, el manejo de la congestión, que adapten la carga que colocan en la red, la calidad del servicio proporcionado, el direcciónamiento, los protocolos diferentes. Todo esto lo realiza para permitir la interconexión de redes heterogéneas.

La capa de transporte: la función es aceptar datos de la capa superior, dividirlos en unidades más pequeñas si es necesario, pasar estos datos a la capa de red y asegurar que todas las piezas lleven al otro extremo, determina el tipo de servicio que debe proveer a la capa de sesión. El tipo más popular, es un **canal punto a punto libre de errores (PPP)**. Es una verdadera capa de extremo a extremo; lleva los datos por toda la ruta desde el origen hasta el destino, el programa en la máquina de origen lleva a cabo una conversación con un programa similar en la máquina de destino. En las capas inferiores, cada uno de los protocolos está entre una máquina y sus vecinos inmediatos, no entre las verdaderas máquinas de origen y de destino.

La capa de sesión: permite a los usuarios en distintas máquinas establecer sesiones entre ellos. Realiza **control del diálogo** (quién va a transmitir), el manejo de **tokens** (evitar operación crítica al mismo tiempo) y la **sincronización** (usar puntos de referencia en las transmisiones extensas)

La capa de presentación: se enfocan principalmente en mover los bits de un lado a otro, en la **sintaxis** y la **semántica** de la información transmitida, definir de una manera abstracta las estructuras de datos que se van a intercambiar.

La capa de aplicación: contiene una variedad de protocolos que los usuarios necesitan con frecuencia. HTTP (Protocolo de Transferencia de Hipertexto, del inglés *HyperText Transfer Protocol*), el cual forma la base para la **World Wide Web**.

El modelo de referencia TCP/IP

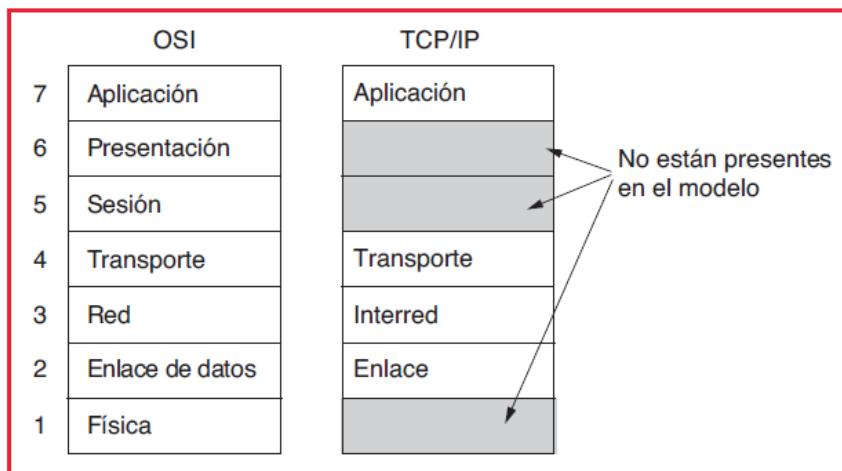
La capa de enlace: Es una red de conmutación de paquetes basada en una **capa sin conexión** que opera a través de distintas redes. Describe qué enlaces se deben llevar a cabo para cumplir con las necesidades de esta capa, es una interfaz entre los hosts y los enlaces de transmisión.

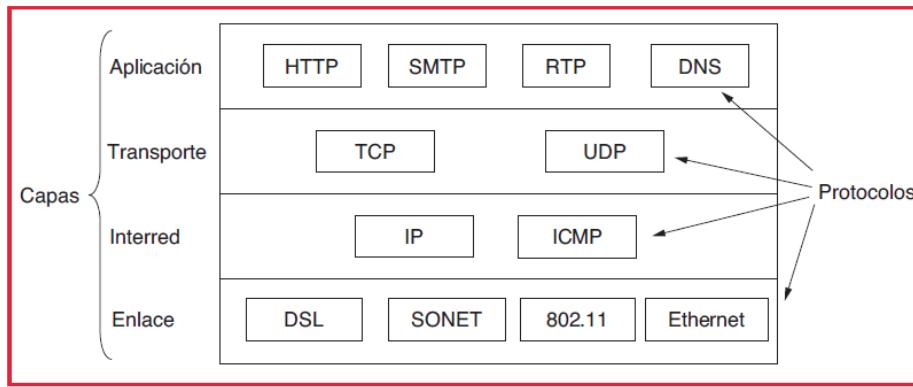
La capa de interred: mantiene unida a toda la arquitectura. Una correspondencia aproximada a la **capa de red** de OSI. Su trabajo es permitir que los hosts inyecten paquetes en cualquier red y que viajen de manera independiente hacia el destino, pueden llegar en un orden totalmente diferente al orden en que se enviaron, en cuyo caso es responsabilidad de las capas más altas volver a ordenarlos. Define un formato de paquete y un protocolo oficial llamado **IP** (Protocolo de Internet, del inglés *Internet Protocol*), un protocolo complementario llamado **ICMP** (Protocolo de Mensajes de Control de Internet, del inglés *Internet Control Message Protocol*) que le ayuda a funcionar. La tarea es entregar los paquetes IP a donde se supone que deben ir. Aquí el **ruteo** de los paquetes es sin duda el principal aspecto, IP no ha demostrado ser efectivo para evitar la congestión.

La capa de transporte: diseñada para permitir que las entidades pares, en los nodos de origen y de destino, lleven a cabo una conversación. Definieron dos protocolos de transporte, **TCP** (Protocolo de Control de la Transmisión, del inglés *Transmission Control Protocol*), es un **protocolo confiable orientado a la conexión** que permite que un flujo de bytes, segmenta el flujo de bytes entrante en mensajes discretos y pasa cada uno a la capa de interred. El proceso **TCP** receptor vuelve a ensamblar los mensajes recibidos para formar el flujo de salida. Maneja el **control de flujo** para que un emisor no pueda inundar a un receptor lento.

El segundo protocolo **UDP** (Protocolo de Datagrama de Usuario, del inglés *User Datagram Protocol*), es un **protocolo sin conexión no confiable**. Se utiliza petición-respuesta, del tipo cliente-servidor, es más importante una entrega oportuna que una entrega precisa, como en la transmisión de voz o video.

La capa de aplicación: TCP/IP no tiene capas de sesión o de presentación, las aplicaciones simplemente incluyen cualquier función de sesión y de presentación que requieran, estas capas se utilizan muy poco en la mayoría de las aplicaciones. Contiene todos los protocolos de alto nivel, como terminal virtual (TELNET), transferencia de archivos (FTP) y correo electrónico (SMTP), el Sistema de nombres de dominio (DNS) para resolución de nombres de hosts a sus direcciones de red; **HTTP**, el protocolo para recuperar páginas de la World Wide Web; y **RTP** (para transmitir medios en tiempo real).





El modelo utilizado en este libro

La fortaleza del modelo de referencia **OSI** es el modelo en sí ha demostrado ser excepcionalmente útil para hablar sobre redes de computadoras, la fortaleza del modelo de referencia **TCP/IP** son los protocolos, que se han utilizado mucho es el modelo híbrido. Tiene cinco capas:



La capa física especifica cómo transmitir bits a través de distintos tipos de medios como señales eléctricas.

La capa de enlace trata sobre cómo enviar mensajes de longitud finita entre computadoras conectadas de manera directa con niveles específicos de confiabilidad. Ethernet y 802.11 son ejemplos.

La capa de red se encarga de combinar varios enlaces múltiples en redes, enviar paquetes entre computadoras distantes, buscar la ruta por la cual enviarán los paquetes. **IP** es el principal protocolo.

La capa de transporte fortalece las garantías de entrega de la capa de Red, provee abstracciones en la entrega, como un flujo de bytes confiable, **TCP** es un ejemplo.

La capa de aplicación contiene programas que hacen uso de la red. Un navegador web se trata del protocolo **HTTP**, **DNS** sería otro ejemplo de protocolo.

Comparación de los modelos de referencia OSI y TCP/IP

Tienen mucho en común. Ambos se basan en el concepto de una pila de protocolos independientes. Además, la funcionalidad de las capas es muy similar, las capas por encima de la de transporte, incluyendo ésta, para proporcionar un servicio de transporte independiente de la red, para los procesos que desean comunicarse. Las capas que están arriba de la de transporte son usuarias orientadas a la aplicación del servicio de transporte.

También tienen diferencias, compararemos los modelos de referencia y no las pilas de protocolos. Conceptos básicos para el modelo OSI:

1. Servicios.
2. Interfaces.
3. Protocolos.

Cada capa desempeña ciertos **servicios** para la capa que está sobre ella. Indica lo que hace la capa, no cómo acceden a ella las entidades superiores ni cómo funciona.

La **interfaz** de una capa indica a los procesos superiores cómo pueden acceder a ella. Especifica cuáles son los parámetros y qué resultados se pueden esperar. Pero no dice nada sobre su funcionamiento interno.

La **capa** es la que debe decidir qué protocolos de peers utilizar. Puede usar los **protocolos** que quiera, siempre y cuando realice el trabajo.

TCP/IP no tenía una distinción clara, aunque las personas han tratado de reajustarlo a fin de hacerlo más parecido al **OSI**. Los únicos servicios que realmente ofrece la capa de interred son SEND IP PACKET y RECEIVE IP PACKET. Los protocolos en el modelo OSI están ocultos. La capacidad de realizar dichos cambios con transparencia es uno de los principales propósitos de tener protocolos en capas. **OSI** se ideó antes de que se inventaran los protocolos correspondientes. Para **TCP/IP** primero llegaron los protocolos y el modelo era en realidad sólo una descripción de los protocolos existentes

Ambos tienen **capas de (inter)red, transporte y aplicación**, pero las demás capas son distintas. El modelo **OSI** soporta ambos tipos de comunicación en la **capa de red**, pero sólo la comunicación orientada a conexión en la **capa de transporte** y el modelo **TCP/IP** sólo soporta un modo en la **capa de red (sin conexión)** pero soporta ambos en la **capa de transporte**.

Una crítica al modelo y los protocolos OS

Ni el modelo OSI ni el modelo TCP/IP, con sus respectivos protocolos son perfectos.

Para el modelo **OSI** posee:

1. Mala sincronización: teoría el apocalipsis de los dos elefantes, los protocolos estándar de OSI quedaron aplastados ya que los protocolos TCP/IP ya se utilizaban mucho en universidades que hacían investigaciones. Para cuando llegó el modelo OSI, los distribuidores no quisieron apoyar una segunda pila de protocolos.

2. Mala tecnología: es muy complejo, los estándares son difíciles de implementar e inefficientes, el control de flujo y el control de errores vuelven a aparecer una y otra vez en cada capa, por lo que es innecesario e ineficiente.

3. Malas implementaciones: gran complejidad del modelo y los protocolos, las implementaciones iniciales fueran enormes, pesadas y lentas.

4. Mala política: mucha gente pensaba que TCP/IP era parte de UNIX, OSI se consideraba como la invención de los ministerios del gobierno y esto no fue de mucha utilidad para la causa de OSI.

Una crítica al modelo de referencia TCP/IP

El modelo no diferencia con claridad los conceptos de **servicios, interfaces y protocolos**, no sirve mucho de guía para diseñar redes modernas que utilicen nuevas tecnologías, no es nada general y no es muy apropiado para describir cualquier pila de protocolos. La **capa de enlace** en realidad no es una capa, es una interfaz (entre las capas de red y de enlace de datos). No distingue entre la **capa física y la de enlace de datos**, la **capa física** trata sobre las características de transmisión y la **capa de enlace** de datos es delimitar el inicio y el fin de las tramas, además de transmitirlas con el grado deseado de contabilidad; **pero el modelo TCP/IP no hace esto**. Otros protocolos se fueron creando según las necesidades del momento.

CAPA FÍSICA

RESUMEN

La capa física es la base de todas las redes. La naturaleza impone en todos los canales dos límites fundamentales que determinan su ancho de banda. Estos límites son: el **límite de Nyquist**, que tiene que ver con los canales sin ruido; y el **límite de Shannon**, para canales con ruido. Los medios de transmisión pueden ser **guiados** y **no guiados**. Los principales medios guiados son el cable de par trenzado, el cable

coaxial y la fibra óptica. Los medios no guiados incluyen la radio terrestre, las microondas, el infrarrojo, los láseres a través del aire y los satélites.

Los **métodos de modulación** digital envían bits a través de los medios guiados y no guiados en forma de señales analógicas. Los **códigos de línea** operan en banda base y las señales se pueden colocar en una banda de paso mediante la modulación de la amplitud, frecuencia y fase de una portadora. Se pueden compartir canales entre los usuarios mediante la **multiplexión por división de tiempo, frecuencia y código**.

El sistema telefónico es un elemento clave en la mayoría de las **WAN**. Sus componentes principales son: lazos locales, troncales y commutadores. **ADSL** ofrece velocidades de hasta 40 Mbps sobre el lazo local al dividirlo en muchas subportadoras que operan en paralelo. Esto excede por mucho las tasas de transmisión de los módems telefónicos. Las **PON** llevan la fibra hasta el hogar para obtener tasas de acceso aún mayores que **ADSL**.

Las **troncales** transportan información digital. Se multiplexan con **WDM** para proveer muchos enlaces de alta capacidad a través de fibras individuales, así como con **TDM** para compartir cada enlace de tasa de transmisión alta entre los usuarios. Tanto la comutación de circuitos como la comutación de **paquetes** son importantes.

Para las aplicaciones móviles, el sistema telefónico fijo no es adecuado. En la actualidad los teléfonos móviles se están usando ampliamente para voz y cada vez más para datos. Han pasado por tres generaciones. La primera generación, 1G, fue analógica y estaba bajo el dominio de **AMPS**. La 2G fue digital, con **GSM** actualmente el sistema de telefonía móvil más implementado en el mundo. La 3G es digital y se basa en la tecnología **CDMA de banda ancha**, aunque también se están implementando **WCDMA** y **CDMA2000**.

El sistema de televisión por cable es un sistema alternativo para acceso a red. Evolucionó de manera gradual del cable coaxial a una red híbrida de fibra óptica y cable coaxial, y de la televisión a televisión e Internet. Potencialmente, ofrece un ancho de banda muy alto, pero en la práctica el ancho de banda real disponible depende mucho de lo que están haciendo los demás usuarios, ya que es compartido.

Capa de Enlace, PPP

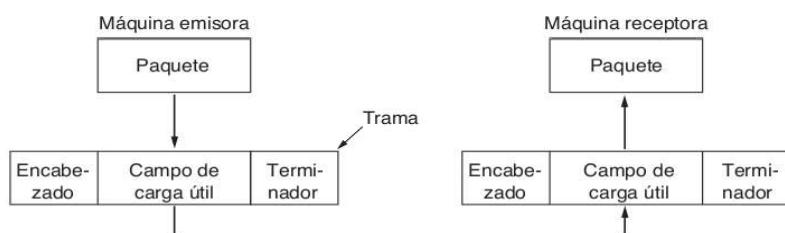
Cuestiones de Diseño de la Capa de Enlace de Datos

Tramas: algoritmos para lograr una comunicación confiable y eficiente de unidades completas de información entre dos máquinas adyacentes.

La capa de enlace de datos utiliza los servicios de la capa física para enviar y recibir bits a través de los canales de comunicación . Tiene varias funciones específicas, entre las que se incluyen:

1. Proporcionar a la capa de red una interfaz de servicio bien definida.
2. Manejar los errores de transmisión.
3. Regular el flujo de datos para que los emisores rápidos no saturen a los receptores lentos.

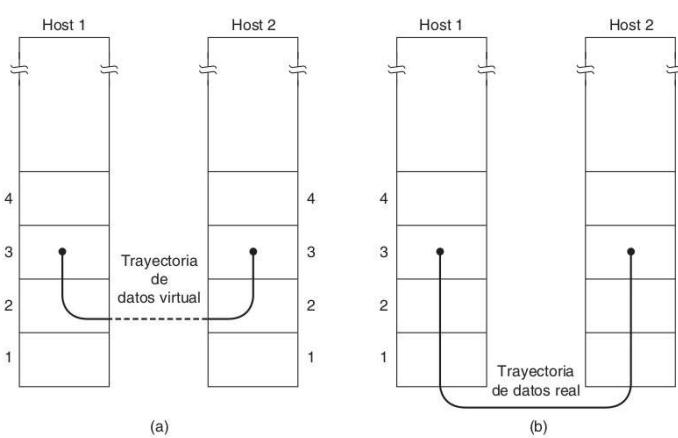
La capa de enlace de datos toma los paquetes que obtiene de la capa de red y los encapsula en tramas para transmitirlos. Cada trama contiene un encabezado, un campo de carga útil (payload) para almacenar el paquete y un terminador.



Servicios proporcionados a la Capa de Red

La función de la capa de enlace de datos es proveer servicios a la capa de red. El servicio principal es la transferencia de datos de la capa de red en la máquina de origen, a la capa de red en la máquina de destino.

En la capa de red de la máquina de origen está una entidad, llamada proceso, que entrega algunos bits a la capa de enlace de datos para que los transmita al destino. La tarea de la capa de enlace de datos es transmitir los bits a la máquina de destino, de modo que se puedan entregar a la capa de red de esa máquina



La capa de enlace de datos puede diseñarse para ofrecer varios servicios. Los servicios reales ofrecidos varían de un protocolo a otro. Normalmente se proporcionan:

1. Servicio sin conexión ni confirmación de recepción.
2. Servicio sin conexión con confirmación de recepción.
3. Servicio orientado a conexión con confirmación de recepción.

Figura 3-2. (a) Comunicación virtual. (b) Comunicación real.

Servicio sin conexión ni confirmación de recepción.	Servicio sin conexión con confirmación de recepción.	Servicio orientado a conexión con confirmación de recepción.
<p>La máquina de origen envía tramas independientes a la máquina de destino sin que ésta confirme la recepción. No se establece una conexión lógica de antemano ni se libera después. Si se pierde una trama debido a ruido en la línea, en la capa de datos no se realiza ningún intento por detectar la pérdida o recuperarse de ella. Utilizada cuando la tasa de error es muy baja, de modo que la recuperación se deja a las capas superiores y es apropiada para el tráfico en tiempo real. Es el modo que usa Ethernet.</p>	<p>No se utilizan conexiones lógicas, pero se confirma de manera individual la recepción de cada trama enviada. Puede ser ineficiente. El emisor sabe si la trama llegó bien o se perdió. Si no ha llegado en un intervalo especificado, se puede enviar de nuevo. Este servicio es útil en canales no confiables (802.11). La confirmación es solo una optimización y no un requerimiento.</p>	<p>Las máquinas de origen y de destino establecen una conexión antes de transferir datos. Cada trama enviada a través de la conexión está numerada, y la capa de enlace de datos garantiza que cada trama enviada llegará a su destino. Es más, garantiza que cada trama se recibirá exactamente una vez y que todas las tramas se recibirán en el orden correcto. apropiado usarlo en enlaces largos y no confiables. Las transferencias pasan por tres fases distintas:</p> <ol style="list-style-type: none"> 1)La conexión se establece haciendo que ambos lados inicialicen las variables y los contadores necesarios para seguir la pista de las tramas que se recibieron y las que no. 2)Se transmiten una o más tramas. 3)la conexión se libera al igual que las variables, los búferes y otros recursos utilizados para mantener la conexión .

Entramado

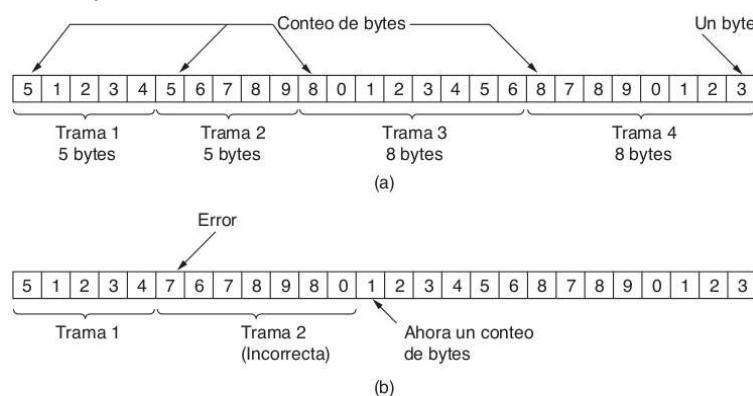
Es responsabilidad de la capa de enlace de datos detectar y, de ser necesario, corregir los errores.

El método común es que la capa de enlace de datos divida el flujo de bits en tramas discretas, calcule un token corto, conocido como suma de verificación, para cada trama, e incluya esa suma de verificación en la trama al momento de transmitirla.

El receptor recalculará la suma de verificación, si es distinta a la que traía la trama, la capa sabe que se ha producido un error en la transmisión. Hay distintas maneras de enviar dicho token y además se tienen que tomar ciertos recaudos por si los mismos bits que forman el token aparecen en los datos enviados. Métodos de entramado:

Conteo de Bytes:

Se vale de un campo en el encabezado para especificar el número de bytes en la trama. Cuando la capa de enlace de datos del destino ve el conteo de bytes, sabe cuántos bytes siguen y, por lo tanto, dónde concluye la trama.



El problema con este algoritmo es que el conteo se puede alterar debido a un error de transmisión, ya que el destino perderá la sincronización y entonces será incapaz de localizar el inicio correcto de la siguiente trama.

Figura 3-3. Un flujo de bytes. (a) Sin errores. (b) Con un error.

Bytes bandera o Relleno de Bytes

Evita el problema de volver a sincronizar nuevamente después de un error al hacer que cada trama inicie y termine con bytes especiales. Con frecuencia se utiliza el mismo byte, denominado byte bandera, como delimitador inicial y final. Dos bytes bandera consecutivos señalan el final de una trama y el inicio de la siguiente. De esta forma, si el receptor pierde alguna vez la sincronización, todo lo que tiene que hacer es buscar dos bytes bandera para encontrar el fin de la trama actual y el inicio de la siguiente.

Técnica relleno de bytes: Una forma de resolver el problema de que los FLAG se encuentren dentro de las tramas de información es hacer que la capa de enlace de datos del emisor inserte un byte de escape especial (ESC) justo antes de cada byte bandera "accidental" en los datos. Luego capa de enlace de datos del lado

receptor quita el byte de escape antes de entregar los datos a la capa de red. Si se repiten los bytes, sigue agregando tantos ESC como hagan falta.

Nota: El esquema de relleno de bytes que se muestra en la figura es una ligera simplificación del esquema empleado en el protocolo PPP (Protocolo Punto a Punto, del inglés Point-to-Point Protocol)

FLAG	Encabezado	Campo de carga útil	Terminador	FLAG
------	------------	---------------------	------------	------

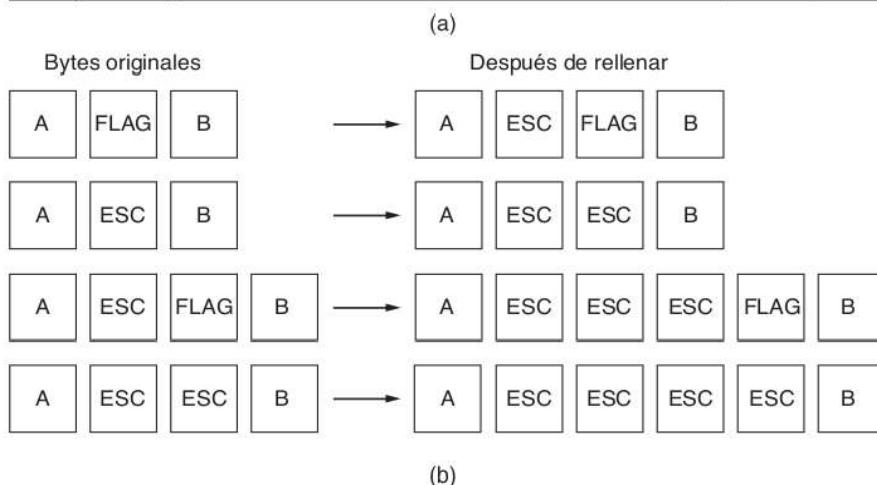


Figura 3-4. (a) Una trama delimitada por bytes bandera. (b) Cuatro ejemplos de secuencias de bytes antes y después del relleno de bytes.

enlace de datos del emisor encuentra cinco bits 1 consecutivos en los datos, inserta automáticamente un O como relleno en el flujo de bits de salida. Este relleno de bits es análogo al relleno de bytes, en el cual se inserta un byte de escape en el flujo de caracteres de salida antes de un byte bandera en los datos. Además asegura una densidad mínima de transiciones que ayudan a la capa física a mantener la sincronización. Cuando el receptor ve cinco bits 1 de entrada consecutivos, seguidos de un bit O, extrae/borra de manera automática el bit O de relleno.

Con el relleno de bits, el límite entre las dos tramas puede ser reconocido sin ambigüedades mediante el patrón bandera

Un efecto secundario del relleno de bits y de bytes es que la longitud de una trama depende ahora del contenido de los datos que lleva.

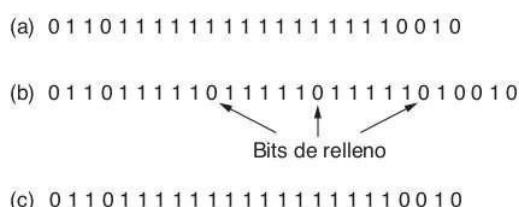


Figura 3-5. Relleno de bits. (a) Los datos originales. (b) Los datos, según aparecen en la línea. (c) Los datos, como se almacenan en la memoria del receptor después de quitar el relleno.

Bits bandera con relleno de bits

El método de delimitar el flujo de bits resuelve una desventaja del relleno de bytes: que está obligado a usar bytes de 8 bits. También se puede realizar el entramado a nivel de bit, de modo que las tramas puedan contener un número arbitrario de bits compuesto por unidades de cualquier tamaño.

Protocolo HDLC (Control de Enlace de Datos de Alto Nivel): Cada trama empieza y termina con un patrón de bits especial, 01111110 o Ox7E en hexadecimal. Este patrón es un byte bandera. Cada vez que la capa de

Violaciones de Codificación de la capa física

Se usan algunas señales reservadas para indicar el inicio y el fin de las tramas. En efecto, estamos usando "violaciones de código" para delimitar tramas. La belleza de este esquema es que, como hay señales reservadas, es fácil encontrar el inicio y final de las tramas y no hay necesidad de llenar los datos. En lugar de transmitir 4bits transmite 5 y no usa todas las combinaciones posibles, por lo que si algo falló lo detecta más fácilmente, además puede enviar códigos especiales.

Ethernet y 802.11 hacen que una trama inicie con un patrón bien definido, conocido como preámbulo

Preámbulo: es un patrón largo de modo que el receptor se pueda preparar para un paquete entrante. El preámbulo va seguido de un campo de longitud (cuenta) en el encabezado, que se utiliza para localizar el final de la trama.

Control de Errores

El protocolo exige que el receptor devuelva tramas de control especiales que contengan confirmaciones de recepción positivas o negativas de las tramas que llegan. Si el emisor recibe una confirmación de recepción positiva de una trama, sabe que la trama llegó de manera correcta . Por otra parte, una confirmación de recepción negativa significa que algo falló y que se debe transmitir la trama otra vez. Debe quedar claro que en un protocolo en el cual el emisor envía una trama y luego espera una confirmación de recepción , positiva o negativa, éste se quedaría esperando eternamente si se perdiera por completo una trama.

Para manejar esta posibilidad se introducen temporizadores en la capa de enlace de datos.

Para evitar que exista una retransmisión continua, generalmente es necesario asignar números de secuencia a las tramas de salida, con el fin de que el receptor pueda distinguir las retransmisiones de las originales.

El asunto de la administración de temporizadores y números de secuencia para asegurar que cada trama llegue finalmente a la capa de red en el destino una sola vez, ni más ni menos, es una parte importante de las tareas de la capa de enlace de datos (y de las capas superiores).

Control de Flujo

Existen 2 métodos para evitar que el transmisor "llene" de mensajes al receptor y consiga un desbordamiento.

Control de flujo basado en retroalimentación: el receptor regresa información al emisor para autorizarle que envíe más datos o por lo menos indicarle su estado. (Utilizado ahora).

El control de flujo basado en tasa: el protocolo tiene un mecanismo integrado que limita la tasa a la que el emisor puede transmitir los datos, sin recurrir a la retroalimentación por parte del receptor.

Se conocen varios esquemas de control de flujo basados en retroalimentación, pero la mayoría se basa en el mismo principio. El protocolo contiene reglas bien definidas respecto al momento en que un emisor puede transmitir la siguiente trama. Con frecuencia estas reglas prohíben el envío de tramas hasta que el receptor lo autorice, ya sea en forma implícita o explícita.

Detección y Corrección de Errores

Estrategias básicas para manejar los errores. Añaden información redundante a los datos que se envían. Una es incluir suficiente información redundante para que el receptor pueda deducir cuáles debieron ser los datos transmitidos. La otra estrategia es incluir sólo suficiente redundancia para permitir que el receptor sepa que ha ocurrido un error (pero no qué error) y entonces solicite una retransmisión. La primera estrategia utiliza códigos de corrección de errores; la segunda usa códigos de detección de errores. El uso de códigos de corrección de errores por lo regular se conoce como FEC (Corrección de Errores hacia Adelante, del inglés Forward Error Correction).

La FEC se utiliza en canales ruidosos puesto que las retransmisiones tienen la misma probabilidad de ser tan erróneas como la primera transmisión.

Una trama consiste en "m" bits de datos y "r" bits redundantes. En un código de bloque, los r bits de verificación se calculan únicamente en función de los m bits de datos con los que se asocian, como si los m bits se buscaran en una gran tabla para encontrar sus correspondientes r bits de verificación .

En un código sistemático, los m bits de datos se envían directamente, junto con los bits de verificación, en vez de que se codifiquen por sí mismos antes de enviarlos.

En un código lineal, los r bits de verificación se calculan como una función lineal de los m bits de datos.

La tasa de código, o simplemente tasa, es la fracción de la palabra codificada que lleva información no redundante.

Código Hamming

Distancia de Hamming:

La cantidad de posiciones de bits en la que difieren dos palabras codificadas. Su significado es que, si dos palabras codificadas están separadas una distancia de Hamming, se requerirán de errores de un solo bit para convertir una en la otra. Dado el algoritmo para calcular los bits de verificación, es posible construir una lista completa de las palabras codificadas válidas, y a partir de esta lista se pueden encontrar las dos palabras codificadas con la menor distancia de Hamming. Esta distancia es la distancia de Hamming del código completo.

Las propiedades de detección y corrección de errores de un código de bloque dependen de su distancia de Hamming.

Funcionamiento

Los bits de la palabra codificada se numeran en forma consecutiva, comenzando por el bit 1 a la izquierda, el bit 2 a su derecha inmediata, etc. Los bits que son potencias de 2 (1, 2, 4, 8, 16, etc.) son bits de verificación . El resto (3, 5, 6, 7, 9, etc.) se llenan con los m bits de datos. El patrón se muestra para un código de Hamming (11,7) con 7 bits de datos y 4 bits de verificación en la figura 3-6.

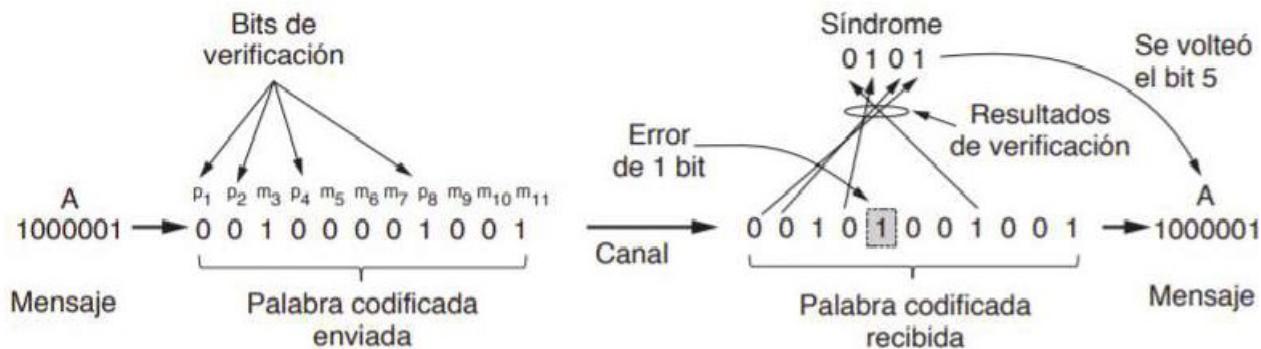


Figura 3-6. Ejemplo de un código de Hamming (11,7) para corregir errores de un solo bit.

Cada bit de verificación obliga a que la suma módulo 2, o paridad de un grupo de bits, incluyéndolo a él mismo, sea par (o impar). Un bit puede estar incluido en varios cálculos de bits de verificación. Para ver a qué bits de verificación contribuye el bit de datos en la posición k, reescriba k como una suma de potencias de 2.

Resultados de verificación : Cuando llega una palabra codificada, el receptor vuelve a realizar los cálculos del bit de verificación, incluyendo los valores de los bits de verificación recibidos. Si los bits de verificación

están correctos, entonces cada resultado de verificación debe ser cero para las sumas de paridad par. En este caso la palabra codificada se acepta como válida.

Síndrome de error: conjunto de resultados de verificación que se utilizan para señalar y corregir el error, cuando el resultado de verificación no da cero.

Las distancias de Hamming son valiosas para comprender los códigos de bloque, y los códigos de Hamming se utilizan en la memoria de corrección de errores.

Código Convolutional

Un codificador procesa una secuencia de bits de entrada y genera una secuencia de bits de salida. No hay un tamaño de mensaje natural, o límite de codificación, como en un código de bloque. La salida depende de los bits de entrada actual y previa. Es decir, el codificador tiene memoria. El número de bits previos de los que depende la salida se denomina longitud de restricción del código. Los códigos convolucionales se especifican en términos de su tasa de transmisión y su longitud de restricción.

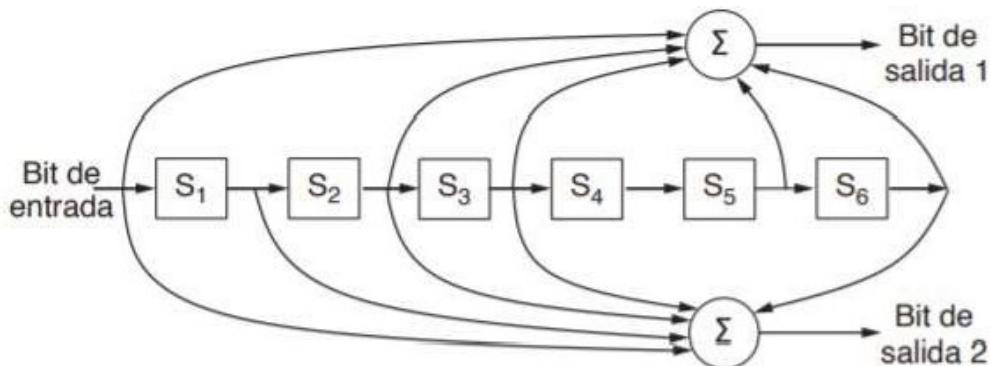


Figura 3-7. El código convolucional binario NASA utilizado en redes 802.11.

Para decodificar un código convolucional es necesario buscar la secuencia de bits de entrada que tenga la mayor probabilidad de haber producido la secuencia observada de bits de salida (incluyendo los errores)

Al final, la secuencia de entrada que requiera la menor cantidad de errores es el mensaje más probable.

Decodificación de decisión suave: método de trabajar con la incertidumbre de un bit

Decodificación de decisión dura: método que trata en decidir si cada bit es un 0 o un 1 antes de la subsiguiente corrección de errores.

Código Reed-Solomon

Son códigos de bloques lineales y con frecuencia también son sistemáticos. A diferencia de los códigos hamming, que operan sobre bits individuales, los códigos reed-solomon operan sobre símbolos de "m" bits.

Los códigos de Reed-Solomon se basan en el hecho de que todo polinomio de n grados se determina en forma única mediante n+1 puntos.

En realidad los códigos de Reed-Solomon se definen como polinomios que operan sobre campos finitos, pero trabajan de una manera similar. Para símbolos de m bits, las palabras codificadas son de 211111 símbolos de longitud.

Los códigos de Reed-Solomon se utilizan mucho en la práctica debido a sus poderosas propiedades de corrección de errores, en especial para los errores de ráfaga.

Puesto que se basan en símbolos de m bits, tanto un error de un solo bit como un error de ráfaga de m bits se tratan simplemente como error de un símbolo.

Cuando se agregan $2t$ símbolos redundantes, un código de Reed-Solomon es capaz de corregir hasta t errores en cualquiera de los símbolos transmitidos.

A menudo los códigos de Reed-Solomon se utilizan en combinación con otros códigos, como el convolucional. Los códigos convolucionales son efectivos a la hora de manejar errores de bits aislados, pero es probable que fallen con una ráfaga de errores, si hay demasiados errores en el flujo de bits recibido. Al agregar un código de Reed-Solomon dentro del código convolucional, la decodificación de Reed-Solomon puede limpiar las ráfagas de errores, una tarea que realiza con mucha eficiencia.

LDPC (verificación de paridad de baja densidad)

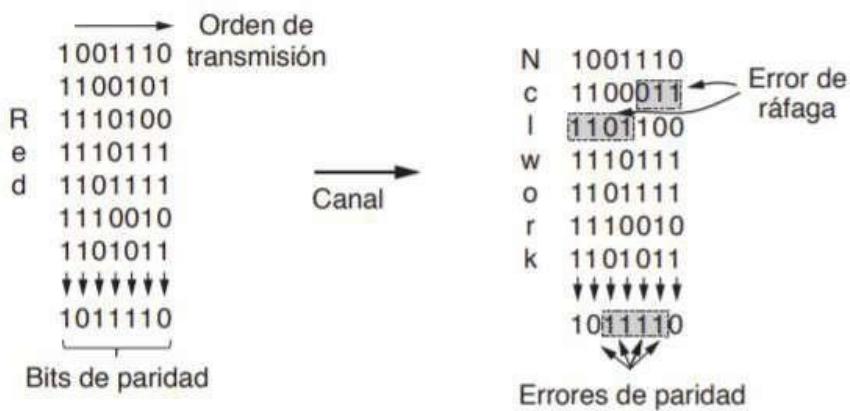
Los códigos LDPC son códigos de bloques lineales. En un código LDPC, cada bit de salida se forma sólo a partir de una fracción de los bits de entrada. Esto conduce a una representación matricial del código con una densidad baja de 1s, razón por la cual tiene ese nombre. Las palabras codificadas recibidas se decodifican con un algoritmo de aproximación que mejora de manera reiterativa con base en el mejor ajuste de los datos recibidos con una palabra codificada válida. Esto corrige los errores.

Los códigos LDPC son prácticos para tamaños grandes de bloques y tienen excelentes habilidades de corrección de errores que superan a las de muchos otros códigos (incluyendo los que vimos antes) en la práctica. ESTE FUNCIONA MEJOR QUE LA COMBINACIÓN DEL REED-SOLOMON + CONVOLUCIONAL

Códigos Detectores de Errores

Paridad

Considere el primer código de detección de errores en el que se adjunta un solo bit de paridad a los datos. El bit de paridad se elige de manera que el número de bits 1 en la palabra codificada sea par (o impar).



Intercalado: proceso que ofrece una mejor protección contra los errores de tal forma que calcula los bits de paridad sobre los datos en un orden distinto al que se transmiten los bits de datos.

El intercalado es una técnica general para convertir un código que detecta (o corrige) los errores aislados en uno que detecta (o corrige) los errores de ráfaga.

Figura 3-8. Intercalado de bits de paridad para detectar un error de ráfaga.

Este método usa n bits de paridad en bloques de kn bits de datos para detectar un solo error de ráfaga de longitud n o menor. Sin embargo, una ráfaga de longitud $n+1$ pasará sin ser detectada si el primer bit está invertido, el último bit está invertido y todos los demás bits son correctos.

Suma de Verificación

La palabra "suma de verificación" se utiliza con frecuencia para indicar un grupo de bits de verificación asociados con un mensaje, sin importar cómo se calculen. Sin embargo, hay otras sumas de verificación más poderosas basadas en la suma acumulada de los bits de datos del mensaje. Por lo general la suma de verificación se coloca al final del mensaje, como el complemento de la función de suma. De esta forma, los errores se pueden detectar al sumar toda la palabra codificada recibida, tanto los bits de datos como la suma de verificación. Si el resultado es cero, no se ha detectado ningún error.

Suma de verificación de Fletcher: es una mejor opción que incluye un componente posicional, en donde se suma el producto de los datos y su posición con la suma acumulada. Este método ofrece una detección más poderosa de los cambios en la posición de los datos.

CRC (Comprobación de Redundancia Cíclica)

En la práctica es la que se utiliza mucho. Es un tipo de código de detección de errores más potente en la capa de enlace. Se lo conoce como código polinomial.

Los códigos polinomiales se basan en el tratamiento de cadenas de bits como representaciones de polinomios con coeficientes de 0 y 1 solamente. Una trama de k bits se considera como la lista de coeficientes de un polinomio con k términos que van de X^{k-1} a X^0 . Se dice que tal polinomio es de grado $k+1$.

La aritmética polinomial se hace mediante una operación módulo 2. (Se dice que un divisor "cabe" en un dividendo si éste tiene tantos bits como el divisor).

Aunque el cálculo requerido para obtener el CRC puede parecer complicado, es fácil calcular y verificar CRC en el hardware mediante circuitos simples de registros de desplazamiento (Peterson y Brown, 1961). En la práctica, casi siempre se usa este hardware. La mayoría de los estándares de red incluyen varios CRC, entre los cuales están casi todas las redes LAN (por ejemplo, Ethernet, 802.11) y los enlaces punto a punto (por ejemplo, paquetes a través de SONET).

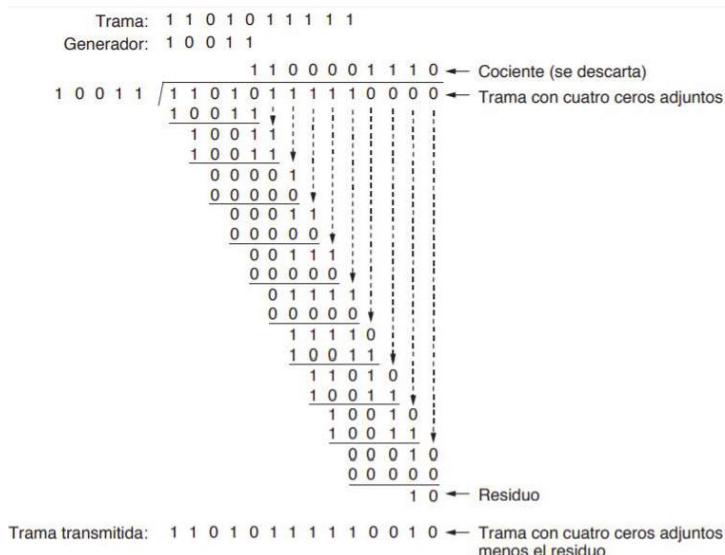


Figura 3-9. Ejemplo del cálculo de CRC.

El algoritmo para calcular el CRC es el siguiente:

1. Sea r el grado de $G(x)$. Anexe r bits cero al final de la trama para que ahora contenga $m + r$ bits y corresponda al polinomio $x^r M(x)$.
2. Divida la cadena de bits correspondiente a $G(x)$ entre la correspondiente a $x^r M(x)$; usando una división módulo 2.
3. Reste el residuo (que siempre es de r o menos bits) a la cadena de bits correspondiente a $x^r M(x)$; usando una resta módulo 2. El resultado es la trama con suma de verificación que va a transmitirse. Llame a su polinomio $T(x)$.

Protocolos de Enlace de Datos

Internet necesita enlaces de punto a punto para estos usos, así como módems de marcación telefónica, líneas rentadas y módems de cable, etc. Un protocolo estándar llamado PPP (Protocolo Punto a Punto) se utiliza para enviar paquetes a través de estos enlaces. Los enlaces SONET y ADSL aplican PPP, sólo que en distintas formas. LOS ENLACES POR CABLE COAXIAL USAN PROTOCOLO DOCSIS

Paquetes sobre SONET

Es el protocolo de capa física que se utiliza con más frecuencia sobre los enlaces de fibra óptica de área amplia que constituyen la espina dorsal de las redes de comunicaciones, incluyendo el sistema telefónico. Provee un flujo de bits que opera a una tasa de transmisión bien definida.

Para transportar paquetes a través de estos enlaces, se necesita cierto mecanismo de entramado para diferenciar los paquetes ocasionales del flujo de bits continuo en el que se transportan. PPP se ejecuta en enrutadores IP para proveer este mecanismo.

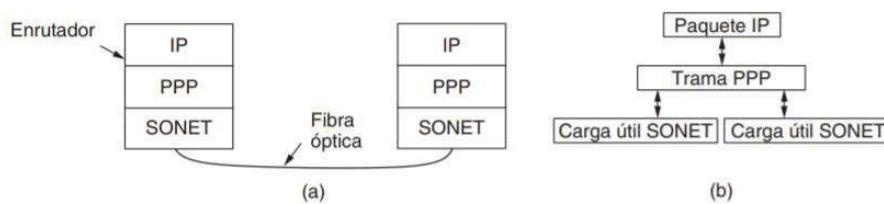


Figura 3-23. Paquete sobre SONET. (a) Una pila de protocolos. (b) Relaciones entre las tramas.

PPP constituye una mejora del protocolo más simple conocido como SLIP (Protocolo de Línea Serial de Internet) se utiliza para manejar la configuración de detección de errores en los enlaces, soporta múltiples protocolos, permite la autentificación y tiene muchas otras funciones.

PPP es un mecanismo de entramado que puede transportar los paquetes de varios protocolos a través de muchos tipos de capas físicas.

Con un amplio conjunto de opciones, PPP provee tres características principales:

1. Un método de entramado que delinea sin ambigüedades el final de una trama y el inicio de la siguiente. El formato de trama también maneja la detección de errores.
2. Un protocolo de control de enlace para activar líneas, probarlas, negociar opciones y desactivarlas en forma ordenada cuando ya no son necesarias. Este protocolo se llama LCP (Protocolo de Control de Enlace, del inglés Link Control Protocol).
3. Un mecanismo para negociar opciones de capa de red con independencia del protocolo de red que se vaya a utilizar. El método elegido debe tener un NCP (Protocolo de Control de Red, del inglés Network Control Protocol) distinto para cada capa de red soportada.

Diferencias con HDLC

La diferencia principal entre PPP y HDLC es que el primero está orientado a bytes, no a bits. En particular, PPP usa el relleno de bytes en las líneas y todas las tramas tienen un número entero de bytes. HDLC utiliza relleno de bytes y permite tramas de, por ejemplo, 30.25 bytes.

La segunda diferencia importante en la práctica es que HDLC provee una transmisión confiable con una ventana deslizante, confirmaciones de recepción y expiración de temporizadores en la forma que hemos visto. PPP también puede proveer una transmisión confiable en entornos ruidosos, como las redes inalámbricas.

[Que verga es HDLC?] HDLC (High-Level Data Link Control, control de enlace de datos de alto nivel) es un protocolo de comunicaciones de propósito general punto a punto, que opera a nivel de enlace de datos. Evolución de SDLC. Proporciona recuperación de errores en caso de pérdida de paquetes de datos, fallos de secuencia y otros, por lo que ofrece una comunicación confiable entre el transmisor y el receptor.

Formato de Trama PPP

Bytes	1	1	1	1 a 2	Variable	2 a 4	1
	Bandera 01111110	Dirección 11111111	Control 00000011	Protocolo	Carga útil }}	Suma de verificación }}	Bandera 01111110

Figura 3-24. El formato de trama completa de PPP para la operación en modo no numerado.

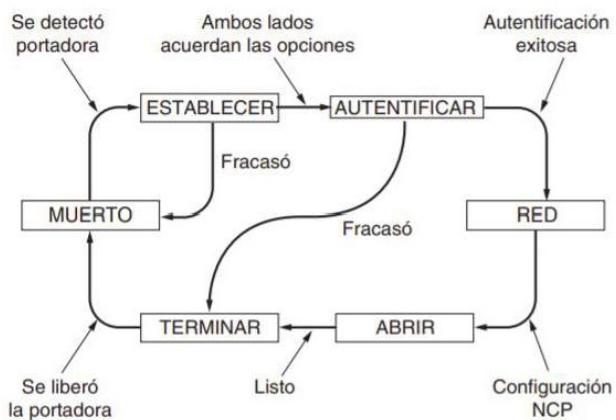
- Bandera: Todas las tramas PPP comienzan con el byte bandera del estándar de HDLC 0x7E (01111110). Este byte de bandera se rellena con bytes si ocurre dentro del campo de carga útil (Payload), mediante el byte de escape 0x7D. El siguiente byte es el resultado de aplicar un XOR al byte de escape y a 0x20, con lo cual se voltean el quinto bit.
- Dirección: Este campo siempre se establece al valor binario 11111111 para indicar que todas las estaciones deben aceptar la trama. Al usar este valor evitamos tener que asignar direcciones de la capa de enlace de datos.
- Control: Su valor predeterminado es 00000011. Este valor indica una trama no numerada.

Dado que los campos Dirección y Control son siempre constantes en la configuración predeterminada, LCP proporciona los mecanismos necesarios para que las dos partes negocien una opción que los omita por completo y ahorre 2 bytes por trama.

- Protocolo: Su tarea es indicar la clase de paquete que está en el campo de Carga útil. Se definen códigos que empiezan con un bit 0 para la versión 4 de IP, la versión 6 de IP y otros protocolos de capa de red que se podrían usar. El tamaño predeterminado del campo Protocolo es de 2 bytes (negociable a 1 byte).
- El campo Carga útil es de longitud variable, hasta cierto máximo negociado. Si la longitud no se negocia mediante LCP durante el establecimiento de la línea, se usa una longitud predeterminada de 1500 bytes. De ser necesario se puede incluir un relleno después de la carga útil.
- Suma de Verificación: Por lo general es de 2 bytes, aunque se puede negociar una suma de verificación de 4 bytes. La suma de verificación es, de hecho, la misma CRC de 32 bits. La suma de verificación de 2 bytes es también una CRC estándar en la industria.

También hay una característica inusual. La carga útil de PPP se mezcla aleatoriamente (scrambled) antes de insertarla en la carga útil de SONET. Porque el flujo de bits de SONET necesita transiciones frecuentes de bits para la sincronización. Con la aleatorización, la probabilidad de que un usuario pueda ocasionar problemas al enviar una larga secuencia de ceros es muy baja.

Antes de transportar las tramas PPP a través de líneas SONET, hay que establecer y configurar el enlace PPP. Las fases por las que pasa el enlace al activarlo, utilizarlo y desactivarlo otra vez se muestran en la figura:



1. Estado MUERTO, no hay conexión en la capa física. Se establece una conexión en la capa física.
2. ESTABLECER. Los iguales de PPP intercambian paquetes LCP (que se transportan en el campo Carga útil de una trama PPP) para configurar la conexión.
3. Negociación exitosa. AUTENTIFICAR. Ahora las dos partes pueden verificar las identidades entre sí. Si la autenticación tiene éxito.
4. Estado RED y se envían una serie de paquetes NCP para configurar la capa de red. Los paquetes intercambiados dependen del protocolo de red en uso.

Figura 3-25. Diagrama de estado para activar y desactivar un enlace PPP.

5. Una vez que el enlace llega a ABRIR, se puede llevar a cabo el transporte de datos. En este estado es en donde se transportan los paquetes IP en tramas PPP a través de la línea SONET. Al terminar el transporte.
6. Estado TERMINAR y de ahí se regresa al estado MUERTO, cuando se desactiva la conexión de la capa física.

Subcapa de Acceso al Medio, Ethernet

Los enlaces de red se pueden dividir en dos categorías: los que utilizan conexiones punto a punto y los canales de difusión, el asunto clave es la manera de determinar quién puede utilizar el canal. Los canales de difusión a veces se denominan canales multiacceso o canales de acceso aleatorio. Los protocolos que se utilizan para determinar quién sigue en un canal multiacceso pertenecen a una subcapa de la capa de enlace de datos llamada subcapa MAC (Control de Acceso al Medio, del inglés Medium Access Control), tiene especial importancia en las LAN, en especial las inalámbricas y las WAN usan enlaces punto a punto. La subcapa MAC es la parte inferior de la capa de enlace.

El Problema de Asignación del Canal

El tema central es la forma de asignar un solo canal de difusión entre usuarios competidores, podría ser una parte del espectro inalámbrico, un solo alambre fibra óptica en donde se conectan varios nodos.

Asignación estática de canal

La manera tradicional de asignar un solo canal es dividir su capacidad mediante la multiplexión, como el FDM (Multiplexión por División de Frecuencia, del inglés Frequency Division Multiplexing). Si hay N usuarios, el ancho de banda se divide en N partes de igual tamaño. cuando el número de emisores es grande y varía continuamente, o el tráfico se hace en ráfagas, el FDM presenta problemas. Se desperdiciaría una buena parte del valioso espectro y si más de N usuarios quieren comunicarse se les negaría el permiso por falta de ancho de banda. Una asignación estática es un mal arreglo para la mayoría de los sistemas de cómputo, retardo promedio, T, al enviar una trama a través de un canal con C bps de capacidad. Suponemos que las tramas llegan al azar con una tasa de llegada promedio de λ tramas/seg y la longitud es variable de promedio de $1/\mu$ bits la tasa de servicio del canal es de μC tramas/seg. El retardo promedio al usar el canal dividido es N veces T.

Supuestos para la asignación dinámica de canales

$$T = \frac{1}{\mu C - \lambda}$$

$$T_N = \frac{N}{\mu C - \lambda}$$

Todo se basa en cinco supuestos clave:

- 1) Tráfico independiente: N estaciones independientes con un programa, que genera tramas para transmisión, con un número esperado de tramas que se generan en un intervalo de longitud Δt es de $\lambda \Delta t$, λ es la tasa de tramas y la estación se bloquea hasta que la trama se haya transmitido con éxito.
- 2) Canal único: Hay un solo canal disponible, todas las estaciones pueden transmitir y recibir de él.
- 3) Colisiones observables: Si dos tramas se transmiten en forma simultánea, la señal se altera, se le llama colisión. Todas las estaciones pueden detectar una colisión y se debe volver a transmitir después. No hay otros errores excepto por las colisiones.
- 4) Tiempo continuo o ranurado: Es continuo, cuando la transmisión de una trama puede comenzar en cualquier momento. Ranurado cuando las transmisiones de las tramas deben empezar al inicio de una ranura. Puede contener 0, 1 o más tramas.
- 5) Detección de portadora o sin detección de portadora: Las estaciones pueden saber si el canal está en uso antes de intentar usarlo. Si se detecta el canal ocupado, ninguna estación intentará utilizarlo. Si no hay detección de portadora, no pueden detectar el canal y simplemente transmiten, después pueden determinar si la transmisión tuvo éxito.

Algunos aspectos respecto a los 5 ítems:

- En el primero, las llegadas de las tramas son independientes, en todas las transmisiones y se generan en forma impredecible, pero a una tasa de transmisión constante. Este supuesto no es en sí un buen modelo, se sabe que llegan en ráfagas durante un rango de escala de tiempo. Sin embargo, es útil matemáticamente ya que ayuda a analizar a grandes rasgos cómo cambia el rendimiento respecto a otros diseños.
- El segundo es la esencia del modelo. No existen formas externas de comunicación, no pueden levantar la mano para solicitar la palabra.
- El tercero, las estaciones necesitan detectar las colisiones si quieren retransmitir las tramas, se puede diseñar hardware para detectar cuando ocurran. Es mucho más difícil para los canales inalámbricos, casi siempre se deducen después de que ocurren, debido a que se esperaba una trama de confirmación de recepción y nunca llegó.
- El cuarto, el tiempo ranurado se puede usar para mejorar el rendimiento, requiere que las estaciones sigan un reloj maestro o que sincronicen sus acciones, esto no siempre está disponible.
- El quinto, una red puede tener detección de portadora o no, las redes alámbricas tienen esta detección, aunque las redes inalámbricas no siempre la pueden utilizar.

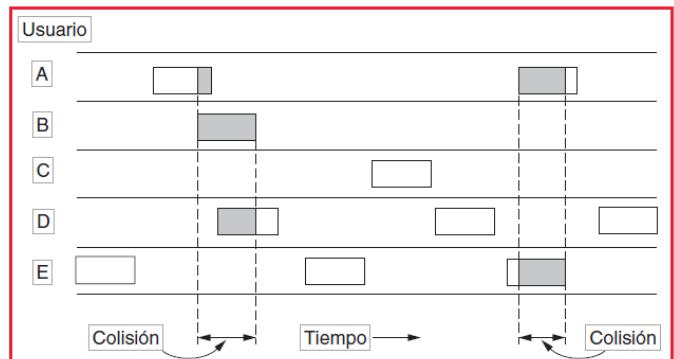
Protocolos de Acceso Múltiple

ALOHA [los inventaron en Hawai.. muy ocurrientes con los nombres]

Trataban de conectar a los usuarios en islas remotas a la computadora principal, utilizaban radios de corto rango, cada terminal de usuario compartía la misma frecuencia para enviar tramas a la computadora central. ALOHA: puro y ranurado. Difieren en cuanto a si el tiempo es continuo, como en la versión pura, o si se divide en ranuras discretas.

ALOHA puro

Permitir que los usuarios transmitan cuando tengan datos por enviar, habrá colisiones. Los emisores necesitan saber, si esto pasó, después de que cada estación envía su trama, la estación receptora vuelve a difundir la trama a todas las estaciones. Así, una estación emisora puede escuchar la difusión de la estación terrena maestra (hub) para ver si pasó su trama o no. En otros sistemas, LAN alámbricas, el emisor podría escuchar si hay colisiones mientras transmite. Si la trama fue destruida, el emisor espera un tiempo aleatorio y la envía de nuevo. Debe ser aleatorio o chocarán una y otra vez, en sincronía. Los sistemas en los cuales varios usuarios comparten un canal común, se conocen como sistemas de contención. Todas las tramas son de la misma longitud porque la velocidad real de transmisión (throughput) de los sistemas ALOHA se maximiza al tener tramas con un tamaño uniforme en lugar de variable. Cada vez que dos tramas traten de ocupar el canal al mismo tiempo, habrá una colisión, tendrán sumas de verificación incorrectas y tendrán que volver a transmitirse.



¿Cuál es la eficiencia de un canal ALOHA? ¿Qué fracción de todas las tramas transmitidas escapa a las colisiones?. Primero consideremos un conjunto infinito de usuarios, un usuario siempre está escribiendo o esperando. Al principio todos están en el estado de escritura. Al terminar una línea, deja de escribir, en espera de una respuesta. Después, la estación transmite una trama hasta la computadora central y verifica el canal para saber si llegó con éxito. De ser así, el usuario ve la respuesta y continúa escribiendo. Si no, continúa esperando mientras la estación transmite la trama una y otra vez hasta que se envía con éxito. El "tiempo de trama" es el tiempo necesario para transmitir la trama estándar de longitud fija, suponemos que las tramas nuevas generadas están bien modeladas según Poisson con una media de N tramas por tiempo de trama, la población infinita es necesaria para asegurar que N no disminuya a medida que se bloquean. Si $N > 1$, los usuarios están generando tramas a una tasa mayor que la que puede manejar el canal, y casi

todas sufrirán una colisión; una velocidad de transmisión razonable esperaríamos que $0 < N < 1$. Además también generan retransmisiones de tramas que con anterioridad sufrieron colisiones. Supongamos que están bien modeladas según Poisson, con una media de G tramas por tiempo de trama, $G \geq N$. Con carga baja $N \approx 0$ habrá pocas colisiones, pocas retransmisiones, $G \approx N$. Con carga alta habrá muchas colisiones, $G > N$. Con todas las cargas, la velocidad real de transmisión S es sólo la carga ofrecida, G , multiplicada por la probabilidad, P_0 , de que una transmisión tenga éxito $S = G P_0$. “ t ” es el tiempo requerido para enviar una trama. Otro usuario que generó una trama entre el tiempo t_0 y t_0+t , colisionará y cualquier otra trama que se inicie entre t_0+t y t_0+2t también lo hará.

La probabilidad de que se generen k tramas durante un tiempo en donde se esperan G tramas, está dada por la distribución de Poisson: =====>

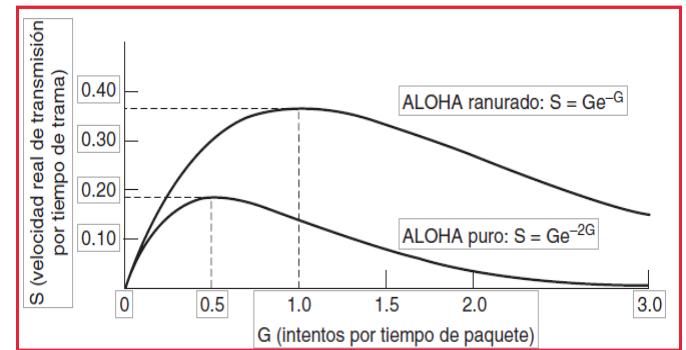
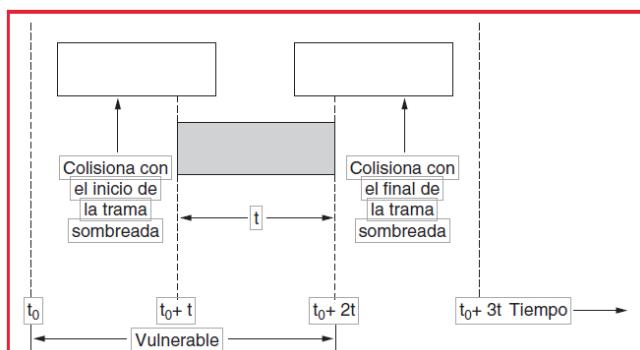
La probabilidad de cero tramas es e^{-G}

$$\Pr[k] = \frac{G^k e^{-G}}{k!}$$

La probabilidad de que no se inicien tramas durante todo el periodo vulnerable está dada entonces por $P_0 = e^{-2G}$. Si $S = G P_0$, obtenemos:=====

$$S = Ge^{-2G}$$

La máxima velocidad real de transmisión ocurre cuando $G=0.5$, con $S=1/2e$, lo más que podemos esperar es un uso del canal de 18%. Si todo mundo estuviera transmitiendo al azar, difícilmente podríamos esperar una tasa de éxito de 100%.



ALOHA ranurado

Un método para duplicar la capacidad de un sistema ALOHA es dividir el tiempo en intervalos discretos llamados ranuras, cada uno de los cuales correspondía a una trama, requiere que los usuarios acuerden límites de ranura. Una manera de lograr la sincronización sería tener una estación especial que emitiera una señal al comienzo de cada intervalo, como un reloj. No se permite que una estación envíe cada vez que el usuario escribe una línea. En cambio, se le obliga a esperar el comienzo de la siguiente ranura. Esto reduce el periodo vulnerable a la mitad.

La probabilidad de que no haya más tráfico durante la misma ranura es e^{-G} . Esto conduce a:=====

$$S = Ge^{-G}$$

Alcanza su máximo valor en $G=1$, su velocidad real de transmisión de $S=1/e$ y podemos esperar un 37% de ranuras vacías, 37% de éxitos y 26% de colisiones. Si se opera con valores mayores de G se reduce el número de ranuras vacías pero aumenta de manera exponencial el número de colisiones.

El número esperado de transmisiones, E , por cada línea que se introduce en la terminal es:

$$E = e^G$$

Como resultado de la dependencia exponencial de E respecto a G , pequeños aumentos en la carga del canal pueden reducir drásticamente su desempeño. ALOHA ranurado se utilizó en algunos sistemas experimentales iniciales, después casi se olvidó por completo.

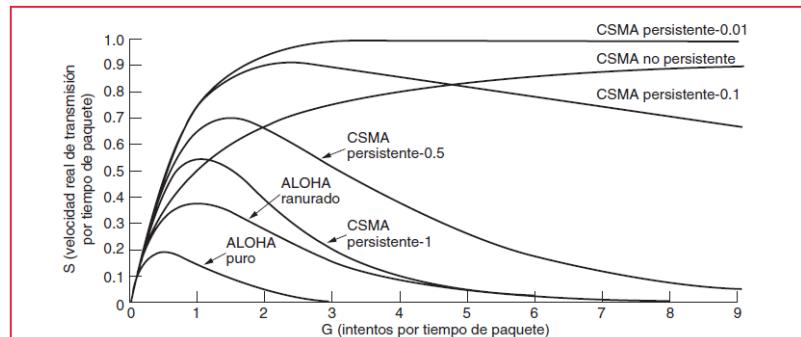
Protocolos de acceso múltiple con detección de portadora

En las redes LAN es posible que las estaciones detecten lo que están haciendo las demás estaciones y adapten su comportamiento, se llaman protocolos de detección de portadora.

CSMA persistente y no persistente

CSMA (Acceso Múltiple con Detección de Portadora, del inglés Carrier Sense Multiple Access) persistente-1. Cuando una estación tiene datos por enviar, primero escucha el canal para saber si alguien más está transmitiendo en ese momento. Si el canal está inactivo, envía sus datos, si está ocupado, espera hasta que se desocupe. La estación transmite una trama, si ocurre una colisión, espera una cantidad aleatoria de tiempo y comienza de nuevo. Se llama persistente-1 porque la estación transmite con una probabilidad de 1 cuando encuentra que el canal está inactivo. Si dos estaciones están listas a la mitad de la transmisión de una tercera estación, esperarán hasta que termine la transmisión y después ambas empezarán a transmitir exactamente al mismo tiempo, esto producirá una colisión. El retardo de propagación tiene un efecto importante sobre las colisiones. Justo después de que una estación comienza a transmitir, otra estación está lista para enviar y detecta el canal. Si la señal de la primera estación no ha llegado aún a la segunda, esta última detectará un canal inactivo y comenzará a transmitir, también dará como resultado una colisión, depende del número de tramas que quepan en el canal, o producto de ancho de banda-retardo del canal. Cuanto mayor sea el producto, más importante será este efecto y peor el desempeño del protocolo. Este protocolo tiene un mejor desempeño que el ALOHA. Un segundo protocolo de detección de portadora es el CSMA no persistente, hace un intento consciente por ser menos egoísta. Una estación escucha el canal cuando desea enviar una trama y, si nadie más está transmitiendo, comienza a hacerlo. Pero si el canal ya está en uso, la estación no lo escuchará de manera continua, sino que esperará un periodo aleatorio y repetirá el algoritmo. Este algoritmo conduce a un mejor uso del canal, pero produce mayores retardos que el CSMA persistente-1.

CSMA persistente-p se aplica a canales ranurados. Cuando una estación está lista para enviar, escucha el canal. Si se encuentra inactivo, la estación transmite con una probabilidad p. Con una probabilidad q=1-p, se posterga hasta la siguiente ranura. Si esa ranura también está inactiva, la estación transmite o posterga una vez más, se repite hasta que se transmite la trama o hasta que otra estación comienza a transmitir. En el segundo caso, actúa como si hubiera ocurrido una colisión. Si detecta que el canal está ocupado, espera hasta la siguiente ranura y aplica el algoritmo anterior. El estándar IEEE 802.11 usa una versión refinada del CSMA persistente-p.

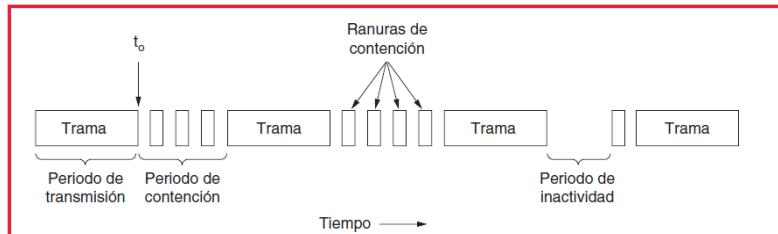


CSMA con detección de colisiones

Los CSMA persistentes y no persistentes son una mejora respecto a ALOHA porque aseguran que ninguna estación empezará a transmitir mientras el canal esté ocupado. Pero si dos estaciones detectan que el canal está inactivo y empiezan a transmitir al mismo tiempo, sufrirán una colisión. Otra mejora es que detecten rápidamente la colisión y dejen de transmitir de inmediato, ahorra tiempo y ancho de banda. CSMA/CD (CSMA con Detección de Colisiones, del inglés CSMA with Collision Detection), es la base de la clásica LAN Ethernet. La detección de colisiones es un proceso analógico. El hardware de la estación debe escuchar el canal mientras transmite. Si la señal que recibe es distinta de la señal que está enviando, sabe que está ocurriendo una colisión. En t=to, una estación ha terminado de transmitir su trama. Cualquier otra estación que tenga una trama por enviar puede intentar hacerlo ahora. Si dos o más estaciones deciden transmitir en forma simultánea, habrá una colisión. Si detecta una colisión, aborta la transmisión, espera un tiempo

aleatorio e intenta de nuevo. CSMA/CD consistirá en períodos alternantes de contención y transmisión, con períodos de inactividad.

Dos estaciones comienzan a transmitir exactamente en el momento t_0 . Sea τ el tiempo que tarda una señal en propagarse entre las dos estaciones más lejanas. En $t_0 + \tau - \epsilon$, un instante antes de que la señal llegue a la estación más lejana, esa estación también comienza a transmitir, detecta la colisión y se detiene, pero la pequeña ráfaga de ruido causada por la colisión no regresa a la estación original, hasta el tiempo $2\tau - \epsilon$. En el peor caso una estación no puede estar segura de que ha tomado el canal hasta que ha transmitido durante 2τ sin detectar una colisión. Mejorará en forma considerable el desempeño si el tiempo de la trama es mucho mayor que el tiempo de propagación.

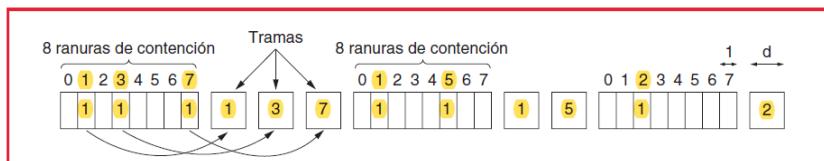


Protocolos libres de colisiones

Las colisiones no ocurren en CSMA/CD una vez capturado el canal, aún pueden ocurrir durante el período de contención. Estas afectan el desempeño del sistema, cuando el producto ancho de banda-retardo es grande, como cuando el cable es largo y las tramas son cortas. También hacen variable el tiempo de envío de una trama, lo cual no es bueno para el tráfico en tiempo real, CSMA/CD no se puede aplicar en forma universal. Hay protocolos que resuelven la contención por el canal sin que haya colisiones. En los protocolos que describiremos supondremos N estaciones, con una dirección única, ¿qué estación obtiene el canal después de una transmisión exitosa?

Un protocolo de mapa de bits

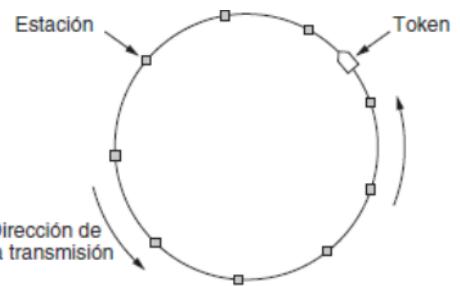
Cada período de contención consiste exactamente de N ranuras, la estación j puede anunciar que tiene una trama por enviar, para lo cual inserta un bit 1 en la ranura j . Una vez que han pasado las N ranuras, cada estación tiene un completo conocimiento acerca de cuáles son las estaciones que quieren transmitir.



Como todos están de acuerdo en quién sigue a continuación, nunca habrá colisiones. Una vez que la última estación lista haya transmitido su trama, comienza otro período de contención de N bits. Si una estación está lista justo después de que ha pasado su ranura de bit, deberá permanecer inactiva hasta que cada una de las demás estaciones haya tenido su oportunidad y el mapa de bits haya comenzado de nuevo. Se llaman protocolos de reservación, debido a que reservan el canal por anticipado y evitan colisiones. Mediremos el tiempo en unidades de la ranura de bit de contención, con tramas de datos en d unidades de tiempo. Con carga baja, el mapa de bits simplemente se repetirá una y otra vez, debido a la falta de tramas de datos. En promedio, la estación tendrá que esperar $N/2$ ranuras para que el escaneo actual termine, además de otras N ranuras para que el siguiente escaneo se ejecute antes de que pueda empezar a transmitir. Las estaciones de menor numeración deben esperar en promedio $1.5N$ ranuras y las estaciones de mayor $0.5N$ ranuras, la media de todas las estaciones es de N ranuras. La eficiencia del canal cuando la carga es baja, con N bits y la cantidad de datos es de d bits, es de $d/(d + N)$. Si la carga es alta, el período de contención de N bits se prorrota entre N tramas, produce una sobrecarga de sólo 1 bit por trama, eficiencia de $d/(d + 1)$.

Paso de token

Otra forma de lograr lo mismo es pasar un pequeño mensaje conocido como token de una estación a otra. Si una estación tiene una trama puesta en cola para transmitirla cuando recibe el token, puede enviar esa trama antes de pasar el token a la siguiente estación. Si no tiene una trama puesta en cola, simplemente pasa el token. En un protocolo token ring, las estaciones están conectadas una con otra en un solo anillo, pasar el token consiste en recibir el token proveniente de una dirección y transmitirlo hacia la otra dirección. Para evitar que la trama circule en forma indefinida, una estación necesita quitarla del anillo.



Esta estación puede ser la que envió originalmente la trama, después de que haya pasado por un ciclo completo. No necesitamos un anillo físico podría ser también un solo bus extenso, para enviar el token en la secuencia predefinida. Se le conoce como token bus. El desempeño es similar al del protocolo de mapa de bits, aunque las ranuras de contención y las tramas de un ciclo están ahora entremezcladas. Después de enviar una trama, cada estación debe esperar a que las N estaciones envíen el token a sus estaciones vecinas y que las otras N – 1 estaciones envíen una trama, si es que la tienen. No es necesario propagar cada token a todas las estaciones antes de que el protocolo avance al siguiente paso.

Conteo descendente binario

Un problema con el mapa de bits, y el paso de token, es la sobrecarga de 1 bit por estación, no se escala bien en redes con miles de estaciones. Podemos tener mejores resultados si usamos direcciones de estación binarias con un canal que combine las transmisiones. Una estación que quiere utilizar el canal difunde su dirección como una cadena binaria de bits, comenzando por el bit de mayor orden. Los bits en cada posición de dirección de las diferentes estaciones se les aplica un OR por el canal cuando se envían al mismo tiempo. Asume que los retardos de transmisión son insignificantes. Para evitar conflictos, tan pronto como una estación ve que una posición de bit de orden alto, cuya dirección es 0, ha sido sobreescrita con un 1, se da por vencida. Las estaciones 0010, 0100, 1001 y 1010 están tratando de obtener el canal, el primer tiempo transmiten 0, 0, 1 y 1, respectivamente, se les aplica el OR para formar un 1, 0010 y 0100 ven el 1 y se dan por vencidas durante esta ronda. Las estaciones 1001 y 1010 continúan. El siguiente bit es 0, y ambas estaciones continúan. El siguiente bit es 1, por lo que la estación 1001 se da por vencida. La ganadora es la estación 1010, debido a que tiene la dirección más alta, ahora puede transmitir una trama. Tiene la propiedad de que estaciones con mayor numeración tienen una prioridad más alta que las de menor numeración. La eficiencia es de $d/(d + \log_2(N))$. Pero si el formato de trama se escoge ingeniosamente de modo que la dirección del emisor sea el primer campo en la trama, la eficiencia es del 100%.

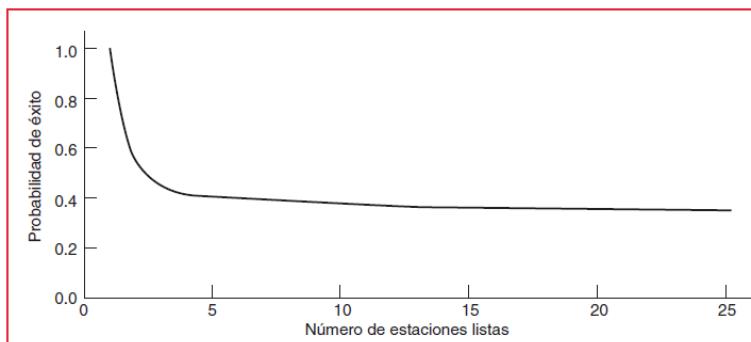
Tiempo de bit	0 1 2 3	GUION INDICA SILENCIO
0 0 1 0	0 - - -	
0 1 0 0	0 - - -	
1 0 0 1	1 0 0 -	
1 0 1 0	1 0 1 0	
Resultado	1 0 1 0	
Las estaciones 0010 y 0100 ven este 1 y se dan por vencidas		
La estación 1001 ve este 1 y se da por vencida		

Protocolos de contención limitada

CSMA, y los protocolos libres de colisión. Cada estrategia se puede recomendar según lo bien que funciona en relación con el retardo con carga baja y la eficiencia del canal con carga alta. En carga ligera, la contención (ALOHA puro o ranurado) es preferible debido a su bajo retardo (ya que las colisiones son raras). A medida que aumenta la carga, la contención es menos atractiva, debido a que la sobrecarga asociada al arbitraje del canal se vuelve mayor. Para los protocolos libres de colisiones. Con carga baja tienen un retardo alto, pero a medida que aumenta la carga mejora la eficiencia del canal. Un nuevo protocolo que usa contención cuando la carga fuera baja y un retardo bajo, una técnica libre de colisiones cuando la carga fuera alta para lograr una buena eficiencia de canal. Los llamaremos protocolos de contención limitada. Los únicos protocolos de contención estudiados han sido simétricos. Cada estación intenta adquirir el canal con cierta probabilidad, p , el desempeño se pueda mejorar mediante el uso de un protocolo que asigna diferentes probabilidades a distintas estaciones. La probabilidad de que una estación adquiera con éxito el canal es $k p(1 - p)^{k-1}$. El valor óptimo de p , obtenemos:

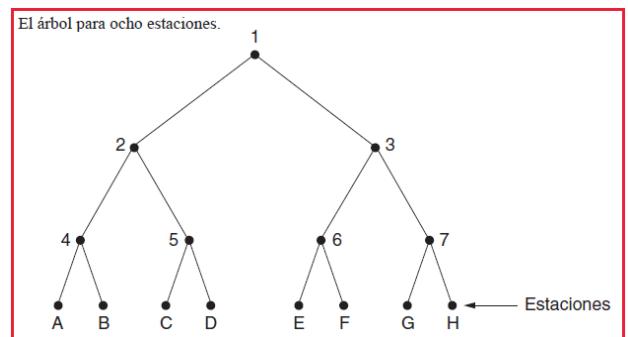
$$\Pr[\text{éxitos con } p \text{ óptima}] = \left[\frac{k-1}{k} \right]^{k-1}$$

Esto se grafica.



Para un número pequeño de estaciones, la posibilidad de éxito es buena, pero la cantidad de estaciones llega a cinco, la probabilidad disminuye hasta $1/e$. La probabilidad de que una estación adquiera el canal sólo puede aumentar si disminuye la cantidad de competencia. Primero dividen las estaciones en grupos, grupo 0 pueden competir por la ranura 0. Si uno de ellos tiene éxito, adquiere el canal

y transmite su trama. Si la ranura permanece desocupada o si hay una colisión, los miembros del grupo 1 compiten por la ranura 1, es posible reducir la cantidad de contenciones para cada ranura se puede operar cada ranura cerca de la parte izquierda de la figura. El truco está en cómo asignar las estaciones a las ranuras. Lo que necesitamos es una manera de asignar dinámicamente las estaciones a las ranuras, con muchas estaciones por ranura cuando la carga es baja y pocas estaciones por ranura cuando la carga es alta.



El protocolo de recorrido de árbol adaptable

Es conveniente considerar a las estaciones como hojas de un árbol binario, en la primera ranura de contención después de la transmisión exitosa de una trama (ranura 0), se permite que todas las estaciones intenten adquirir el canal. Si hay una colisión, durante la ranura 1, sólo aquellas estaciones que queden bajo el nodo 2 podrán competir. Si alguna de ellas adquiere el canal, la ranura que siga se reservará para las estaciones que están bajo el nodo 3, si dos o más estaciones bajo el nodo 2 quieren transmitir, habrá una colisión durante la ranura 1, será el turno del nodo 4 durante la ranura 2. Si ocurre una colisión durante la ranura 0, se examina todo el árbol para localizar todas las estaciones listas. Cada ranura de bits está asociada a un nodo. Si ocurre una colisión, continúa la búsqueda en forma recursiva. Si una ranura de bits está inactiva o si sólo una estación que transmite en ella, se puede detener la búsqueda, ya que se han localizado todas las estaciones listas (si hubiera existido más de una, habría ocurrido una colisión). Cuando la carga es pesada, no vale la pena dedicarle la ranura 0 al nodo 1, sólo tiene sentido en el caso poco probable de que haya precisamente una estación que tenga una trama por enviar. Sería conveniente saltar los nodos 2 y 3 por la misma razón, a mayor carga, la búsqueda debe comenzar más abajo en el árbol.

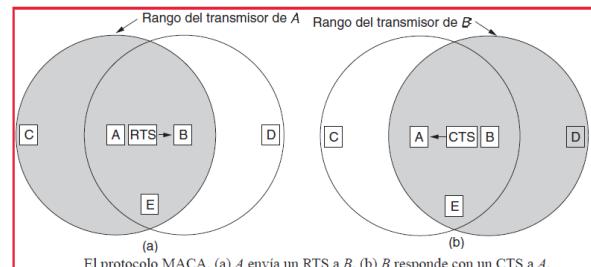
Protocolos de LAN inalámbrica

Un canal de difusión requiere distintos protocolos MAC. Una LAN inalámbrica es un edificio de oficinas con puntos de acceso (AP). Los sistemas inalámbricos no pueden, por lo general, detectar una colisión al momento en que ocurre. La señal recibida en una estación puede ser débil, se utilizan confirmaciones de recepción para descubrir las colisiones y otros errores. Una estación en una LAN inalámbrica no puede transmitir ni recibir tramas de todas las demás estaciones debido al rango de radio limitado de éstas. En las LAN alámbricas, una estación envía una trama, todas las demás estaciones la reciben. La ausencia de esta propiedad en las LAN inalámbricas provoca complicaciones. En los siguientes análisis cada transmisor de radio tiene cierto rango físico, mediante una región de cobertura circular. Una LAN inalámbrica podría ser

probada con CSMA: escuchar si hay otras transmisiones y sólo transmitir si nadie más lo está haciendo. El problema, lo que importa, es la interferencia en el receptor, no en el emisor.

En la figura se ilustran cuatro estaciones inalámbricas.

No importa cuáles son AP ni cuáles son computadoras portátiles. A y C transmiten hacia B, si A envía y C detecta el medio de inmediato, no podrá escuchar a A porque está fuera de su alcance. Por lo tanto, C concluirá falsamente que puede transmitir a B. Si C comienza a transmitir, interferirá en B, eliminando la trama de A. Queremos un protocolo MAC que evite colisión, debido a que desperdicia ancho de banda. El problema de que no pueda detectar a un competidor, debido a que está demasiado lejos, se denomina problema de terminal oculta. Una situación distinta: B transmite a A al mismo tiempo que C desea transmitir a D. Si C detecta el medio, escuchará una transmisión y concluirá equivocadamente que no puede enviar a D, esa transmisión provocaría una mala recepción sólo en la zona entre B y C, donde no hay ninguno de los receptores deseados. Queremos un protocolo MAC que evite aplazamiento, ya que desperdicia ancho de banda. A esta situación se le denomina problema de terminal expuesta.



Uno de los primeros protocolos MACA (Acceso Múltiple con Prevención de Colisiones, del inglés Multiple Access with Collision Avoidance). Se basa es que el emisor estimule al receptor para que envíe una trama corta, de manera que las estaciones cercanas puedan detectar esta transmisión y eviten ellas mismas hacerlo durante la siguiente trama de datos. Se utiliza esta técnica en vez de la detección de portadora. A comienza enviando una trama RTS (Solicitud de Envío, del inglés Request To Send) a B. Esta trama corta (30 bytes) contiene la longitud de la trama de datos que seguirá después, luego B contesta con una trama CTS (Libre para Envío, del inglés Clear To Send), CTS contiene la longitud de los datos (que copia de la trama RTS). Al recibir la trama CTS, A comienza a transmitir. Cualquier estación que escuche el RTS está bastante cerca de A y debe permanecer en silencio durante el tiempo suficiente para que el CTS se transmita de regreso a A sin conflicto. Cualquier estación que escuche el CTS está bastante cerca de B y debe permanecer en silencio durante la siguiente transmisión de datos, cuya longitud puede determinar examinando la trama CTS. C está en el alcance de A pero no en el alcance de B, escucha el RTS de A pero no el CTS de B. En tanto no interfiera con el CTS, está libre para transmitir mientras se envía la trama de datos. D está en el alcance de B pero no de A. No escucha el RTS pero sí el CTS. Al escuchar el CTS sabe que está cerca de una estación que está a punto de recibir una trama, difiere el envío de cualquier cosa hasta el momento en que se espera la terminación de esa trama. E escucha ambos mensajes de control y, al igual que D, debe permanecer en silencio hasta que se haya completado la trama de datos. Aún pueden ocurrir colisiones. B y C podrían enviar tramas RTS a A al mismo tiempo, chocarán y se perderán. En el caso de una colisión, un transmisor que no escucha un CTS, espera un tiempo aleatorio y vuelve a intentar más tarde.

ETHERNET

Existen dos tipos de Ethernet: Ethernet clásica, que resuelve el problema de acceso múltiple mediante el uso de las técnicas que hemos estudiado en este capítulo; el segundo tipo es la Ethernet comutada, los dispositivos llamados switches se utilizan para conectar distintas computadoras, ambas son muy diferentes. La Ethernet clásica es la forma original que operaba a tasas de transmisión de 3 a 10 Mbps. La Ethernet comutada es Ethernet y opera a 100, 1 000 y 10 000 Mbps, conocidas como Fast Ethernet, Gigabit Ethernet y 10 Gigabit Ethernet. Actualmente, se utiliza Ethernet comutada. Ethernet y el IEEE 802.3 son idénticos, excepto por una pequeña diferencia.

Capa física de Ethernet clásica

La Xerox Ethernet fue tan exitosa que DEC, Intel y Xerox idearon un estándar en 1978 para una Ethernet de 10 Mbps, conocido como estándar DIX. Con una modificación menor, el estándar DIX se convirtió en el estándar IEEE 802.3 en 1983. La Ethernet clásica se tendía alrededor del edificio como un solo cable largo.

La primera variedad, conocida como Ethernet gruesa, con marcas cada 2.5 metros para mostrar en dónde conectar las computadoras. Después le siguió la Ethernet delgada, que se doblaba con más facilidad y las conexiones se realizaban mediante conectores BNC, era mucho más económica y fácil de instalar, se podían tender 185 metros por segmento (en vez de los 500 m con la Ethernet gruesa), podía manejar 30 máquinas (en vez de 100). Para permitir redes más grandes, se pueden conectar varios cables mediante repetidores, un dispositivo de capa física que recibe, regenera y retransmite las señales en ambas direcciones. La información se enviaba mediante la codificación Manchester. Podía contener varios segmentos de cable y múltiples repetidores, pero no dos transceptores separados por más de 2.5 km, y no más de cuatro repetidores. La razón era para que el protocolo MAC pudiera funcionar.

El protocolo de subcapa MAC de la Ethernet clásica

El formato utilizado para enviar tramas. Primero viene un Preámbulo de 8 bytes, contiene el patrón de bits 10101010 (con la excepción del último byte, que los últimos 2 bits se establecen a 11). Este último byte se llama delimitador de Inicio de trama en el 802.3.

La codificación de Manchester de este patrón produce una onda cuadrada de 10 MHz durante 6.4 µseg para permitir que el reloj del receptor se sincronice con el del emisor. Los últimos dos bits indican al receptor que está a punto de empezar el resto de la trama.

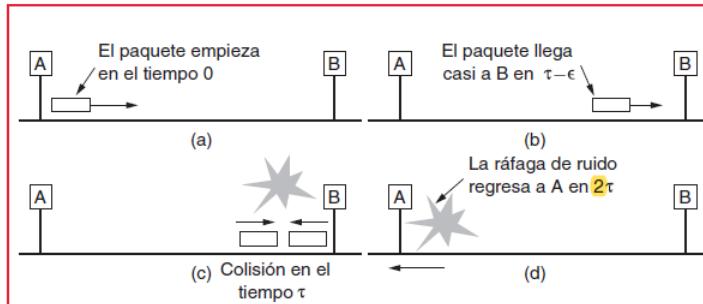
Bytes	8	6	6	2	0-1500	0-46	4
(a)	Preámbulo	Dirección de destino	Dirección de origen	Tipo	Datos	Relleno	Suma de verificación
(b)	Preámbulo	S O F	Dirección de destino	Dirección de origen	Longitud	Datos	Suma de verificación

Figura 4-14. Formatos de trama. (a) Ethernet (DIX). (b) IEEE 802.3.

Después vienen dos direcciones, una para el destino y una para el origen, tienen una longitud de 6 bytes. El primer bit de la dirección de destino es un 0 para direcciones ordinarias y un 1 para direcciones de grupo. Las de grupo permiten que varias estaciones escuchen en una sola dirección. Se llama multidifusión (multicasting). La dirección especial que consiste únicamente en bits 1 está reservada para difusión (broadcasting), se acepta en todas las estaciones de la red. La multidifusión es más selectiva, por el contrario, la difusión no hace ninguna diferencia entre las estaciones. Las direcciones de origen son globalmente únicas; el IEEE las asigna de manera central para asegurar que no haya dos estaciones en el mundo con la misma dirección. La idea es que cualquier estación pueda direccionar de manera exclusiva con sólo dar el número correcto de 48 bits. Se utilizan los primeros 3 bytes para un OUI (Identificador Único Organizacional, del inglés Organizationally Unique Identifier). El IEEE asigna los valores para este campo, e indican un fabricante. se les asignan bloques de 2^{24} direcciones. El fabricante asigna los últimos 3 bytes de la dirección. Tipo o Longitud, dependiendo de si la trama es Ethernet o IEEE 802.3. Ethernet usa un campo Tipo para indicar al receptor qué hacer con la trama. Es posible utilizar múltiples protocolos de capa de red cuando llega una trama el sistema operativo tiene que saber a cuál entregarle la trama. Tipo especifica a qué proceso darle la trama. La IEEE 802.3 decidió que este campo transportaría la longitud de la trama, ya que para determinar la longitud de Ethernet había que ver dentro de los datos; una violación del uso de capas. No había forma de que el receptor averiguara qué hacer con una trama entrante. Para resolver ese problema se agregó otro encabezado para el protocolo LLC (Control de Enlace Lógico, del inglés Logical Link Control) dentro de los datos. Utiliza 8 bytes para transportar los 2 bytes de información del tipo del protocolo. Cuando se publicó el estándar 802.3, había hardware y software para DIX Ethernet en uso que pocos fabricantes y usuarios se esforzaron en reempaquetar. En 1997, el IEEE desistió los campos Tipo que se usaban antes de 1997 tenían valores mayores que 1500, la regla es que cualquier número ahí que sea menor o igual a 0x600 (1536) se puede interpretar como Longitud, y cualquier número mayor de 0x600 se puede interpretar como Tipo.

Después están los datos, de hasta 1500 bytes. Este límite fue elegido en base al hecho de que un transceptor necesita suficiente RAM para mantener toda una trama y era muy costosa en 1978. También hay una longitud mínima, un campo de datos de 0 bytes causa problemas. Cuando un transceptor detecta una colisión, trunca la trama actual, significa que los bits perdidos y las piezas de las tramas aparecen todo el tiempo en el cable. Para distinguir con facilidad las tramas válidas de lo inservible, necesita por lo menos 64 bytes, de la dirección de destino a la suma de verificación, incluyendo ambas. Si la porción de datos de una

trama es menor que 46 bytes, el campo de relleno se utiliza para completar la trama al tamaño mínimo. Otra razón (más importante) para tener una trama de longitud mínima es evitar que una estación complete la transmisión de una trama corta antes de que el primer bit llegue al extremo más alejado del cable, donde podría tener una colisión con otra trama. Si una estación intenta transmitir una trama muy corta, ocurrirá una colisión, pero la transmisión se completará antes de que la ráfaga de ruido llegue de regreso a la estación en 2τ . El emisor supondrá que la trama se envió con éxito. Para evitar esta situación, todas las tramas deberán tardar más de 2τ para enviarse, de manera que la transmisión aún se esté llevando a cabo cuando la ráfaga de ruido regrese al emisor. La LAN de 10 Mbps con una longitud máxima de 2 500 metros y cuatro repetidores (de la especificación 802.3), el tiempo de ida y vuelta es de 50 μ seg en el peor de los casos. A 10 Mbps, un bit tarda 100 nseg, 500 bits es la trama más pequeña que se garantiza funcionará, seguridad, este número se redondeó a 512 bits o 64 bytes. El campo final de es la Suma de verificación, es un CRC de 32 bits, se define mediante el polinomio generador, que funciona también para PPP, ADSL y otros enlaces. Es un código de detección de errores y la trama se desecha si se detecta uno.



CSMA/CD con retroceso exponencial binario

La Ethernet clásica utiliza CSMA/CD persistente-1, las estaciones detectan el medio cuando tienen una trama que desean enviar, y la envían tan pronto como el medio está inactivo. Si hay una colisión, abortan la transmisión y vuelven a transmitir después de un intervalo aleatorio. Tras una colisión, el tiempo se divide en ranuras discretas cuya longitud es igual al tiempo de propagación de ida y vuelta para el peor de los casos 2τ . Despues de la primera colisión, cada estación espera 0 o 1 tiempos de ranura al azar antes de intentarlo de nuevo. Si ambas escogen el mismo número aleatorio, habrá una nueva colisión. Despues cada una escoge 0, 1, 2 o 3 al azar y espera ese tiempo de ranura. Si ocurre una tercera colisión la siguiente vez el número de ranuras a esperar se escogerá al azar del intervalo 0 a $2^3 - 1$. Despues de i colisiones se elige un número aleatorio entre 0 y $2^i - 1$, y se salta ese número de ranuras, al llegar a 10 colisiones el intervalo se congela en un máximo de 1 023 ranuras. Despues de 16 colisiones, el controlador informa a la computadora que fracasó. La recuperación posterior es responsabilidad de las capas superiores. Este algoritmo, se llama retroceso exponencial binario. Al hacer que el intervalo de aleatorización crezca de manera exponencial a medida que ocurren cada vez más colisiones, el algoritmo asegura un retardo pequeño cuando sólo unas cuantas estaciones entran en colisión, asegura que la colisión se resuelva en un intervalo razonable cuando haya muchas. Al truncar el retroceso a 1023, evitamos que el límite crezca demasiado. Si no hay colisión, el emisor supone que se entregó con éxito, ni CSMA/CD ni Ethernet proveen confirmaciones de recepción. Esta es apropiada para los canales de cable de cobre y de fibra óptica. Cualquier error debe detectarse y recuperarse en las capas superiores. Para los canales inalámbricos que tienen más errores, se utilizan confirmaciones de recepción.

Desempeño de Ethernet

La probabilidad A de que una estación adquiera el canal durante esa ranura, la trama promedio tarda P segundos en transmitirse. La distancia máxima de cable entre dos estaciones entra en el cálculo de desempeño. Cuanto mayor sea la longitud del cable, mayor será el intervalo de contención, razón por la cual el estándar Ethernet especifica una longitud máxima de cable.

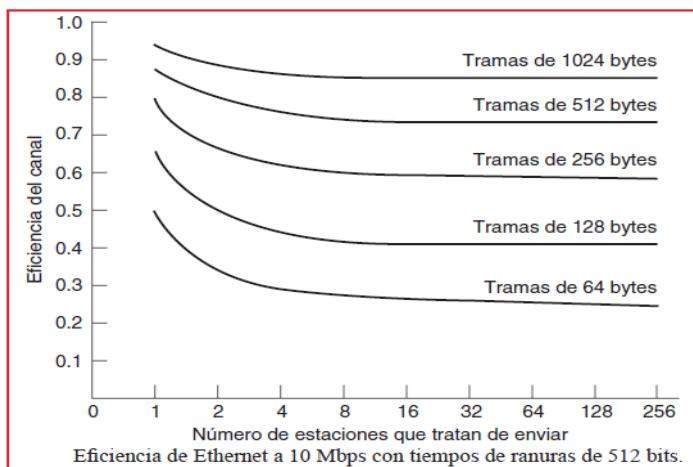
$$\text{Eficiencia del canal} = \frac{P}{P + 2\tau/A}$$

Un aumento en el ancho de banda o la distancia de la red reduce la eficiencia para una trama de un tamaño dado. La gente quiere un gran ancho de banda a través de distancias grandes, quizás la Ethernet clásica

implementada de esta forma no sea el mejor sistema para estas aplicaciones. Se presenta una gráfica de la eficiencia del canal contra el número de estaciones listas para $2T = 51.2 \mu\text{seg}$ y una tasa de transmisión de datos de 10 Mbps, con tramas de 1 024 bytes el periodo de contención tiene 174 bytes y la eficiencia es del 85%.

Ethernet conmutada

Ethernet empezó a evolucionar y el hecho de encontrar interrupciones condujeron hacia un patrón de cableado, en donde cada estación cuenta con un cable dedicado que llega a un hub (concentrador) central, simplemente conecta de manera eléctrica todos los cables que llegan a él, como si estuvieran soldados en conjunto. Los cables eran pares trenzados de la compañía telefónica, se redujo la distancia máxima de cable del hub hasta 100 metros (200 metros pares trenzados categoría 5 de alta calidad). Es más simple agregar o quitar una estación, los cables rotos se pueden detectar con facilidad. Los hubs no incrementan la capacidad debido a que son lógicamente equivalentes al cable extenso individual. A medida que se agregan más estaciones, cada estación recibe una parte cada vez menor de la capacidad fija, la LAN se saturara. Una forma de solucionar esto es usar una velocidad más alta.



Existe otra forma de tratar con el aumento de carga: una Ethernet conmutada. El corazón de este sistema es un conmutador (switch) que contiene un plano posterior (backplane) de alta velocidad, el cual conecta a todos los puertos, contienen de 4 a 48 puertos, cada uno con un conector estándar RJ-45r. Cada cable conecta al switch o hub con una sola computadora. Un switch tiene también las mismas ventajas que un hub. Es fácil agregar o quitar una nueva estación y encontrar la mayoría de las fallas, ya que un cable o puerto defectuoso afectará a una sola estación. Los switches sólo envían tramas a los puertos para los cuales están destinadas, verifica las direcciones de Ethernet para ver cuál es el puerto de destino de la trama. Requiere que el switch sea capaz de deducir qué puertos corresponden a qué direcciones. El switch reenvía la trama a través de su plano posterior de alta velocidad hacia el puerto de destino. El plano posterior opera a muchos Gbps mediante el uso de un protocolo propietario que no necesita estandarización, ya que está completamente oculto. Después, el puerto de destino transmite la trama sobre el cable. ¿Qué ocurre si más de una estación o puerto desea enviar una trama al mismo tiempo?, los switches difieren de los hubs. En un hub, todas las estaciones están en el mismo dominio de colisión. Deben usar el algoritmo CSMA/CD. En un switch, cada puerto es su propio dominio de colisión independiente, el cable es full-dúplex, tanto la estación como el puerto pueden enviar una trama en el cable al mismo tiempo, sin preocuparse por los demás puertos y estaciones. Ahora las colisiones son imposibles y no se necesita CSMA/CD. Pero si el cable es half-dúplex, la estación y el puerto deben competir por la transmisión con CSMA/CD.

Un switch mejora el desempeño de la red en comparación con un hub, como no hay colisiones, la capacidad se utiliza con más eficiencia, con un switch se pueden enviar varias tramas al mismo tiempo. Estas tramas llegarán a los puertos y viajarán hacia el plano posterior para enviarlos por los puertos apropiados, como se podrían enviar dos tramas al mismo puerto de salida y al mismo tiempo, el switch debe tener un búfer para que pueda poner temporalmente en cola una trama de entrada hasta que se pueda transmitir al puerto de salida.

El cambio, en los puertos por donde se envían las tramas incluye beneficios de seguridad, las interfaces de LAN tienen un modo promiscuo, todas las tramas se entregan a cada computadora y no sólo las que van dirigidas a ella. En un hub, cualquier ordenador conectado puede ver el tráfico transmitido. En un switch, el tráfico se reenvía sólo a los puertos a los que está destinado.

Fast Ethernet

Los switches ganaban popularidad, la velocidad de 10 Mbps de Ethernet estaba bajo una presión cada vez mayor. La IEEE convocó al comité 802.3 en 1992 con instrucciones de idear una LAN más rápida. El nuevo

diseño también sería compatible con las versiones previas de redes LAN Ethernet existentes. Todos lo llaman Fast Ethernet en vez de 802.3u. La idea básica era simple: mantener todos los formatos, interfaces y reglas de procedimientos anteriores, pero reducir el tiempo de bits de 100 nseg a 10 nseg, habría sido posible copiar la Ethernet clásica de 10 Mbps y aún detectar colisiones a tiempo con sólo reducir la longitud máxima de cable por un factor de 10. Sin embargo, las ventajas del cableado de par trenzado eran tan abrumadoras que Fast Ethernet se basa por completo en este diseño. Todos los sistemas utilizan hubs y switches, había que tomar decisiones de qué tipos de cable soportar; cable de par trenzado categoría 3. El argumento a su favor era que se podrían cablear las computadoras sin tener que volver a cablear el edificio. La principal desventaja es su incapacidad de transportar 100 Mbps a más de 100 metros. En contraste, el cable de par trenzado categoría 5 puede manejar 100 metros con facilidad, y la fibra puede recorrer mucha más distancia. El compromiso elegido fue permitir las tres posibilidades. UTP categoría 3, llamado 100Base-T4, utilizaba una velocidad de señalización de 25 MHz, 25% más rápida que la Ethernet estándar, para alcanzar la tasa de bits necesaria, 100Base-T4 requiere cuatro cables de par trenzado, uno siempre va al hub, uno siempre sale del hub y los otros dos se pueden comutar a la dirección actual de la transmisión. Para obtener 100 Mbps de los tres pares trenzados en la dirección de la transmisión, implica enviar dígitos ternarios con tres distintos niveles de voltaje. Esto significa renunciar al teléfono de su oficina. 100Base-T4 quedó al borde del camino debido a que se actualizó el cableado por UTP categoría 5 para Ethernet 100Base-TX. Este diseño es más simple puesto que los cables pueden manejar velocidades de reloj de 125 MHz, se utilizan dos pares trenzados por estación, uno que va al hub y otro que viene de él. Se utiliza la codificación 4B/5B. Se codifican 4 bits de datos como 5 bits de señal y se envían a 125 MHz para proveer 100 Mbps. Tiene suficientes transiciones para la sincronización, utiliza muy bien el ancho de banda del cable, es full-duplex, pueden transmitir a 100 Mbps en un par trenzado y recibir en el otro par al mismo tiempo.

100Base-FX, utiliza dos filamentos de fibra multimodo, una para cada dirección, es full-dúplex con 100 Mbps en cada dirección. La distancia

Nombre	Cable	Segmento máximo	Ventajas
100Base-T4	Par trenzado	100 m	Utiliza UTP categoría 3.
100Base-TX	Par trenzado	100 m	Full-dúplex a 100 Mbps (UTP cat 5).
100Base-FX	Fibra óptica	2000 m	Full-dúplex a 100 Mbps; distancias largas.

entre una estación y el switch puede ser de hasta 2 km. Para asegurar que el algoritmo CSMA/CD siga trabajando, es necesario mantener la relación entre el tamaño mínimo de trama y la longitud máxima del cable a medida que la velocidad de la red aumenta de 10 Mbps a 100 Mbps. Así, el tamaño mínimo de trama de 64 bytes debe aumentar o la longitud máxima de cable de 2 500 debe disminuir. La elección fácil fue reducir la distancia máxima entre dos estaciones cualesquiera por un factor de 10, los cables 100Base-FX de 2 km son demasiado largos como para permitir un hub de 100 Mbps con el algoritmo de colisiones normal. Estos cables se deben conectar a un switch y operar en un modo full-dúplex para que no haya colisiones. Los usuarios no deseaban tirar las tarjetas Ethernet de 10 Mbps en las computadoras antiguas, casi todos los switches Fast Ethernet pueden manejar una mezcla de estaciones de 10 Mbps y 100 Mbps. Este provee por sí solo un mecanismo llamado autonegociación, el cual permite que dos estaciones negocien de manera automática la velocidad óptima y la duplicidad.

Gigabit Ethernet

Los objetivos eran en esencia los mismos que los del comité para Fast Ethernet: que tuviera un desempeño 10 veces mayor y que mantuviera la compatibilidad, mantener el mismo formato de trama, incluyendo los tamaños mínimo y máximo de trama. Las configuraciones de Gigabit Ethernet usan enlaces punto a punto, el caso más común es tener un switch o un hub conectado a varias computadoras y quizás a switches o hubs adicionales, cada cable Ethernet individual tiene exactamente dos dispositivos en él. Soporta dos modos diferentes de funcionamiento: modo full-dúplex y modo half-dúplex. El modo "normal" es el modo full-dúplex, que permite tráfico en ambas direcciones al mismo tiempo. Cuando hay un switch central conectado a computadoras todas las líneas se almacenan en el búfer con el fin de que cada computadora y switch pueda enviar tramas siempre que lo desee. Debido a que no hay contención, no se utiliza el protocolo CSMA/CD y la longitud máxima del cable se determina con base en los aspectos relacionados con la fuerza de la señal, en vez de basarse en el tiempo que tarda una ráfaga de ruido en propagarse de vuelta al emisor. Los switches tienen la libertad de mezclar e igualar velocidades. El otro modo de operación es half-dúplex y

se utiliza cuando las computadoras están conectadas a un hub en vez de un switch. Un hub no almacena las tramas entrantes. Puede haber colisiones, por lo que se requiere el protocolo CSMA/CD estándar, se puede transmitir una trama de 64 bytes, 100 veces más rápido que en la Ethernet clásica, la longitud máxima del cable debe ser 100 veces menor, o de 25 metros. Esta restricción de longitud fue tan dolorosa que se agregaron dos características al estándar para incrementar la longitud máxima del cable a 200 metros. La primera, llamada extensión de portadora, indica al hardware que agregue su propio relleno después de la trama normal para extenderla a 512 bytes. El hardware emisor agrega este relleno y el hardware receptor lo elimina, el software no toma parte en esto. La desventaja es que tiene una eficiencia muy baja. La segunda, llamada ráfagas de trama, permite que un emisor transmita una secuencia concatenada de múltiples tramas en una sola transmisión, si hay suficientes tramas esperando su transmisión. Gigabit Ethernet soporta tanto el cableado de cobre como el de fibra óptica, requiere codificar y enviar un bit cada nanosegundo.

En estas versiones de Gigabit Ethernet, la codificación 8B/10B codifica 8 bits de datos en palabras codificadas de 10 bits. Las palabras codificadas se eligieron de modo que se pudieran balancear con suficientes transiciones para la recuperación del reloj. Todas estas opciones requerían nuevos cables de cobre o de fibra. Ninguna de ellas hizo uso de la gran cantidad de cable UTP categoría 5 llegó 1000Base-T para llenar este vacío. Se necesita una señalización más complicada. Se utilizan los cuatro pares trenzados en el cable, y cada par se utiliza en ambas direcciones al mismo tiempo mediante el uso de un procesamiento de señales digitales para separar las señales. Se utilizan cinco niveles de voltaje que transportan 2 bits para una señalización y se requiere un mezclado (scrambling) para las transiciones, seguido de un código de corrección de errores. Gigabit Ethernet soporta el control de flujo consiste en que un extremo envíe una trama de control especial al otro extremo para indicarle que se detenga durante cierto periodo, tramas de control PAUSE. Las tramas Jumbo que permiten que las tramas tengan una longitud mayor de 1500 bytes, hasta 9 KB. No la reconoce el estándar, pero la mayoría de los distribuidores la soporta. La tasa de transmisión de tramas se puede reducir, junto con el procesamiento asociado, como interrumpir al procesador para decir que llegó una trama, o dividir y recombinar mensajes que eran demasiado largos.

10 Gigabit Ethernet

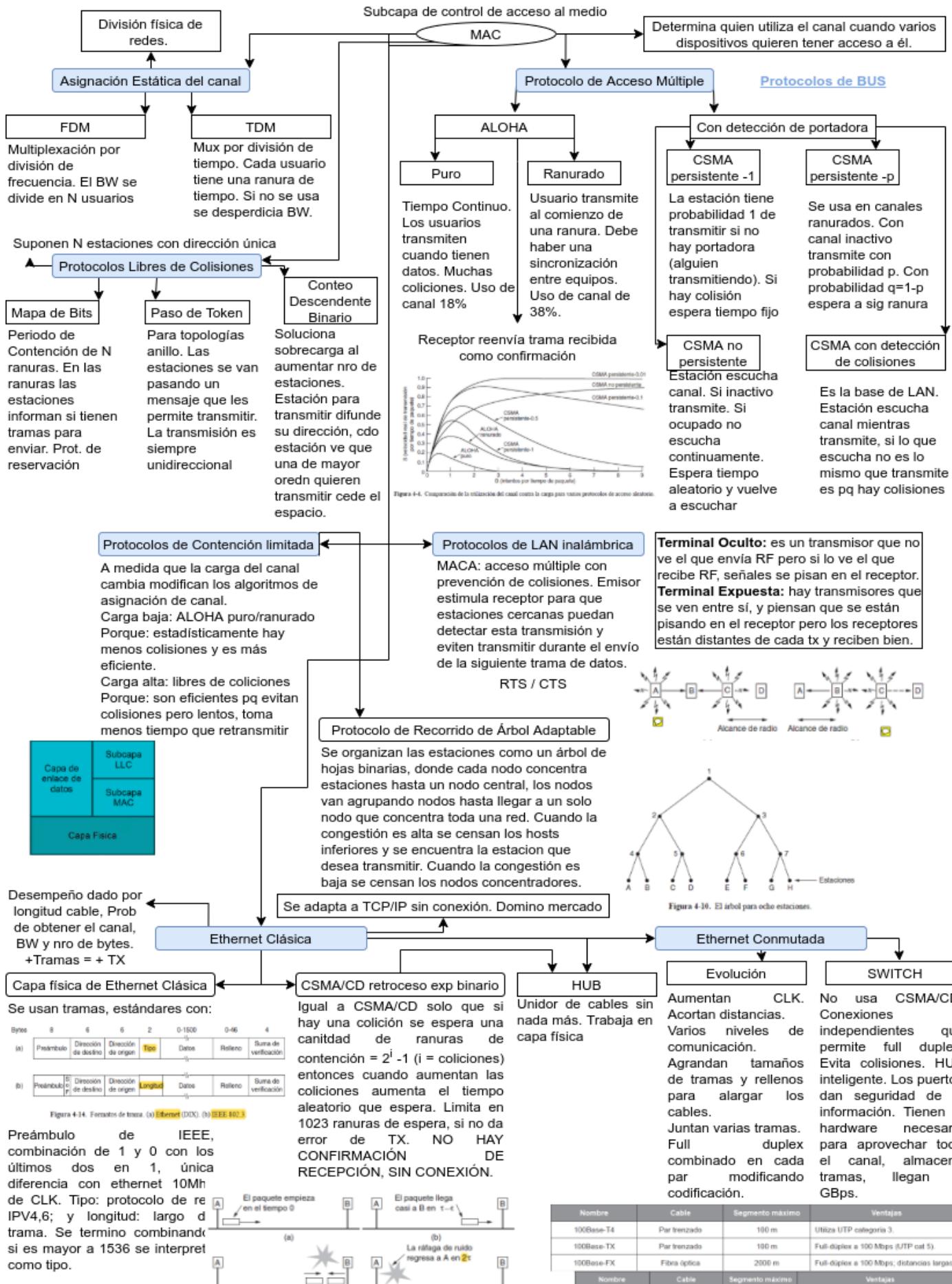
Aparecieron estándares

Nombre	Cable	Segmento máximo	Ventajas
1000Base-SX	Fibra óptica	550 m	Fibra multimodo (50, 62.5 micras)
1000Base-LX	Fibra óptica	5000 m	Monomodo (10 μ) o multimodo (50, 62.5μ)
1000Base-CX	2 pares de STP	25 m	Par trenzado blindado
1000Base-T	4 pares de UTP	100 m	UTP estándar categoría 5

para fibra y cable de cobre blindado, 1 000 veces más rápida que la Ethernet original. Se podría necesitar dentro de los centros e intercambios de datos para conectar enruteadores, switches y servidores de gama alta, troncales de larga distancia con alto ancho de banda, redes de área metropolitana. Las conexiones de larga distancia usan fibra óptica, mientras que las conexiones cortas pueden usar cobre o fibra, soportan sólo la operación full-dúplex. CSMA/CD ya no forma parte del diseño y los estándares se concentran en las capas físicas, la compatibilidad aún sigue siendo importante, usan la autonegociación. Se utiliza fibra multimodo y la fibra monomodo, envían un flujo serial de información que se produce mediante el mezclado de los bits de datos, después codificarlos mediante un código 64B/66B, esta tiene menos sobrecarga que un código 8B/10B. 10GBase-T es la versión que usa cables UTP. Requiere cableado categoría 6a, en distancias más cortas puede usar categoría 5. Cada uno de los cuatro pares trenzados se utiliza para enviar 2 500 Mbps en ambas direcciones. Para llegar a esta velocidad se utiliza una tasa de señalización de 800 M símbolos/seg, con símbolos que usan 16 niveles de voltaje, se protegen con un código LDPC (Verificación de Paridad de Baja Densidad, del inglés Low Density Parity Check). La IEEE creó un grupo para estandarizar la Ethernet que opera a 40 Gbps y 100 Gbps, permitirá competir en ambientes de muy alto rendimiento.

Retrospectiva de Ethernet

Ethernet interactúa fácilmente con TCP/IP, el cual se ha vuelto dominante. IP es un protocolo sin conexión, por lo que se adapta muy bien a Ethernet, que también es sin conexión. Junto con ATM, esta lista incluye la FDDI (Interfaz de Datos Distribuidos por Fibra, del inglés Fiber Distributed Data Interface) y el Canal de fibra (dos redes LAN ópticas basadas en anillos).



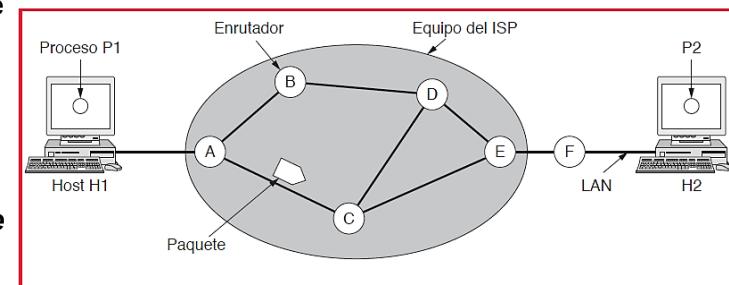
Capa de Red

Se encarga de llevar los paquetes todo el camino, desde el origen hasta el destino, tal vez sea necesario realizar muchos saltos por enrutadores. Contrastá con la de la capa de enlace de datos, cuya única meta es mover tramas de un extremo del cable al otro. Debe conocer la topología de la red y elegir las rutas apropiadas, debe tener cuidado al escoger las rutas para no sobrecargar las líneas y dejar inactivos a otros. Cuando el origen y el destino están en redes diferentes, la capa de red es la encargada de solucionarlo.

ASPECTOS DE DISEÑO DE LA CAPA DE RED

Comunicación de paquetes de almacenamiento y reenvío

Los componentes principales de la red son el equipo del Proveedor del Servicio de Internet (ISP) dentro del óvalo sombreado, y el equipo de los clientes, fuera del óvalo. H1 conectado de manera directa a un enrutador del ISP, A, en forma de una computadora en el hogar conectada a un módem DSL. H2 en una LAN Ethernet de oficina con un enrutador, F, propiedad del cliente. Este tiene una línea alquilada que va al equipo del ISP. F fuera del óvalo porque no pertenece al ISP. Para los fines de este capítulo, los enrutadores locales se consideran parte de la red del ISP. Un host que desea enviar un paquete lo transmite al enrutador más cercano, se almacena ahí hasta que haya llegado por completo y el enlace haya terminado su procesamiento comprobando la suma de verificación. Despues se reenvía al siguiente enrutador, en donde se entrega. Este mecanismo se denomina **comunicación de almacenamiento y envío**.



Servicios proporcionados a la capa de transporte

La capa de red proporciona servicios a la capa de transporte en la interfaz entre la capa de red y de transporte. Tiene los siguientes objetivos:

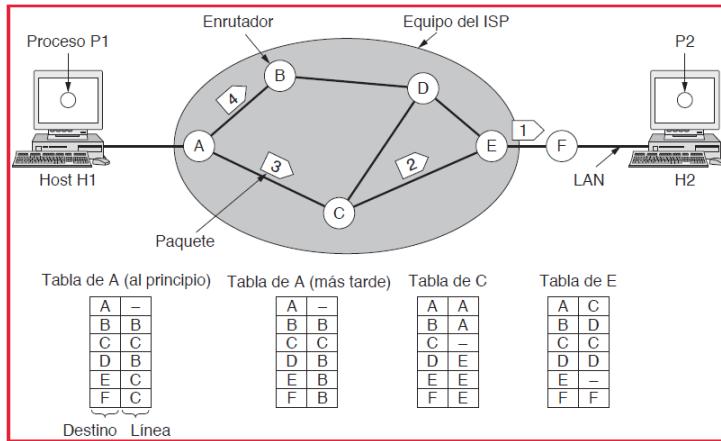
1. Los servicios deben ser independientes de la tecnología del enrutador.
2. La capa de transporte debe estar aislada de la cantidad, tipo y topología de los enrutadores presentes.
3. Las direcciones de red disponibles para la capa de transporte que deben usar un plan de numeración uniforme, incluso a través de redes LAN y WAN.

Los diseñadores de la capa de red tienen mucha libertad de los servicios que se ofrecerán a la capa de transporte. Esta libertad degenera en una discusión, se centra en determinar si la capa de red debe proporcionar un servicio orientado a conexión o un servicio sin conexión. La comunidad de Internet declara que la tarea de los enrutadores es mover paquetes de un lado a otro, y nada más. La red es de naturaleza no confiable, sin importar su diseño, los hosts deben aceptar este hecho y efectuar ellos mismos el control de errores y el control de flujo. Este punto de vista conduce a la conclusión de que el servicio de red debe ser sin conexión y debe contar tan sólo con las primitivas SEND PACKET y RECEIVE PACKET. No debe efectuarse ningún ordenamiento de paquetes ni control de flujo, un ejemplo del argumento extremo a extremo (end-to-end argument), ha sido muy influyente para dar forma a Internet. Cada paquete debe llevar la dirección de destino completa. El otro bando, las compañías telefónicas, argumenta que la red debe proporcionar un servicio confiable, orientado a conexión, la calidad del servicio es el factor dominante, es muy difícil de alcanzar para el tráfico de tiempo real como la voz y el video. La popularidad de las capas de red sin conexión ha aumentado en forma considerable, el protocolo IP es un símbolo constante de éxito. Una tecnología orientada a conexión llamada ATM, se usa en nichos. Dos ejemplos de tecnologías orientadas a conexión son MPLS (Comunicación Multiprotocolo Mediante Etiquetas) y las redes VLAN.

Implementación del servicio sin conexión

Servicio sin conexión, los paquetes se transmiten por separado en la red y se enrutan de manera independiente. No se necesita una configuración por adelantado. Los paquetes se conocen como

datagramas y la red se conoce como red de datagramas. Si se utiliza el servicio orientado a conexión, hay que establecer una ruta del enrutador de origen al enrutador de destino antes de poder enviar cualquier paquete de datos, se conoce como VC (circuito virtual), y la red se denomina red de circuitos virtuales. En una red de datagramas, el proceso P1 tiene un mensaje largo para P2. Dicho proceso entrega el mensaje a la capa de transporte y le indica a ésta que lo envíe al proceso P2 en el host H2. El código de la capa de transporte se ejecuta en H1, dentro del sistema operativo, agrega un encabezado de transporte al frente del mensaje y entrega el resultado a la capa de red, otro procedimiento dentro del sistema operativo.



Para este ejemplo, que el mensaje es cuatro veces más largo que el tamaño máximo, la capa de red tiene que dividirlo en cuatro paquetes: 1, 2, 3 y 4; y enviar cada uno por turnos al A mediante algún protocolo punto a punto; entra en acción el ISP. Cada enrutador tiene una tabla interna que le indica a dónde enviar paquetes. Cada entrada en la tabla es un par que consiste en un destino y la línea de salida, se pueden utilizar líneas conectadas en forma directa. A sólo tiene dos líneas de salida (B y C) cada paquete entrante se debe enviar a uno de estos, incluso si el destino final es algún otro enrutador. En A, 1, 2 y 3 se almacenan unos momentos, después de haber llegado y de haber comprobado sus sumas de verificación. Se reenvía de acuerdo con la tabla de A, por el enlace de salida a C dentro de una nueva trama. Después, el 1 se reenvía a E y después a F. Cuando llega a F, se envía dentro de una trama a H2 a través de la LAN, 2 y 3 siguen la misma ruta. Con el paquete 4 llega a A se envía al enrutador B, está destinado a F, A decidió enviar el paquete 4 por una ruta diferente había alguna congestión de tráfico en alguna parte de la ruta ACE y actualizó su tabla de enrutamiento. El algoritmo que maneja las tablas y realiza las decisiones, se conoce como algoritmo de enrutamiento. IP (Protocolo Internet), que constituye la base de Internet, es el ejemplo dominante de un servicio de red sin conexión. Cada paquete transporta una dirección IP de destino que los enrutadores usan para reenviar cada paquete por separado. Las direcciones son de 32 bits en los paquetes IPv4 y de 128 bits en los paquetes IPv6.

Implementación del servicio orientado a conexión

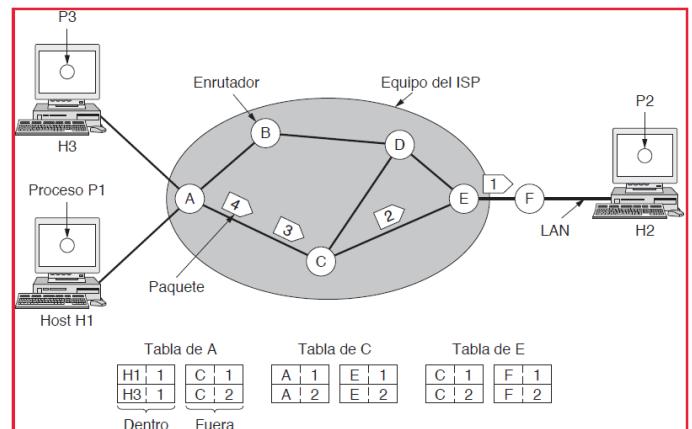
Para el servicio orientado a conexión necesitamos una red de circuitos virtuales. La idea es evitar la necesidad de elegir una nueva ruta para cada paquete enviado, cuando se establece una conexión, se elige una ruta de la máquina de origen a la máquina de destino como parte de la configuración de conexión y se almacena en tablas dentro de los enrutadores. Esa ruta se utiliza para todo el tráfico que fluye a través de la conexión. Cuando se libera la conexión, también se termina el circuito virtual. Cada paquete lleva un identificador que indica a cuál circuito virtual pertenece. H1 ha establecido una conexión 1 con el host H2, se recuerda como la primera entrada en cada una de las tablas de enrutamiento. La primera línea de la tabla A indica que, si un paquete con el identificador de conexión 1 viene de H1, se enviará al enrutador C y se le dará el identificador de conexión 1, la primera entrada en C enruta el paquete a E, también con el identificador de conexión 1. Si H3 también desea establecer una conexión con H2. Elige el identificador de conexión 1, está iniciando la conexión y ésta es su única conexión e indica a la red que establezca el circuito virtual. Esto nos lleva a la segunda fila de las tablas. Surge un problema, A, sí puede saber con facilidad cuáles paquetes de conexión 1 provienen de H1 y cuáles provienen de H3, C no. Por esta razón, A asigna un identificador de conexión diferente al tráfico de salida, los enrutadores necesitan la habilidad de reemplazar identificadores de conexión en los paquetes de salida. A este proceso se le conoce como conmutación mediante etiquetas. MPLS (Comutación Multiprotocolo Mediante Etiquetas, del inglés MultiProtocol Label

Switching), es un servicio de red orientado a conexión. Se utiliza dentro de las redes de ISP en Internet, los paquetes IP se envuelven en un encabezado MPLS tiene un identificador de 20 bits. MPLS se oculta de los clientes y el ISP establece conexiones de largo plazo para grandes cantidades de tráfico, se utiliza cuando la calidad del servicio es importante.

Comparación entre las redes de circuitos virtuales

Existen ventajas y desventajas entre los circuitos virtuales y los datagramas. Tiene que ver con el tiempo de configuración y el tiempo de análisis de la dirección. Los circuitos virtuales requieren una fase de configuración que necesita tiempo y recursos, una vez que se paga este precio, es fácil averiguar qué hacer con un paquete de datos en una red de circuitos virtuales. En una red de datagramas no se requiere configuración, pero se requiere un procedimiento más complicado para localizar la entrada correspondiente al destino, las direcciones de destino que se utilizan en las redes de datagramas son más largas que los números de los circuitos que se utilizan en las redes de circuitos virtuales, si los paquetes tienden a ser bastante cortos, incluir una dirección de destino completa puede representar una sobrecarga y, un desperdicio de ancho de banda.

Otro aspecto más es la cantidad de espacio de tabla requerido en la memoria del enrutador. Una red de datagramas necesita tener una entrada para cada destino posible, mientras que una red de circuitos virtuales sólo necesita una entrada para cada circuito virtual. Esta ventaja es engañosa, los paquetes de configuración de conexión también tienen que enrutar y utilizan direcciones de destino, al igual que los datagramas. Los circuitos virtuales garantizan la calidad del servicio y evitar congestiones en la red, ya que los recursos se pueden reservar. Una vez que comienzan a llegar los paquetes, el ancho de banda y la capacidad de enruteamiento ya estarán disponibles. En una red de datagramas es más difícil evitar la congestión. En los sistemas de procesamiento de transacciones la sobrecarga requerida para establecer y terminar un circuito virtual puede ocupar mucho más tiempo que el uso real del circuito, aquí los circuitos virtuales tienen poco sentido. Para usos en donde el tiempo de ejecución sea extenso, pueden ser de utilidad los circuitos virtuales permanentes que duran meses o años. También tienen un problema de vulnerabilidad. Si falla un enrutador y pierde su memoria, todos los circuitos virtuales que pasan por él tendrán que abortarse. Si un enrutador de datagramas falla, sólo sufirán los usuarios cuyos paquetes se pusieron en cola en el enrutador en ese momento, el emisor los retransmita poco tiempo después. La pérdida de una línea de comunicación es fatal para los circuitos virtuales, pero se puede compensar con facilidad si se usan datagramas, permiten que los enrutadores balanceen el tráfico a través de la red, ya que las rutas se pueden cambiar.



Asunto	Red de datagramas	Red de circuitos virtuales
Configuración del circuito.	No necesaria.	Requerida.
Direccionamiento.	Cada paquete contiene la dirección de origen y de destino completas.	Cada paquete contiene un número de CV corto.
Información de estado.	Los enrutadores no contienen información de estado sobre las conexiones.	Cada CV requiere espacio de tabla del enrutador por cada conexión.
Enrutamiento.	Cada paquete se enruta de manera independiente.	La ruta se elige cuando se establece el CV; todos los paquetes siguen esa ruta.
Efecto de fallas del enrutador.	Ninguno, excepto para paquetes perdidos durante una caída.	Terminan todos los CVs que pasaron por el enrutador defectuoso.
Calidad del servicio.	Difícil.	Fácil si se pueden asignar suficientes recursos por adelantado para cada CV.
Control de congestión.	Difícil.	Fácil si se pueden asignar suficientes recursos por adelantado para cada CV.

INTERCONEXIÓN DE REDES

Hemos supuesto de manera implícita que hay una sola red homogénea, cada máquina usa el mismo protocolo en cada capa, este supuesto es demasiado optimista. Existen PAN, LAN, MAN y WAN. Hay numerosos protocolos en cada capa. Sería más simple unir redes si todos usaran una sola tecnología de red. El propósito de unir todas estas redes es permitir que los usuarios en cualquiera de ellas se comuniquen con los usuarios de las otras redes. Por lo general las redes difieren en formas importantes, debemos lidiar con problemas de heterogeneidad. La conciliación de estas diferencias es lo que hace más difícil la interconexión de redes.

Cómo difieren las redes

Las redes pueden diferir de muchas maneras, como técnicas de modulación o formatos de tramas diferentes, se encuentran en la capa física y en la de enlace de datos. Los paquetes enviados por una fuente en una red deben transitar a través de más redes foráneas antes de llegar a la red de destino, pueden ocurrir problemas en las interfaces, ejemplo, la fuente está en una red Ethernet y el destino en una red WiMAX. Los paquetes pasarían de una red sin conexión a una orientada a conexión, tal vez sea necesario establecer una conexión improvisada, lo cual introduce un retardo y mucha sobrecarga, debemos tener en cuenta muchas diferencias específicas. Los diferentes tamaños máximos de paquete usados por las diferentes redes también pueden ser una gran molestia. Si los paquetes en una red orientada a conexión transitan en una red sin conexión, pueden llegar en un orden distinto al que se enviaron. Estos tipos de diferencias se pueden enmendar, por ejemplo, con una puerta de enlace que une dos redes podría generar paquetes separados para cada destino en vez de un mejor soporte para la multidifusión. Un paquete extenso se podría dividir, enviar en piezas y unir de vuelta. Los receptores podrían colocar los paquetes en un búfer y entregarlos en orden.

Las redes también pueden diferir en la calidad del servicio. Si una red tiene una QoS sólida y la otra ofrece un servicio del mejor esfuerzo, será imposible hacer garantías de ancho de banda, sólo se puedan hacer mientras la red del mejor esfuerzo opere con poco uso, no es muy probable que sea el objetivo de la mayoría de los ISP. Los mecanismos de seguridad son problemáticos.

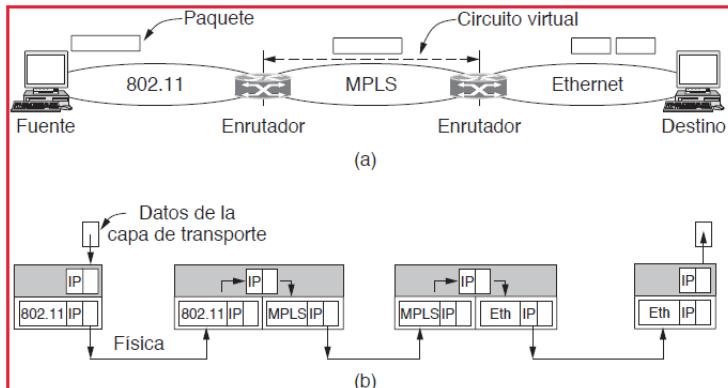
Cómo se pueden conectar las redes

Dos opciones básicas, podemos construir dispositivos que traduzcan o conviertan los paquetes de cada tipo de red en paquetes para otra red o, tratar de resolver el problema al agregar una capa de indirección y construir una capa común encima de las distintas redes. Los dispositivos se colocan en los límites entre las redes. Una capa común para ocultar las diferencias de las redes existentes ha tenido un gran éxito, se separó eventualmente en los protocolos TCP e IP. IP proporciona un formato de paquete universal que todos los enruteadores reconocen y que se puede pasar casi por cualquier red. IP ha extendido su alcance a la red telefónica, redes de sensores y en otros dispositivos. Hay varios dispositivos diferentes que conectan redes, repetidores, hubs, switches, puentes, enruteadores y puertas de enlace. Los repetidores y hubs sólo desplazan bits de un cable a otro, son dispositivos analógicos y no comprenden nada sobre los protocolos de las capas superiores. Los puentes y switches operan en la capa de enlace. Se pueden usar para construir redes, pero sólo con una pequeña traducción de protocolos. Nuestro enfoque es en los dispositivos de interconexión, los enruteadores; las puertas de enlace son dispositivos de interconexión de las capas superiores. Explicaremos la forma en que se puede usar la interconexión con una capa de red común para interconectar redes distintas.

En la figura (a) se muestra una interred compuesta por redes 802.11, MPLS y Ethernet, la máquina fuente en la red 802.11 desea enviar un paquete a la máquina de destino en la red Ethernet, estas tecnologías son distintas y están separadas por una red (MPLS), se requiere un procesamiento adicional en los límites entre las redes. Pueden tener distintas formas de direccionamiento, el paquete transporta una dirección de capa de red que puede identificar cualquier host a través de las tres redes. El primer límite es cuando cambia de una red 802.11 a una red MPLS. 802.11 proporciona un servicio sin conexión, pero MPLS provee un servicio orientado a conexión, se debe establecer un circuito virtual para cruzar esa red. Una vez que el paquete viaje por el circuito virtual, llegará a la red Ethernet. Tal vez el paquete sea demasiado grande como para transportarlo, ya que 802.11 puede trabajar con tramas más grandes que Ethernet. Para manejar este

problema, el paquete se divide en fragmentos y cada fragmento se envía por separado. Los fragmentos llegan a su destino, se vuelven a ensamblar.

En (b) muestra el procesamiento de los protocolos. La fuente acepta los datos de la capa de transporte y genera un paquete con el encabezado de la capa de red común, es IP. El encabezado contiene la dirección de destino final, para determinar que se debe enviar el paquete a través del primer enrutador, se encapsula en una trama 802.11, cuyo destino es el primer enrutador. En el enrutador, se extrae del campo de datos de la trama y se descarta el encabezado de la trama 802.11, examina la dirección IP y busca esta dirección en su tabla de enrutamiento. Con base en esta dirección, decide enviar a continuación el paquete al segundo enrutador. Para esta parte es necesario establecer un circuito virtual MPLS hacia el segundo enrutador y encapsular el paquete con encabezados MPLS. En el extremo se descarta el encabezado MPLS y de nuevo se consulta la dirección de red el destino, el paquete es demasiado largo para enviarlo a través de Ethernet, se divide en dos partes y se coloca en el campo de datos de una trama Ethernet y se envía a la dirección Ethernet del destino. Ya en el destino, se elimina el encabezado Ethernet de cada una de las tramas y se vuelve a ensamblar el contenido. El paquete ha llegado a su destino.



Con un enrutador, el paquete se extrae de la trama, y la dirección de red del paquete se utiliza para decidir a dónde enviarlo. Con un switch (o puente), toda la trama se transporta con base en su dirección MAC. Los switches no tienen que entender el protocolo de capa de red que se utiliza para conmutar los paquetes. Los enrutadores sí tienen que hacerlo. Cuando se introdujeron los puentes, la intención era que unieran distintos tipos de redes, debían traducir tramas de una LAN en tramas de otra LAN. Esto no funcionó bien debido a las diferencias en las características de las LAN (distintos tamaños máximos, con y sin clases de prioridad) son difíciles de enmascarar. El uso más común es para conectar el mismo tipo de red en la capa de enlace, y los enrutadores conectan distintas redes en la capa de red. La interconexión de redes sólo funciona cuando hay una capa de red común. Es difícil lograr que todos estén de acuerdo.

Un enrutador que puede manejar múltiples protocolos de red se denomina enrutador multiprotocolo, debe traducir los protocolos o dejar una conexión para una capa de protocolo superior. Ninguna de las dos es totalmente satisfactoria. Para una conexión en una capa superior, mediante el uso de TCP, se requiere que todas las redes implementen TCP, se limita el uso en las redes a las aplicaciones que usan TCP y no incluye a muchas aplicaciones en tiempo real. La alternativa es traducir los paquetes entre las redes, a menos que los formatos de los paquetes sean muy similares y tengan los mismos campos de información; siempre serán incompletas y con frecuencia estarán destinadas al fracaso. La conversión sólo se intenta raras veces. IP ha funcionado tan bien que se debe a que sirve como un tipo de mínimo común denominador. Requiere muy poco de las redes y ofrece como resultado sólo un servicio del mejor esfuerzo.

Tunelización

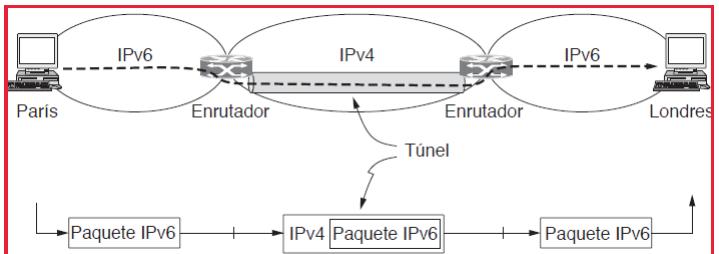
Es difícil manejar el caso general de hacer que dos redes distintas se interconecten, hay un caso especial que se puede manejar, cuando el host de origen y el de destino están en el mismo tipo de red, pero hay una red diferente en medio. Tenemos una red IPv6 en París, una en Londres y una conectividad entre las oficinas a través de la Internet IPv4. Es una técnica llamada tunelización (tunneling). Para enviar un paquete IP a un host en la oficina de Londres, un host de París construye el paquete que contiene una dirección IPv6 en Londres y la envía al enrutador multiprotocolo. Este enrutador recibe el paquete IPv6, lo encapsula con un encabezado IPv4 dirigido al lado IPv4 del enrutador multiprotocolo que se conecta con la red IPv6 de Londres, coloca un paquete (IPv6) dentro de un paquete (IPv4). Cuando llega este paquete envuelto, el enrutador de Londres extrae el paquete IPv6 original y lo envía hacia el host de destino, IPv4 es como un gran túnel que se extiende de un enrutador multiprotocolo al otro, IPv6 simplemente viaja de un extremo del túnel al otro. No tiene que preocuparse por lidiar con IPv4, tampoco los hosts en París o en Londres, solo

los enruteadores multiprotocolo tienen que entender los paquetes IPv4 e IPv6. Se utiliza mucho para conectar hosts y redes aisladas mediante el uso de otras redes. Se le denomina red superpuesta (overlay), realmente está superpuesta sobre la red base.

Se indica en nuestro ejemplo de “IPv6 sobre IPv4”. La desventaja de la tunelización es que no se puede llegar a ninguno de los hosts en la red que se tuneliza debido a que los paquetes no pueden escapar a mitad del túnel. Esta limitación de los túneles se convierte en una ventaja gracias a las redes VPN (Redes Privadas Virtuales, del inglés Virtual Private Network). Una VPN es simplemente una red superpuesta que se utiliza para proporcionar una medida de seguridad.

Enrutamiento entre redes

El enruteamiento a través de una interred presenta el mismo problema que el enruteamiento en una sola red, pero con algunas complicaciones adicionales, las redes pueden usar internamente distintos algoritmos de enruteamiento. Un enruteamiento por estado del enlace y otra por vector de distancia. Los



algoritmos de estado del enlace necesitan conocer la topología, pero los de vector de distancia no, dificultaría mucho el proceso de encontrar las rutas más cortas. Las redes manejadas por distintos operadores conducen a problemas más grandes, tal vez los operadores tengan distintas ideas sobre lo que sería una buena ruta a través de la red, quizás un operador quiera la ruta con el menor retardo, mientras que otro prefiera la ruta menos costosa, las rutas más cortas en la interred no estarán bien definidas. Tal vez un operador no quiera que otro conozca siquiera los detalles de las rutas en su red, la interred puede ser mucho más grande que cualquiera de las redes que la conforman. Puede requerir algoritmos de enruteamiento que escalen el uso de una jerarquía.

Todas conducen a un algoritmo de enruteamiento de dos niveles, se utiliza un protocolo intradominio o de puerta de enlace interior para el enruteamiento. Podría ser un protocolo de estado del enlace. Todas las redes pueden usar distintos protocolos intradominio, pero deben usar el mismo protocolo interdominio, se denomina BGP (Protocolo de Puerta de Enlace de Frontera, del inglés Border Gateway Protocol). Cada red se opera de manera independiente, se conoce como un AS (Sistema Autónomo, del inglés Autonomous System). Una red de un ISP es un modelo de un AS. Los dos niveles no son estrictamente jerárquicos, pues se podrían generar rutas muy subóptimas. Esto ayuda a lidiar con todas las complicaciones. Mejora el escalamiento y permite a los operadores elegir libremente las rutas dentro de sus propias redes. En Internet, consiste en los arreglos comerciales entre los ISP. Cada ISP puede cobrar o recibir dinero de los otros ISP por transportar tráfico, si el enruteamiento de la interred requiere cruzar límites internacionales, pueden entrar en juego varias leyes.

Fragmentación de paquetes

Cada red o enlace impone un tamaño máximo a sus paquetes. Estos límites tienen varias razones:

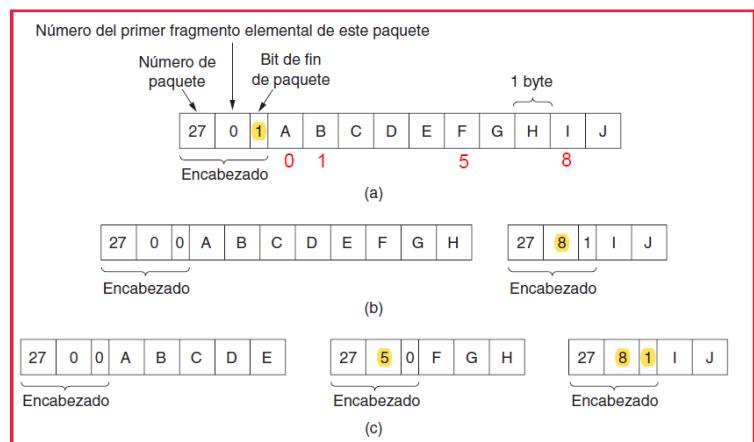
1. El hardware (el tamaño de una trama Ethernet).
2. El sistema operativo (los búferes son de 512 bytes).
3. Los protocolos (cantidad de bits en el campo de longitud de paquete).
4. El cumplimiento de algún estándar (internacional).
5. Reducir las retransmisiones inducidas por errores.
6. Evitar que un paquete ocupe el canal demasiado tiempo.

El resultado es que los diseñadores de redes no tienen la libertad de elegir cualquier tamaño máximo de paquetes. Las cargas útiles máximas son de 1500 bytes para Ethernet y 2 272 para 802.11. El protocolo IP es más generoso, permite hasta 65 515 bytes. Los hosts prefieren transmitir paquetes grandes, ya que esto reduce las sobrecargas de paquetes y el ancho de banda desperdiciado en los bytes de encabezado. Surge un problema cuando un paquete grande quiere viajar a través de una red cuyo tamaño máximo de paquete es muy pequeño. Una fuente no conoce la ruta que tomará un paquete a través de la red hacia un destino, por lo que no sabe qué tan pequeños deben ser los paquetes.

A este tamaño de paquete se le conoce como MTU de la ruta (Unidad de Transmisión Máxima de la ruta, del inglés Path Maximum Transmission Unit). Incluso si la fuente conociera el MTU de la ruta, los paquetes se enrutan de manera independiente las rutas pueden cambiar de repente, lo que puede cambiar el MTU de la ruta.

La solución alternativa es permitir que los enrutadores dividan los paquetes en fragmentos y envíen cada uno como un paquete de red separado. Las redes también tienen problemas al unir nuevamente los fragmentos. Existen dos estrategias para recombinar los fragmentos de vuelta en el paquete original. La primera es hacer que la fragmentación producida por una red sea transparente para cualquier red subsecuente por la que deba pasar. Cuando un paquete de tamaño excesivo llega a G1, el enrutador lo divide en fragmentos. Cada fragmento es dirigido al mismo enrutador de salida, G2, en donde se recombinan las piezas. Se ha hecho transparente el paso a través de la red de paquete pequeño. Las redes subsecuentes ni siquiera se enteran de que ha ocurrido una fragmentación. Es sencilla, pero tiene algunos problemas. El enrutador de salida debe saber cuándo ha recibido todas las piezas, por lo que debe incluirse un campo de conteo de "fin de paquete". Como todos los paquetes deben salir por el mismo enrutador para volver a ensamblarlos, las rutas están restringidas. Al no permitir que algunos fragmentos sigan una ruta distinta, puede bajar un poco el desempeño. Tal vez necesite colocar los fragmentos en un búfer a medida que vayan llegando, para después decidir descartarlos si no llegan todos los fragmentos.

La otra estrategia de fragmentación es abstenerse de recombinar los fragmentos en los enrutadores intermedios. Una vez que se ha fragmentado un paquete, cada fragmento se trata como si fuera un paquete original, la recombinación ocurre sólo en el host de destino. La principal ventaja de la fragmentación no transparente es que los enrutadores tienen que trabajar menos, requiere que los fragmentos se enumeren de tal forma que se pueda reconstruir el flujo de datos original. El diseño utilizado por IP es proporcionar a cada fragmento un número de paquete, un desplazamiento de bytes absoluto dentro del paquete, y una bandera que indica si es el final del paquete. Los fragmentos se pueden colocar en un búfer en el destino, en el lugar apropiado para recombinarlos, aun cuando lleguen desordenados, se pueden fragmentar si pasan a través de una red con una MTU aún más pequeña. Las retransmisiones del paquete se pueden fragmentar en distintas piezas, los fragmentos pueden ser de un tamaño arbitrario. El destino simplemente usa el número de paquete y el desplazamiento del fragmento para colocar los datos en la posición correcta, y la bandera de fin de paquete para determinar cuándo tiene el paquete completo. Este diseño tiene problemas. La sobrecarga puede ser mayor que con la fragmentación transparente, ya que los encabezados de los fragmentos ahora se transportan a través de algunos enlaces en donde tal vez no sean necesarios. La fragmentación es perjudicial para el desempeño, ya que un paquete completo se extravía si cualquiera de sus fragmentos se pierde, y porque la fragmentación representa una carga para los hosts.

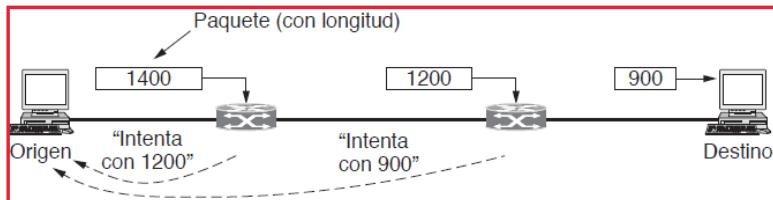


La fragmentación cuando el tamaño de datos elemental es de 1 byte. (a) El paquete original contiene 10 bytes de datos. (b) Los fragmentos después de pasar por una red con un tamaño máximo de paquete de 8 bytes de carga útil más encabezado. (c) Fragmentos después de pasar a través de una puerta de enlace de tamaño 5.

Esto nos conduce de vuelta a la solución original de deshacernos de la fragmentación en la red, la estrategia que se utiliza en la Internet moderna. Se le conoce como descubrimiento de MTU de la ruta. Cada paquete IP se envía con sus bits de encabezado establecidos para indicar que no se puede realizar ningún tipo de fragmentación. Si un enrutador recibe un paquete demasiado grande, genera un paquete de error, lo devuelve a la fuente y descarta el paquete. Cuando el origen recibe el paquete de error, usa la información

interna para volver a fragmentar el paquete en piezas que sean lo bastante pequeñas como para que el enrutador las pueda manejar. Si un enrutador más adelante en la ruta tiene una MTU aún más pequeña, se repite el proceso.

La ventaja del descubrimiento de MTU de la ruta es que ahora la fuente sabe la longitud del paquete que puede enviar; la fragmentación se sacó de la red y se pasó a los hosts. La desventaja del descubrimiento de MTU de la ruta es que puede haber retrasos iniciales adicionales sólo por enviar un paquete.



LA CAPA DE RED DE INTERNET

Los principios que guiaron su diseño en el pasado:

1. Asegurarse de que funciona. No termine el diseño hasta que múltiples prototipos se hayan comunicado entre sí de manera exitosa.
2. Mantener la simplicidad. Si una característica no es absolutamente esencial, descártela.
3. Elegir opciones claras. Si hay varias maneras para realizar la misma tarea, elija sólo una.
4. Explotar el modularidad. Lleva directamente a la idea de tener pilas de protocolos, cuyas capas sean independientes entre sí. Si las circunstancias requieren que una capa cambie, los otros no se verán afectados.
5. Prevenir la heterogeneidad. En cualquier red habrá diferentes tipos de hardware, el diseño de la red debe ser simple, general y flexible.
6. Evitar las opciones y parámetros estáticos. Es mejor hacer que el emisor y el receptor negocien.
7. Buscar un buen diseño, no es necesario que sea perfecto.
8. Ser estricto cuando envíe y tolerante cuando reciba. Sólo envíe paquetes que cumplan rigurosamente con los estándares, pero espere paquetes que tal vez no cumplan del todo y trate de lidiar con ellos.
9. Pensar en la escalabilidad. Debe manejar de manera efectiva millones de hosts, no se toleran bases de datos centralizadas y la carga se debe dispersar de la manera más equitativa entre los recursos disponibles.
10. Considerar el desempeño y el costo.

Internet puede verse como un conjunto de redes, o Sistemas Autónomos (AS) interconectados. Existen varias redes troncales (backbones), se construyen líneas de alto ancho de banda y enrutadores rápidos. Las más grandes de estas redes troncales, se llaman redes de Nivel 1. Conectadas a las redes troncales hay ISP que proporcionan acceso a hogares, centros de datos, servidores y redes regionales. Los centros de datos sirven gran parte del contenido que se envía a través de Internet. A las redes regionales están conectados más ISP.

El que mantiene unida a Internet es el protocolo de capa de red, IP (Protocolo de Internet, del inglés Internet Protocol). IP se diseñó desde el principio con la interconexión de redes en mente. Su trabajo es proporcionar un medio (sin garantía) para transportar paquetes de la fuente al destino, sin importar si están en la misma red o si hay otras redes entre ellas. La capa de transporte toma flujos de datos y los divide para poder enviarlos como paquetes IP. Los paquetes pueden ser de hasta 64 Kbytes cada uno, pero en la práctica por lo general no sobrepasan los 1500 bytes (ya que son colocados en una trama de Ethernet). Los enrutadores IP reenvían cada paquete a través de Internet, a lo largo de una ruta de un enrutador a otro, hasta llegar al destino. La capa de red las vuelve a ensamblar para formar el datagrama original y este se entrega a la capa de transporte. Hay muchas rutas posibles entre dos hosts. Los protocolos de enrutamiento IP tienen la tarea de decidir qué rutas usar.

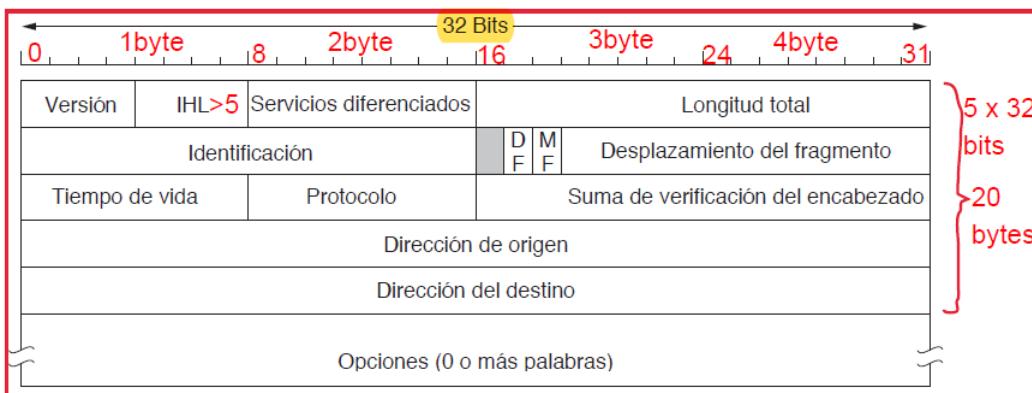
El protocolo IP versión 4

Un datagrama IPv4 consiste en dos partes: el encabezado y carga útil. El encabezado tiene una parte fija de 20 bytes y una parte opcional de longitud variable. Los bits se transmiten en orden de izquierda a derecha y de arriba hacia abajo, comenzando por el bit de mayor orden del campo Versión (este es un orden big

endian. En las máquinas little endian, como Intel x86, se requiere una conversión por software tanto para la transmisión como para la recepción). El formato little endian hubiera sido mejor, pero al momento de diseñar IP nadie sabía que llegaría a dominar la computación.

Versión lleva el registro de la versión del protocolo al que pertenece el datagrama. La versión 4 es la que domina Internet en la actualidad, es posible tener una transición entre versiones, IPv6 se está empezando a implementar. IPv5 fue un protocolo de flujo experimental en tiempo real que nunca fue muy popular.

Dado que la longitud del encabezado no es constante, se incluye un campo en el encabezado (IHL) para indicar su longitud en palabras de 32 bits. El valor mínimo es de 5, el valor máximo 15 (es de 4 bits el campo), lo que limita el encabezado a 60 bytes y, por lo tanto, el campo Opciones a 40 bytes. Para algunas opciones, por ejemplo, una que registre la ruta que ha seguido un paquete.



Servicio diferenciado. En un principio el nombre de este campo era Tipo de servicio. Su propósito es distinguir entre las diferentes clases de servicios, como confiabilidad y velocidad. Para voz digitalizada, la entrega rápida le gana a la entrega precisa. Para la transferencia de archivos, es más importante la transmisión libre de errores. Contaba con 3 bits para indicar la prioridad y 3 bits para indicar si a un host le preocupaba más el retardo, la velocidad real o la confiabilidad. Ahora, los 6 bits superiores se utilizan para marcar el paquete con su clase de servicio; expedited y asegurado. Los 2 bits inferiores se utilizan para transportar información sobre la congestión.

Longitud total incluye tanto el encabezado como los datos. La longitud máxima es de 65 535 bytes.

Identificación para que el host de destino determine a qué paquete pertenece un fragmento recién llegado. Todos los fragmentos contienen el mismo valor de Identificación.

A continuación, viene un bit sin uso, el “bit malo”.

Después vienen dos campos de 1 bit relacionados con la fragmentación. DF significa no fragmentar (Don't Fragment), es una orden para que los enruteadores, estaban destinados para soportar a los hosts incapaces de reunir las piezas otra vez. Ahora se utiliza como parte del proceso para descubrir la MTU de la ruta: el paquete más grande que puede viajar sin necesidad de fragmentarse.

MF significa más fragmentos (More Fragments). Todos los fragmentos excepto el último tienen establecido este bit.

Desplazamiento del fragmento indica a qué parte del paquete actual pertenece este fragmento. Todos los fragmentos excepto el último del datagrama deben ser un múltiplo de 8 bytes, puede haber un máximo de 8 192 fragmentos por datagrama, los campos Identificación, MF y Desplazamiento del fragmento se utilizan para implementar la fragmentación

TtL (Tiempo de vida) es un contador que se utiliza para limitar el tiempo de vida de un paquete. Se suponía que iba a contar el tiempo en segundos, en la práctica, simplemente cuenta los saltos. Cuando el contador llega a cero, el paquete se descarta y se envía de regreso un paquete de aviso al host de origen.

Una vez que la capa de red ha ensamblado un paquete, el campo Protocolo le indica a cuál proceso de transporte debe entregar el paquete, como TCP UDP y otros más. Se listan en www.iana.org.

El encabezado transporta información vital, estima su propia suma de verificación por protección, la Suma de verificación del encabezado, suma todas las medias palabras de 16 bits del encabezado a medida que vayan llegando, mediante complemento a uno, y después obtiene el complemento a uno del resultado. Es útil para detectar errores se debe recalcular en cada salto, ya que por lo menos hay un campo que siempre cambia (el campo Tiempo de vida)

Dirección de origen y Dirección de destino indican la dirección IP de las interfaces de red de la fuente y del destino.

Opciones se diseñó para proporcionar un recurso que permitiera que las versiones subsiguientes del protocolo incluyeran información que no estuviera presente en el diseño original, son de longitud variable. Cada una empieza con un código de 1 byte que identifica la opción. Algunas van seguidas de un campo de longitud de la opción de 1 byte. El campo Opciones se rellena para completar múltiplos de 4 bytes.

Opción	Descripción
Seguridad.	Especifica qué tan secreta es la información, actualmente todos los enrutadores lo ignoran.
Enrutamiento estricto desde el origen.	Proporciona la ruta completa a seguir.
Enrutamiento libre desde el origen.	Proporciona una lista de enrutadores que no se deben omitir.
Registrar ruta.	Hace que cada enrutador adjunte su dirección IP.
Estampa de tiempo.	Hace que cada enrutador adjunte su dirección y su etiqueta de tiempo.

Seguridad indica qué tan secreta es la información, actualmente todos los enrutadores lo ignoran.

Enrutamiento estricto desde el origen proporciona la ruta completa desde el origen hasta el destino como una secuencia de direcciones IP. Se requiere que el datagrama siga esa ruta exacta, se usa cuando necesitan enviar paquetes de emergencia cuando se corrompen las tablas de enrutamiento, o para hacer mediciones de sincronización.

Enrutamiento libre desde el origen requiere que se le permite pasar a través de otros enrutadores en el camino. Por lo general esta opción sólo indicará unos cuantos enrutadores para forzar una ruta específica. Es de mucha utilidad para evitar ciertos países.

Registrar ruta indica a cada enrutador a lo largo de la ruta que adjunte su dirección IP al campo Opciones. Esto permite a los administradores del sistema buscar fallas. En un principio 40 bytes de opciones eran más que suficientes, ahora son muy pocos.

Estampa de tiempo es como la opción Registrar ruta, excepto que además cada enrutador también registra una estampa de tiempo de 32 bits, es útil para la medición en las redes.

Hoy en día se han dejado de usar las opciones IP, se ignoran directamente.

Direcciones IP

IPv4 consiste en direcciones de 32 bits. Cada host y enrutador tiene una dirección IP que se puede usar en los campos Dirección de origen y Dirección de destino de los paquetes. Una dirección IP en realidad no se refiere a un host, sino a una interfaz de red, si un host está en dos redes, debe tener dos direcciones IP, la mayoría de los hosts están en una red y tienen una dirección IP. Los enrutadores tienen varias interfaces y múltiples direcciones IP.

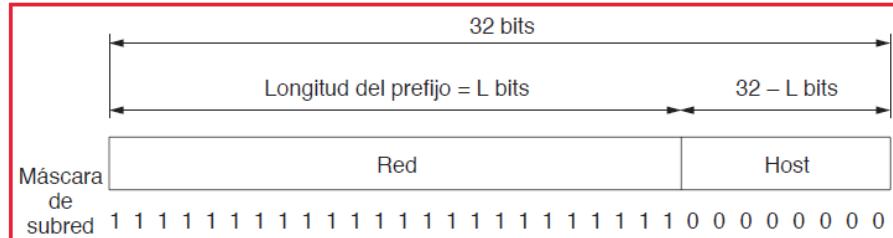
Prefijos

Las direcciones IP son jerárquicas. Cada dirección está compuesta de una porción de red de longitud variable en los bits superiores, y de una porción de host en los bits inferiores. La porción de red tiene el mismo valor para todos los hosts en una sola red, una red corresponde a un bloque contiguo de espacio de direcciones IP. A este bloque se le llama prefijo.

Las direcciones IP se escriben en notación decimal con puntos, de 0 a 255. Para escribir los prefijos, se proporciona la dirección IP menor en el bloque y el tamaño de este se determina mediante el número de bits

en la porción de red, debe ser una potencia de dos. Por convención, se escribe después de la dirección IP como una barra diagonal seguida de la longitud en bits de la porción de red. Si el prefijo contiene 28 direcciones y, por lo tanto, deja 24 bits para la porción de red, se escribe /24. La longitud del prefijo no se puede inferir, los protocolos de enrutamiento deben transportar los prefijos hasta los enrutadores. La longitud del prefijo corresponde a una máscara binaria de 1s en la porción de red, se denomina máscara de subred. Se puede aplicar un AND a la máscara con la dirección IP para extraer la porción de la red.

Direcciones jerárquicas: Las ventajas clave de los prefijos es que los enrutadores pueden reenviar paquetes con base en la porción de red de la dirección, la porción del host no importa a los enrutadores, ya que enviarán en la misma

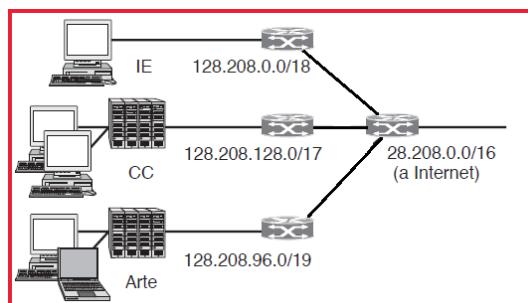


dirección a todos los hosts de la misma red, los paquetes llegan a la red de destino se reenvían al host correcto y hace que las tablas de enrutamiento sean mucho más pequeñas. La desventaja es que la dirección IP de un host depende de su ubicación en la red. Una dirección Ethernet se puede usar en cualquier parte del mundo, pero cada dirección IP pertenece a una red específica, por lo que los enrutadores sólo podrán entregar paquetes destinados a esa dirección en la red. Se necesitan diseños como IP móvil para soportar hosts que se desplazan de una red a otra, pero que deseen mantener las mismas direcciones IP. La jerarquía desperdicia direcciones, se asignan direcciones a las redes en bloques grandes, habrá direcciones que se asignen, pero no se utilicen. No sería de gran importancia si hubiera muchas direcciones, pero el crecimiento de Internet está agotando el espacio de direcciones libres. IPv6 es la solución.

Subredes

Los números de red se administran a través de ICANN (Corporación de Internet para la Asignación de Nombres y Números, del inglés Internet Corporation for Assigned Names and Numbers), la asignación de direcciones IP es continua a medida que crecen las compañías. El enrutamiento por prefijo requiere que todos los hosts en una red tengan el mismo número de red, puede provocar problemas a medida que las redes aumentan su tamaño. Con prefijo /16, para el Departamento de Ciencias Computacionales. Un año después, Ingeniería Eléctrica desea entrar a Internet, le sigue Arte; ¿Qué direcciones IP deben usar? El prefijo /16 que ya está asignado tiene suficientes direcciones para más de 60 000 hosts. Se requiere una organización diferente.

La solución es dividir el bloque de direcciones en varias partes para uso interno en forma de múltiples redes, pero actuar como una sola red para el mundo exterior, se les llama subredes. El prefijo /16 se dividió en varias piezas, no necesita ser uniforme, hay que alinear cada pieza de manera que se pueda usar cualquier bit en la porción inferior del host.



Ciencias Computacionales:	10000000	11010000	1lxxxxxx	xxxxxxxx	/17 - CC
Ingeniería Eléctrica:	10000000	11010000	00lxxxxx	xxxxxxxx	/18 - IE
Arte:	10000000	11010000	011 xxxx	xxxxxxxx	/19-Arte
Libre:	10000000	11010000	010 xxxx	xxxxxxxx	/19-Libre

La mitad del bloque (un /17) se asigna a Ciencias Computacionales, una cuarta parte se asigna Ingeniería Eléctrica (un /18) y una octava parte (un /19) para Arte. El octavo restante queda sin asignar. () muestra el límite entre el número de la subred y la porción del host.

Cuando llega un paquete al enrutador principal, entran los detalles de nuestros prefijos. Una forma sería que cada enrutador tuviera una tabla con 65.535 entradas para cada host socavaría el principal beneficio de usar una jerarquía. En cambio, los enrutadores simplemente tienen que conocer las máscaras de subred, llega un paquete, el enrutador analiza su dirección de destino y verifica a qué subred pertenece, el enrutador aplica un AND a la dirección del destino con la máscara para cada subred.

Un paquete destinado para la dirección IP 128.208.2.151. Para ver si está dirigido al CC, le aplicamos un AND con 255.255.128.0 para tomar los primeros 17 bits (que son 128.208.0.0) y ver si coinciden con 128.208.128.0, no coinciden. Si verificamos los primeros 18 bits para IE, obtenemos 128.208.0.0. Este resultado coincide con la dirección del prefijo, por lo que el paquete se reenvía por la interfaz que conduce a la red. Las divisiones se pueden cambiar si es necesario, se actualizan todas las máscaras de subred en los enrutadores. Fuera de la red, las subredes no son visibles.

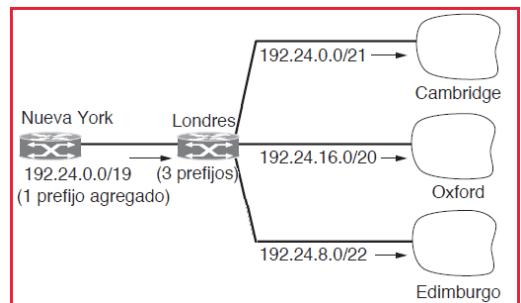
CIDR: Enrutamiento Interdominio sin Clases

Incluso si se asignan bloques de direcciones IP de manera eficiente, hay un problema presente: la explosión de las tablas de enrutamiento. Los enrutadores en las organizaciones, tienen que tener una entrada para cada una de sus subredes. Para las rutas a destinos fuera de la organización, sólo necesitan usar la línea hacia el ISP. Los enrutadores en los ISP y las redes troncales deben saber qué ruta tomar hacia cada una de las redes; no funciona la regla predeterminada. Estos enrutadores básicos están en la zona libre predeterminada. Esto puede generar una tabla muy grande. Los algoritmos de enrutamiento requieren que cada enrutador intercambie información sobre las direcciones con otros enrutadores. El problema de las tablas de enrutamiento se podría haber resuelto mediante el uso de una jerarquía más profunda, hacer que cada dirección IP contenga campos de país, estado/provincia, ciudad, red y host. Esta solución requeriría mucho más de 32 bits y utilizaría las direcciones de una manera ineficiente.

Podemos aplicar la misma perspectiva que en las subredes: los enrutadores en distintas ubicaciones pueden saber acerca de una dirección IP dada que pertenece a prefijos de distintas zonas. Sin embargo, en vez de dividir un bloque de direcciones en subredes, aquí combinamos varios prefijos pequeños en un solo prefijo más grande. Se le conoce como agregación de rutas, al prefijo más grande resultante se le denomina superred.

Las direcciones IP están contenidas en prefijos de diversos tamaños. Prefijo /22 contiene 2^{10} direcciones, puede ser tratada por otro enrutador como parte de un prefijo /20 más grande (que contiene 2 direcciones). Es responsabilidad de cada enrutador tener la información del prefijo correspondiente. Se conoce como CIDR (Enrutamiento Interdominio sin Clases, del inglés Classless InterDomain Routing), se especifica en el RFC 4632. Vamos a considerar un ejemplo, hay un bloque de 8 192 direcciones IP (/19) disponible, empezando en 194.24.0.0.

Universidad	Primera dirección	Última dirección	Cuántas	Prefijo
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edimburgo	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Disponible)	194.24.12.0	194.24.15.255	1024	194.24.12.0/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20



Los enrutadores en la zona libre predeterminada se les dice ahora sobre las direcciones IP en las tres redes. Los enrutadores cercanos necesiten enviar por una línea de salida distinta para cada uno de los prefijos, necesitan una entrada para cada uno en sus tablas de enrutamiento. Veamos tres universidades desde el punto de vista de un enrutador distante en Nueva York. Las direcciones IP en los tres prefijos se deben enviar de Nueva York a Londres. El proceso de enrutamiento en Londres detecta esto y combina los tres prefijos en una sola entrada agregada para el prefijo 194.24.0.0/19, que pasa al enrutador de Nueva York. Este prefijo contiene 8K direcciones y cubre las tres universidades, junto con las otras 1024 direcciones no asignadas, estos prefijos se redujeron a uno. Es posible traslapar a los prefijos. La regla es que los paquetes se envíen, al prefijo más largo coincidente que tenga la menor cantidad de direcciones IP. Ofrece un grado conveniente de flexibilidad. Se ha asignado ahora a una red en San Francisco, el enrutador de Nueva York mantiene cuatro prefijos, tres de ellos a Londres y cuarto prefijo a San Francisco. Las direcciones IP dentro de la red de San Francisco se enviarán en la línea de salida a San Francisco, y todas las demás direcciones IP en el prefijo más grande se enviarán a Londres. Cuando llega un paquete, se explora la tabla de enrutamiento para determinar si el destino está dentro del prefijo. Es posible que coincidan varias entradas con distintas longitudes de prefijos, en cuyo caso se utiliza la entrada con el prefijo más largo. Por ende, si

entretenimiento dificultó repartir las IPv4. IETF empezó a trabajar sobre una nueva versión de IP, una que nunca se quedará sin direcciones y fuera más flexible y eficiente. Sus objetivos eran:

1. Soportar miles de millones de hosts, incluso con una asignación de direcciones ineficiente.
2. Reducir el tamaño de las tablas de enrutamiento.
3. Simplificar el protocolo para permitir a los enrutadores procesar los paquetes con más rapidez.
4. Proporcionar mayor seguridad (autentificación y privacidad).
5. Poner más atención en cuanto al tipo de servicio, en especial para los datos en tiempo real.
6. Ayudar a la multidifusión al permitir la especificación de alcances.
7. Permitir que un host deambule libremente sin tener que cambiar su dirección.
8. Permitir que el protocolo evolucione en el futuro.
9. Permitir que el protocolo viejo y el nuevo coexistan durante años.

La IETF emitió una llamada para propuestas y discusión en el RFC 1550. Una fue la de operar TCP sobre CLNP, el protocolo de capa de red diseñado para OSI. Con sus direcciones de 160 bits. Esta habría unificado dos de los principales protocolos de capa de red. Muchas personas sentían que esto hubiera sido como admitir que algo en el mundo de OSI se había hecho bien, una declaración considerada como políticamente incorrecta. CLNP estaba modelado parecido a IP, un golpe contra CLNP era su defectuoso soporte para los tipos de servicios. Tres de las mejores propuestas se publicaron en IEEE Network, después de mucha discusión, se seleccionó una versión combinada modificada SIPP (Protocolo Simple de Internet Mejorado, del inglés Simple Internet Protocol Plus), designado IPv6, cumple bastante bien con los objetivos de la IETF. Pero no es compatible con IPv4, pero es compatible con los otros protocolos auxiliares de Internet, incluyendo TCP, UDP, ICMP, IGMP, OSPF, BGP y DNS. Cinco características claves:

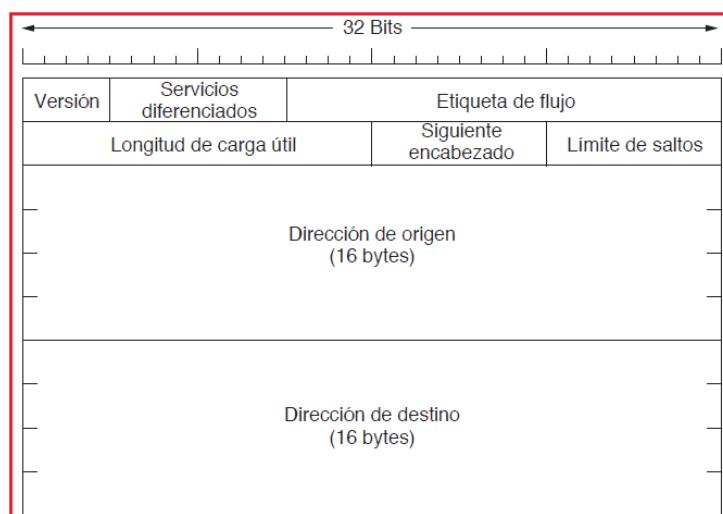
1. IPv6 tiene direcciones de 128 bits.
2. La simplificación del encabezado. Contiene siete campos en comparación con los 13 de IPv4, mejora la velocidad de transmisión real y el retardo.
3. Es un soporte mejorado para las opciones, los campos que antes eran requeridos ahora son opcionales, es más fácil para los enrutadores omitir las porciones que no están destinadas para ellos, agiliza el tiempo de procesamiento de los paquetes.
4. Representa un gran avance en la seguridad. La autentificación y la privacidad son características clave.
5. Se puso más énfasis en la calidad del servicio.

El encabezado principal de IPv6

El campo Versión siempre es 6 para IPv6 (y 4 para IPv4). Los enrutadores podrán examinar este campo para saber qué tipo de paquete tienen. Algunos enrutadores omiten la verificación.

Servicios diferenciados (originalmente conocido como Clase de tráfico) se utiliza para distinguir la clase de servicio con distintos requerimientos de entrega en tiempo real. Los 2 bits de menor orden se usan para señalar las indicaciones explícitas de congestión, en la misma forma que IPv4.

Etiqueta de flujo para que un origen y un destino marquen grupos de paquetes que tengan los mismos requerimientos y que la red deba tratar de la misma forma, para formar una pseudoconexión. Se puede establecer por adelantado, dándole un identificador. Con una Etiqueta de flujo diferente de cero, los enrutadores pueden buscarla en sus tablas internas para ver el tipo de tratamiento requiere. Cada flujo está designado con base en la dirección de origen, la dirección de destino y el número de flujo, pueden estar activos hasta 2^{20} flujos al mismo tiempo entre un par dado de direcciones IP. Lo ideal es que las etiquetas de flujo se escojan al azar, de modo que los enrutadores tengan que usar hashing.



Longitud de carga útil indica cuántos bytes van después del encabezado de 40 bytes, se cambió de Longitud total en el IPv4 porque el significado cambió, ya que los 40 bytes del encabezado ya no se cuentan. Significa que ahora la carga útil puede ser de 65 535 bytes en vez de sólo 65 515 bytes.

Siguiente encabezado. La razón por la que pudo simplificarse el encabezado es que puede haber encabezados adicionales (opcionales). Este campo indica cuál de los seis encabezados de extensión (en la actualidad), siguen de este. Si este encabezado es el último encabezado de IP, el campo Siguiente encabezado indica el protocolo de transporte al que se entregará el paquete.

Límite de saltos para evitar que los paquetes vivan eternamente. Igual al campo Tiempo de vida del IPv4; es un campo que se disminuye en cada salto.

Dirección de origen y Dirección de destino. Se decidió que la mejor medida sería una dirección de 16 bytes de longitud fija. Se ha desarrollado una nueva notación para escribir estas direcciones. Se escriben como ocho grupos de cuatro dígitos hexadecimales, separando los grupos por dos puntos. Muchas direcciones tendrán muchos ceros en ellas, se han autorizado tres optimizaciones, se pueden omitir los ceros a la izquierda dentro de un grupo, se pueden reemplazar uno o más grupos de 16 bits cero por un par de signos de dos puntos (8000::123:4567:89AB:CDEF) y las direcciones IPv4 se pueden escribir como un par de signos de dos puntos y un número decimal anterior separado por puntos (::192.31.20.46). Igualmente, el espacio de direcciones no se usará de manera eficiente. Podemos comparar el encabezado de IPv4 con el de IPv6 para ver lo que se ha dejado fuera. IHL se fue porque el encabezado de IPv6 tiene una longitud fija. Protocolo se retiró porque el campo Siguiente encabezado indica lo que sigue después del último encabezado IP. Se eliminaron todos los campos relacionados con la fragmentación, puesto que todos los hosts que cumplen con el IPv6 deben determinar en forma dinámica el tamaño de paquete a usar. Utilizan el procedimiento de descubrimiento de MTU de la ruta. Cuando un host envía un paquete IPv6 demasiado grande, en vez de fragmentarlo, el enrutador que no puede reenviarlo descarta el paquete y envía un mensaje de error de vuelta al host emisor. Este mensaje indica al host que dividirá todos los paquetes futuros, el tamaño mínimo de paquete se incrementó de 576 a 1280 bytes para permitir 1024 bytes de datos y muchos encabezados.

Suma de verificación desapareció porque al calcularlo se reduce en gran medida el desempeño. La capa de enlace de datos y las capas de transporte por lo general tienen sus propias sumas de verificación, la ventaja de tener otra suma de verificación no valía el costo, el resultado es un protocolo de capa de red compacto y sólido.

Encabezados de extensión

Ocasionalmente se requieren algunos de los campos faltantes del IPv4, el IPv6 introdujo el concepto de encabezados de extensión, se pueden usar para proporcionar información adicional, pero codificada de una manera eficiente. Hay seis tipos, todos son opcionales, si hay más de uno, deben aparecer justo después del encabezado fijo y de preferencia en el orden listado. Algunos de los encabezados tienen un formato fijo; otros variables, cada elemento está codificado (Tipo, Longitud, Valor).

Encabezado de extensión	Descripción
Opciones salto por salto.	Información diversa para los enrutadores.
Opciones de destino.	Información adicional para el destino.
Enrutamiento.	Lista informal de los enrutadores a visitar.
Fragmentación.	Manejo de fragmentos de datagramas.
Autenticación.	Verificación de la identidad del emisor.
Carga útil de seguridad cifrada.	Información sobre el contenido cifrado.

Siguiente encabezado	0	194	4
Longitud de carga útil de tamaño extra			

Encabezado de extensión para jumbogramas

Tipo es un campo de 1 byte que indica la opción de la que se trata, se han escogido de modo que los 2 primeros bits indiquen a los enrutadores, que no saben cómo procesar la opción, lo que tienen que hacer. Las posibilidades son: omitir la opción, descartar el paquete, descartar y enviar de vuelta un paquete ICMP, y descartar, pero no enviar paquetes ICMP para direcciones de multidifusión.

La Longitud es de 1 byte, e indica la longitud del valor (0 a 255 bytes). El Valor es cualquier información requerida de hasta 255 bytes. El encabezado de salto por salto se usa para la información que deben examinar todos los enrutadores a lo largo de la ruta. Hasta ahora, se ha definido una opción: manejo de datagramas de más de 64K. Cuando se utiliza, el campo Longitud de carga útil del encabezado fijo se establece a 0.

Encabezado de extensión, éste comienza con un byte que indica el tipo de encabezado que sigue. A este byte le sigue uno que indica la longitud del encabezado salto por salto en bytes, excluyendo los primeros 8 bytes, que son obligatorios. Todas las extensiones empiezan de esta manera.

Los siguientes 2 bytes define el tamaño del datagrama (código 194) y que el tamaño es un número de 4 bytes. Los últimos 4 bytes indican el tamaño del datagrama. No se permiten tamaños menores a 65 536 bytes, pues de lo contrario el primer enrutador descartará el paquete y devolverá ICMP de error. Los datagramas que usan este encabezado se llaman jumbogramas. Es importante para las aplicaciones de supercomputadoras que deben transferir gigabytes de datos. El encabezado de opciones de destino está reservado para campos que sólo se necesitan interpretar en el host de destino. En un principio no se usará. Se incluyó para asegurarse que el nuevo enrutamiento y el software del host pudieran manejarlo.

Siguiente encabezado	Longitud de extensión de encabezado	Tipo de enrutamiento	Segmentos restantes
Datos de tipo específico			

El encabezado de enrutamiento lista uno o más enrutadores que deben visitarse en el camino hacia el destino, similar al enrutamiento de origen libre del IPv4. Los primeros 4 bytes contienen cuatro enteros de 1 byte. Anteriormente describimos los campos Siguiente encabezado y Longitud de extensión del encabezado. El campo Tipo de enrutamiento proporciona el formato del resto del encabezado. El tipo 0 indica que una palabra reservada de 32 bits va después de la primera palabra, seguida por cierto número de direcciones de IPv6. Pueden inventarse otros tipos. Segmentos restantes registra cuántas direcciones de la lista no se han visitado. Se decrementa cada vez que se visita una. Cuando llega a 0, el paquete está por su cuenta, sin más guía

El encabezado de fragmentación parecida a la del IPv4. El encabezado contiene el identificador del datagrama, el número de fragmento y un bit que indica si seguirán más, a diferencia del IPv4, en el IPv6 sólo el host de origen puede fragmentar un paquete. Los enrutadores no pueden hacerlo. Simplifica el trabajo del enrutador y acelera el proceso de enrutamiento. Si un enrutador confronta un paquete demasiado grande, lo descarta y devuelve un paquete de error ICMP al origen, permite que el host de origen fragmente el paquete en piezas más pequeñas mediante el uso de este encabezado y lo intente de nuevo. El encabezado de autenticación proporciona un mecanismo mediante el cual el receptor de un paquete puede estar seguro de quién lo envió. La carga útil de seguridad de cifrado hace posible cifrar el contenido de un paquete.

Protocolos de control en Internet

Internet tiene varios protocolos de control complementarios como ICMP, ARP y DHCP. ICMP y DHCP tienen versiones similares para IPv6; el equivalente de ARP se llama NDP (Protocolo de Descubrimiento de Vecino, del inglés Neighbor Discovery Protocol) para IPv6.

ICMP: Protocolo de Mensajes de Control en Internet

Cuando ocurre algo inesperado durante el procesamiento de un paquete en un enrutador, ICMP (Protocolo de Mensajes de Control en Internet, del inglés Internet Control Message Protocol) informa sobre el evento al emisor, también se utiliza para probar Internet. Hay definidos alrededor de una docena de tipos de mensajes ICMP. La lista en línea se conserva ahora en www.iana.org/assignments/icmp-parameters.

ARP: Protocolo de Resolución de Direcciones

Las NIC (Tarjetas de Interfaz de Red) de la capa de enlace de datos no entienden las direcciones de Internet, cada NIC viene equipada con una dirección Ethernet única de 48 bits. Las NIC envían y reciben tramas

basadas en direcciones Ethernet de 48 bits. No saben nada sobre direcciones IP de 32 bits. ¿cómo se convierten las direcciones IP en direcciones de la capa de enlace de datos, como Ethernet? se muestra una universidad pequeña con dos redes /24. Una red (CC) es una Ethernet commutada con prefijo 192.32.65.0/24. La otra LAN (IE), que también es Ethernet commutada, tiene el prefijo 192.32.63.0/24. Las dos LAN están conectadas por un enrutador IP. Cada máquina en una Ethernet y cada interfaz en el enrutador tienen una dirección única de Ethernet. El emisor sabe el nombre del receptor pretendido, eagle.cc.uni.edu. El primer paso es encontrar la dirección IP para el host 2. Esta consulta es realizada por el DNS que devuelve la dirección IP del host 2 (192.32.65.5).

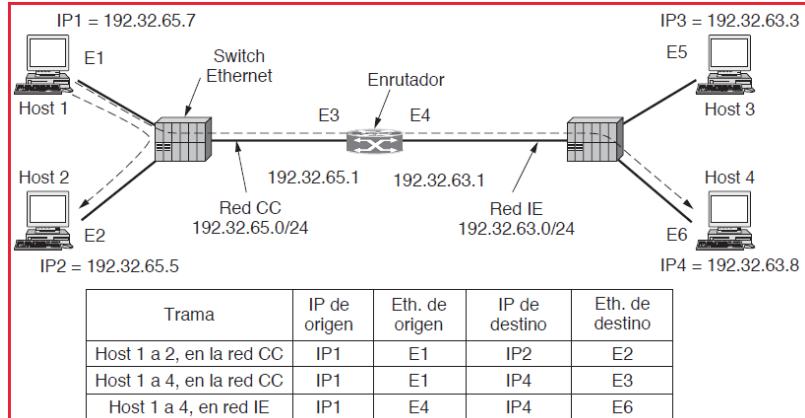
El software de la capa superior en el host 1 elabora ahora un paquete con 192.32.65.5 en el campo Dirección de destino y lo entrega al software de IP. El software IP puede buscar la dirección y ver que el destino está en la red CC. De todas formas, necesita alguna manera de encontrar la dirección Ethernet de destino para enviar la trama. Una solución es tener un archivo de configuración en alguna parte del sistema que asocie las direcciones IP con direcciones Ethernet, es una tarea propensa a errores y consume mucho tiempo.

Una mejor solución es que el host 1 envíe un paquete de difusión hacia Ethernet y pregunte quién posee la dirección IP 192.32.65.5, llegará a cada máquina en la Ethernet CC y cada una verificará su dirección IP. Al host 2 le bastará responder con su dirección de Ethernet (E2). El host 1 aprende que la dirección IP 192.32.65.5 está en el host con la dirección Ethernet E2. El protocolo utilizado se llama ARP (Protocolo de Resolución de Direcciones, del inglés Address Resolution Protocol), ARP está en el RFC 826. El administrador del sistema sólo tiene que asignar a cada máquina una dirección IP y decidir respecto a las máscaras de subred. ARP se hace cargo del resto. A estas alturas, el software IP en el host 1 crea una trama Ethernet dirigida a E2, pone el paquete IP (dirigido a 192.32.65.5) en el campo de carga útil y lo descarga hacia la Ethernet. La NIC Ethernet del host 2 detecta esta trama, la reconoce como una trama para sí mismo, la recoge y provoca una interrupción, extrae el paquete IP de la carga útil y lo pasa al software IP, ve que esté direccionado de forma correcta y lo procesa.

Hay varias optimizaciones para ARP. Una máquina ha ejecutado ARP, guarda el resultado en caché. La siguiente vez encontrará la asociación en su propio caché, con lo cual se elimina la necesidad de una segunda difusión. El host 2 necesitará devolver una respuesta y se verá forzado también a ejecutar el ARP. Podemos evitar esta difusión de ARP haciendo que el host 1 incluya su asociación IP a Ethernet en el paquete ARP. Cuando la difusión de ARP llega al host 2, se introduce el par (192.32.65.7, E1) en la caché ARP del host 2. Todas las máquinas en Ethernet pueden introducir esta asociación en su caché ARP.

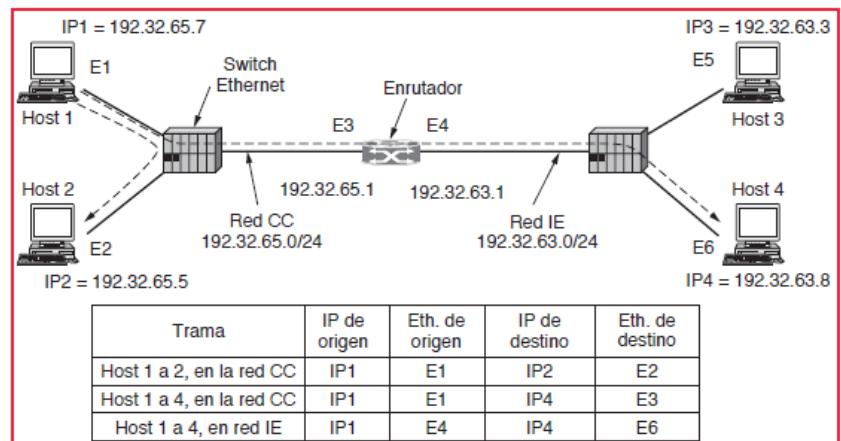
Las entradas en la caché ARP deben expirar después de unos cuantos minutos para ayudar a mantener actualizada la información en la caché y optimizar el desempeño, y que cada máquina difunda su asociación cuando se configure. Esta difusión se realiza en forma de un ARP que busca su propia dirección IP. No debe haber una respuesta (pero un efecto es actualizar una entrada en la caché ARP de todos, se le conoce como ARP gratuito). Si una respuesta llega, quiere decir que se asignó la misma dirección IP a dos máquinas. El administrador de la red debe resolver este error. Ahora veamos de nuevo el host 1 quiere enviar un paquete

Tipo de mensaje	Descripción
<i>Destination unreachable</i> (Destino inaccesible).	No se pudo entregar el paquete.
<i>Time exceeded</i> (Tiempo excedido).	El tiempo de vida llegó a cero.
<i>Parameter problem</i> (Problema de parámetros).	Campo de encabezado inválido.
<i>Source quench</i> (Fuente disminuida).	Paquete regulador.
<i>Redirect</i> (Redireccional).	Enseña a un enrutador la geografía.
<i>Echo and echo reply</i> (Eco y respuesta de eco).	Verifica si una máquina está viva.
<i>Timestamp request/reply</i> (Estampa de tiempo, Petición/respuesta).	Igual que solicitud de eco, pero con marca de tiempo.
<i>Router advertisement/solicitation</i> (Enrutamiento anuncio/solicitud).	Busca un enrutador cercano.



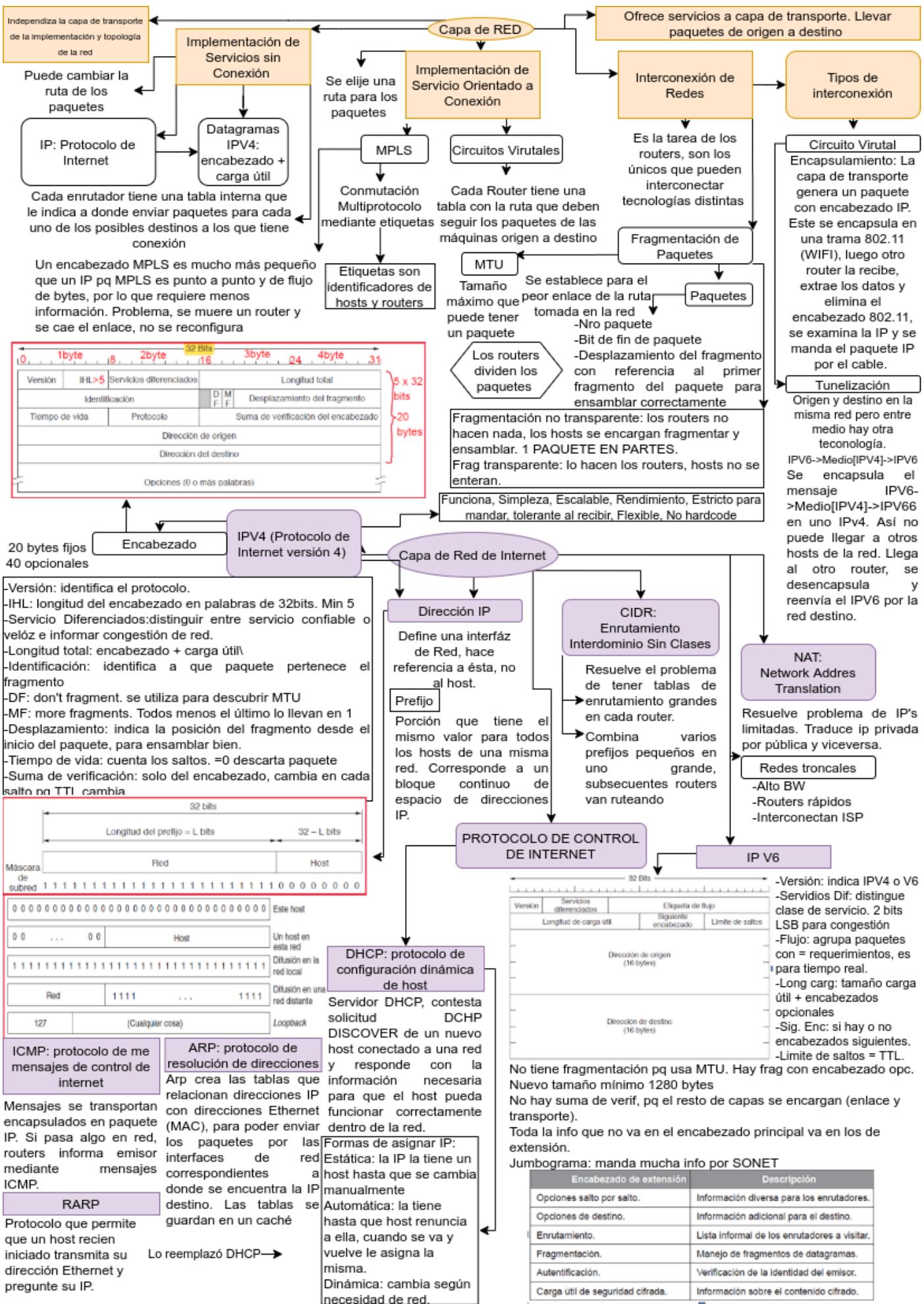
al host 4 (192.32.63.8) en la red IE. El host 1 verá que la dirección IP de destino no está en la red CC. Sabe enviar todo ese tráfico fuera de la red al enrutador, se conoce como puerta de enlace predeterminada. Por convención, es la dirección más baja en la red (198.31.65.1), el host 1 debe conocer de todas formas la dirección Ethernet de la interfaz del enrutador en la red CC. Para descubrirla envía una difusión ARP para 198.31.65.1, a partir de la cual aprende E3. Después envía la trama.

Cuando la NIC Ethernet del enrutador recibe esta trama, entrega el paquete al software IP. Sabe con base en las máscaras de red que el paquete se debe enviar a la red IE, en donde alcanzará al host 4. Si el enrutador no conoce la dirección Ethernet para el host 4, entonces usará ARP de nuevo, las direcciones Ethernet cambian con la trama en cada red, mientras que las direcciones IP permanecen constantes. Es posible enviar un paquete del host 1 al host 4 sin que el primero sepa que el segundo está en una red diferente. La solución es hacer que el enrutador responda a los ARP en la red CC para el host 4 y proporcione su dirección Ethernet, E3, como respuesta. No es posible hacer que el host 4 responda directamente, ya que no verá la solicitud ARP. El enrutador recibirá las tramas enviadas a 192.32.63.8 y las reenviará a la red IE. A esta solución se le conoce como ARP por proxy y se utiliza en casos especiales en los que un host desea aparecer en una red, aun cuando en realidad reside en otra.



DHCP: el Protocolo de Configuración Dinámica de Host

ARP asume que los hosts están configurados con cierta información básica, como sus propias direcciones IP. Esta información es posible configurar en forma manual cada computadora, pero es un proceso tedioso y propenso a errores. Una mejor manera de hacerlo, conocida como DHCP (Protocolo de Configuración Dinámica de Host, del inglés Dynamic Host Configuration Protocol). Cada red debe tener un servidor DHCP responsable de la configuración. Al iniciar una computadora, ésta tiene integrada una dirección Ethernet u otro tipo de dirección de capa de enlace de datos en la NIC, pero no cuenta con una IP. En forma muy parecida al ARP, difunde una solicitud de una dirección IP en su red, usa un paquete llamado DHCP DISCOVER. Este debe llegar al servidor DHCP. Si el servidor no está conectado directamente a la red, el enrutador las transmite al servidor DHCP en donde quiera que se encuentre. El servidor asigna una dirección IP libre y la envía al host en un paquete DHCP OFFER. Para que esto pueda funcionar incluso cuando los hosts no tienen direcciones IP, el servidor identifica a un host mediante su dirección Ethernet (la cual se transporta en el paquete DHCP DISCOVER). Un problema que surge es determinar qué tanto tiempo se debe asignar una dirección IP. Si un host sale de la red esa dirección se perderá en forma permanente. Para evitar eso, la asignación de direcciones IP puede ser por un periodo fijo de tiempo, una técnica conocida como arrendamiento, antes que expire el host debe pedir una renovación al DHCP. El DHCP se describe en los RFC 2131 y 2132. Los ISP usan DHCP para establecer los parámetros de los dispositivos a través del enlace de acceso a Internet, la información que se configura incluye la máscara de red, la dirección IP de la puerta de enlace predeterminada y las direcciones IP de los servidores DNS y de tiempo.



Capa de Transporte [esto si lo retoco porque está DEMASIADO TEXXXXTO]

La capa de transporte es el corazón de la jerarquía de protocolos. La capa de red provee una entrega de paquetes punto a punto mediante datagramas o circuitos virtuales. La capa de transporte se basa en la capa de red para proveer transporte de datos de un proceso en una máquina de origen a un proceso en una máquina de destino, con un nivel de confiabilidad independiente de las redes físicas. Ofrece las abstracciones que necesitan las aplicaciones para usar la red.

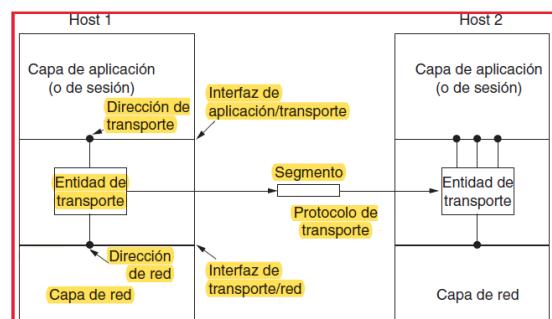
EL SERVICIO DE TRANSPORTE

Servicios que se proporcionan a las capas superiores

La meta es proporcionar un servicio de transmisión de datos eficiente, confiable y económico a sus usuarios o procesos de la capa de aplicación. Utiliza los servicios proporcionados por la capa de red. El hardware o software que se encarga del trabajo se llama entidad de transporte, puede localizarse en el kernel, un paquete de biblioteca, un proceso de usuario o en la tarjeta de interfaz de red. Las primeras dos son comunes en Internet.

Hay dos tipos de servicio de transporte:

- 1) Orientado a Conexión: las conexiones tienen tres fases: establecimiento, transferencia de datos y liberación. El direccionamiento y el control de flujo son similares en ambas capas.
- 2) Sin conexión: se usa cuando los paquetes a enviar son pocos, no vale la pena establecer una conexión.



Capas de RED y TRANSPORTE muy parecidas. Se separan porque corren en equipos distintos. Capa de red \Rightarrow Enrutadores del ISP. Puede fallar o dar un mal servicio. Se soluciona con otra capa... Capa de Transporte \Rightarrow Corre en los clientes. Puede compensar errores en la red y mejorar la calidad del servicio mediante retransmisiones. También puede reiniciar conexiones y VC si se caen durante el uso. HACE QUE EL SERVICIO SEA CONFIABLE.

Las primitivas de transporte se pueden implementar como llamadas a procedimientos de biblioteca para que sean independientes de las primitivas de red. Esta capa cumple la función clave de aislar a las capas superiores de la tecnología, el diseño y las imperfecciones de la red. Las cuatro capas inferiores se pueden ver como el proveedor del servicio de transporte, mientras que las superiores son el usuario del servicio de transporte, la capa de transporte constituye el límite principal entre el proveedor y el usuario.

Primitivas del servicio de transporte

Para que los usuarios accedan al servicio de transporte, debe proporcionar una interfaz. Cada servicio de transporte tiene su propia interfaz. El propósito del servicio de red es modelar el servicio ofrecido por las redes reales, pueden perder paquetes, el servicio no es confiable. El servicio de transporte orientado a conexión sí es confiable. Las redes reales no están libres de errores, pero el propósito es ofrecer un servicio confiable en una red no confiable. La capa de transporte puede proporcionar un servicio no confiable (datagramas), usado para tiempo real, vemos el confiable.

Servicios de transporte los usan programadores, servicio de transporte debe ser conveniente y fácil de usar. Servicio de red lo usan unos pocos, no tiene porqué ser fácil.

Las cinco primitivas, muestran la esencia de lo que debe hacer una interfaz de transporte orientada a conexión, para que los programas establezcan, usen y liberen las conexiones.

Primitiva	Paquete enviado	Significado
LISTEN	(ninguno)	Se bloquea hasta que algún proceso intenta conectarse.
CONNECT	CONNECTION REQ.	Intenta activamente establecer una conexión.
SEND	DATA	Envía información.
RECEIVE	(ninguno)	Se bloquea hasta que llegue un paquete DATA.
DISCONNECT	DISCONNECTION REQ.	Solicita que se libere la conexión

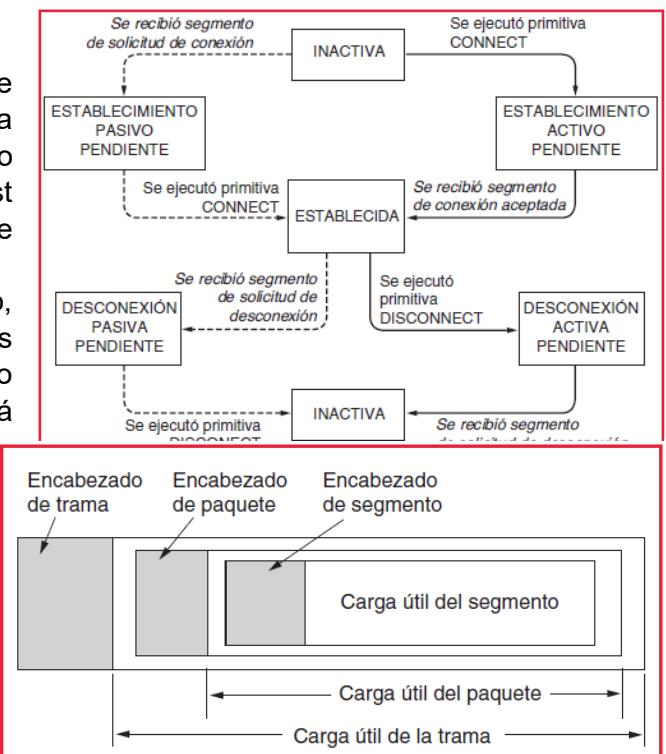
1. Servidor ejecuta LISTEN, llamada bloquea el proceso hasta que aparece un cliente.
2. Cliente desea comunicarse con el servidor, ejecuta CONNECT. La entidad de transporte ejecuta esta primitiva, bloquea al invocador y envía un paquete al servidor. En la carga útil de este paquete se encuentra un mensaje de capa de transporte encapsulado, dirigido a la entidad de transporte del servidor. Usaremos “segmento” para indicar los mensajes TCP, UDP y otros protocolos; es el TPDU (Unidad de Datos del Protocolo de Transporte, del inglés Transport Protocol Data Unit). Los segmentos intercambiados están contenidos en paquetes intercambiados por la capa de enlace de datos. Cuando llega una trama, la capa de enlace de datos procesa el encabezado si la dirección de destino coincide, pasa el contenido del campo de carga útil de la trama a la entidad de red. Esta procesa el encabezado del paquete y pasa la carga útil del paquete a la entidad de transporte.
3. CONNECT del cliente ocasiona el envío de un segmento CONNECTION REQUEST al servidor. Al llegar, la entidad de transporte verifica que el servidor esté bloqueado en LISTEN, desbloquea el servidor y envía
4. CONNECTION ACCEPTED. Al llegar, el cliente se desbloquea y se establece la conexión. Ahora pueden intercambiarse datos mediante
5. SEND y RECEIVE. Cualquiera puede emitir un RECEIVE (bloqueo) para esperar que la otra parte emita un SEND. Al llegar el segmento, el receptor se desbloquea, puede procesar el segmento y enviar una respuesta. Mientras puedan llevar el control de quién tiene el turno funciona bien.

Se confirma la recepción de cada paquete de datos enviado. La recepción de los paquetes que llevan segmentos de control que se confirmarán de manera implícita o explícita. Se manejan mediante las entidades de transporte usando el protocolo de capa de red, no son visibles para los usuarios. Las entidades de transporte tienen que preocuparse por los temporizadores y las retransmisiones. Los usuarios de transporte no se enteran de ningún aspecto. Para ellos es un conducto de bits confiable. Cuando ya no es necesaria una conexión, hay que liberarla para desocupar espacio en las tablas de transporte.

La desconexión puede ser:

- Asimétrica, cualquiera de los dos usuarios de transporte puede emitir una primitiva DISCONNECT, se envía un segmento DISCONNECT y libera su conexión. El otro host verá que no le responden los mensajes de confirmación y la libera.
- Simétrica, cada dirección se cierra por separado, independientemente de la otra. Cuando una de las partes emite una primitiva DISCONNECT, ya no tiene más datos por enviar, pero aún está dispuesta a recibir. En este caso, se libera cuando ambas partes han emitido una primitiva DISCONNECT..

Ejemplo de trama completa, con paquete y segmento. [no lo puse más arriba porque se pijea].



Sockets de Berkeley

Las primitivas de socket que se utilizan para TCP. Se utilizan en los sistemas basados en UNIX; hay una API estilo sockets para Windows, llamada winsock. [wincock] [aguante linuzx vieja]

Primitiva	Significado	
SOCKET	Crea un nuevo punto terminal de comunicación.	4 primeras las hacen clientes y servidores en ese orden. Un socket recien creado no tiene dirección de red, se le da una con BIND.
BIND	Asocia una dirección local con un socket.	
LISTEN	Anuncia la disposición de aceptar conexiones; indica el tamaño de la cola.	Listen asigna espacio de cola para poner clientes en espera bloqueados
ACCEPT	Establece en forma pasiva una conexión entrante.	Accept bloquea el proceso y devuelve un descriptor del socket.
CONNECT	Intenta establecer activamente una conexión.	
SEND	Envía datos a través de la conexión.	
RECEIVE	Recibe datos de la conexión.	Cliente crea socket, no requiere BIND. Connect bloquea proceso cliente hasta que se establece conexión
CLOSE	Libera la conexión.	

El servidor:

- SOCKET crea un nuevo punto terminal y le asigna espacio en las tablas de la entidad de transporte. Los parámetros especifican el formato de direccionamiento que se utilizará, el tipo de servicio y el protocolo. Una llamada SOCKET con éxito devuelve un descriptor de archivo que se utiliza con las siguientes llamadas. Los sockets recién creados no tienen direcciones de red. Se asignan mediante
- BIND. Una vez que un servidor ha destinado una dirección, los clientes se pueden conectar. La razón para que no cree directamente una dirección es que algunos procesos se encargan de sus direcciones, mientras que otros no lo hacen.
- LISTEN, asigna espacio para poner en cola las llamadas por si varios clientes intentan conectarse, en el modelo de sockets, LISTEN no es una llamada bloqueante.
- ACCEPT. Bloquea al llamador hasta que haya conexión entrante. Cuando llega un segmento que solicita una conexión, la entidad de transporte crea un socket nuevo con las mismas propiedades que el original y devuelve un descriptor de archivo para él. El servidor puede ramificar un proceso o hilo para manejar la conexión en el socket nuevo y regresar a esperar la siguiente conexión en el socket original. ACCEPT devuelve un descriptor de archivo que se puede utilizar para leer y escribir de la forma estándar, osea con WRITE Y READ.

El cliente:

- Primero crea un socket mediante la primitiva SOCKET, pero no se requiere BIND puesto que la dirección usada no le importa al servidor, se le asigna un puerto aleatorio.
- CONNECT bloquea al invocador y comienza el proceso de conexión. Al completar este proceso, el proceso cliente se desbloquea y se establece la conexión.
- Ambos lados pueden usar ahora SEND y RECEIVE a través de la conexión full-dúplex. READ y WRITE de UNIX también se pueden utilizar.
- La liberación de las conexiones a los sockets es simétrica, se libera cuando ambos ejecutan una primitiva CLOSE.

La API de sockets se utiliza comúnmente con el protocolo TCP para ofrecer un servicio orientado a conexión, conocido como flujo confiable de bytes, que consiste en el conducto de bits confiable, se podrían usar otros protocolos para implementar este servicio mediante el uso de la misma API.

Un punto fuerte de la API es que cualquier aplicación la puede utilizar para otros servicios de transporte. Por ejemplo, con un servicio de transporte sin conexión. CONNECT establece la dirección del transporte del igual remoto, mientras que SEND y RECEIVE envían y reciben datagramas hacia/desde el igual remoto (también es común usar SENDTO y RECEIVEFROM). Los sockets también se pueden usar con protocolos de transporte que proporcionen un flujo continuo de mensajes en vez de un flujo continuo de bytes, y que dispongan o no de un control de congestión.

DCCP (Protocolo de Control de Congestión de Datagramas, del inglés Datagram Congestion Controlled Protocol) es una versión de UDP con control de congestión, trabajan con un grupo de flujos relacionados, un navegador web solicita varios objetos al mismo servidor. Es subóptimo.

Un ejemplo de programación de sockets: un servidor de archivos de Internet [meh, no lo borro por las dudas]

El código del servidor y del cliente se puede compilar y ejecutar en cualquier máquina UNIX, en cualquier parte del mundo. El cliente se puede ejecutar con los parámetros apropiados para obtener cualquier archivo al que el servidor tenga acceso. El archivo se escribe a la salida estándar que, se puede redirigir a un archivo o conducto. En un servidor, la dirección IP se enlaza con el socket y se realiza una verificación para ver si la llamada a bind tuvo éxito. La llamada a listen es para anunciar que el servidor está dispuesto a aceptar llamadas e indicar al sistema que almacene la cantidad determinada de otras llamadas, en caso de que lleguen más solicitudes mientras el servidor aún esté procesando la actual. Si la cola está llena y llegan solicitudes, se descartan. La llamada a accept bloquea el servidor hasta que algún cliente trata de establecer una conexión. Si accept tiene éxito, se devuelve un descriptor que se puede utilizar para leer y escribir. Una vez que se establece la conexión, el servidor lee en ella el nombre del archivo. Si el nombre aún no está disponible, el servidor se bloquea y lo espera. Una vez que obtiene el nombre, abre el archivo y luego entra en un ciclo que alterna leer bloques del archivo y escribirlos en el socket el servidor cierra el archivo y la conexión, y espera hasta que aparezca la siguiente conexión.

El código del cliente inicia, verifica primero si se invocó el código con el número correcto de argumentos (argc=3 significa el nombre del programa más dos argumentos), argv [1] contiene el nombre del servidor gethostbyname lo convierte en una dirección IP. Esta función utiliza DNS para buscar el nombre, se crea e inicializa un socket, el cliente intenta establecer una conexión TCP con connect. Si el servidor está activo y ejecutándose y enlazado al puerto de server, si está inactivo o tiene espacio en su cola listen, la conexión se establecerá. El cliente entra en un ciclo, lee el archivo bloque por bloque desde el socket y lo copia a la salida estándar. Cuando termina, abandona la conexión.

Este servidor no es lo último en tecnología de servidores. Maneja todas las solicitudes estrictamente en forma secuencial, ya que tiene solo un hilo, su desempeño es pobre, no tiene seguridad y utilizar sólo llamadas de sistema UNIX no es la manera de obtener independencia de la plataforma.

ELEMENTOS DE LOS PROTOCOLOS DE TRANSPORTE

El servicio de transporte se implementa entre las dos entidades de transporte, se parecen a los protocolos de enlace de datos, ambos se encargan del control de errores, la secuenciación y el control de flujo. Existen diferencias.

- Capa de enlace de datos, enruteadores se comunican de forma directa mediante un canal físico. Lidia con conexiones punto a punto, no requiere especificar el receptor, cada línea de salida conduce de manera directa a un enruteador específico. La conexión se establece fácilmente pq están unidos por el medio físico los puntos.
- Capa de transporte, el canal físico se sustituye por la red completa. Se requiere el direccionamiento explícito. El establecimiento inicial de la conexión es complicado. Otra diferencia es la existencia potencial de capacidad de almacenamiento en la red. Cuando un enruteador envía un paquete, puede llegar o perderse, pero no debe andar de un lado a otro y aparecer de repente después de otros paquetes.

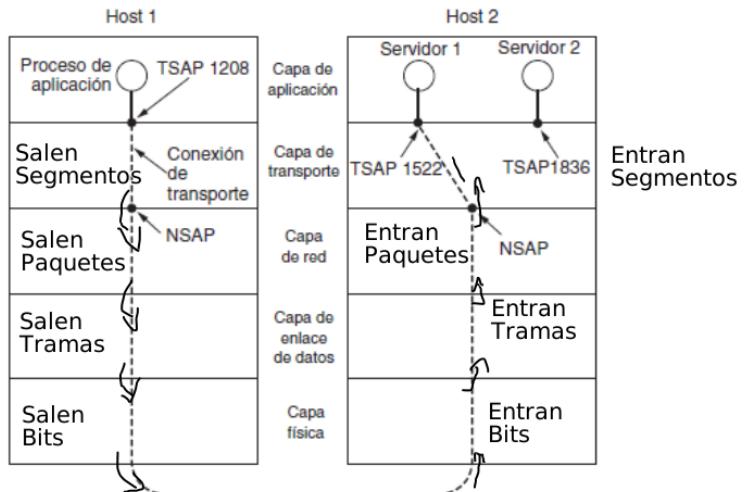
Se requieren búferes y control de flujo en ambas capas, pero la presencia de conexiones grande y variable en la capa de transporte, con un ancho de banda que fluctúa, puede requerir un enfoque distinto. Algunos de los protocolos asignan una cantidad fija de búferes, de modo que, siempre hay un búfer disponible. En la capa de transporte, las conexiones que se deben manejar hacen menos atractiva dedicar muchos búferes a cada una.

Direccionamiento

Cuando un proceso desea establecer una conexión con un proceso de aplicación remoto, debe especificar a cuál se conectará. El método que se emplea es definir direcciones de transporte en las que los procesos puedan escuchar las solicitudes, se denominan puertos:

- TSAP: Transport Service Access Point. Es el puerto genérico en capa de transporte.
- NSAP: Network Service Access Point. Es el análogo en la capa de red. Las direcciones IP son estos puertos.

Possible escenario para una conexión de transporte:



1. Un proceso servidor de correo se enlaza con el TSAP 1522 en el host 2 para esperar una llamada entrante. La manera en que un proceso se enlaza con un TSAP depende del sistema operativo, podría usar una llamada LISTEN.

2. Un proceso de aplicación en el host 1 quiere enviar un mensaje de correo, por lo que se enlaza con el TSAP 1208 y emite una solicitud CONNECT. Esta solicitud especifica el TSAP 1208 en el host 1 como el origen y el TSAP 1522 en el host 2 como destino. Provoca que se establezca una conexión de transporte.

3. El proceso de aplicación envía el mensaje de correo.

4. El servidor de correo responde para decir que entregará el mensaje.

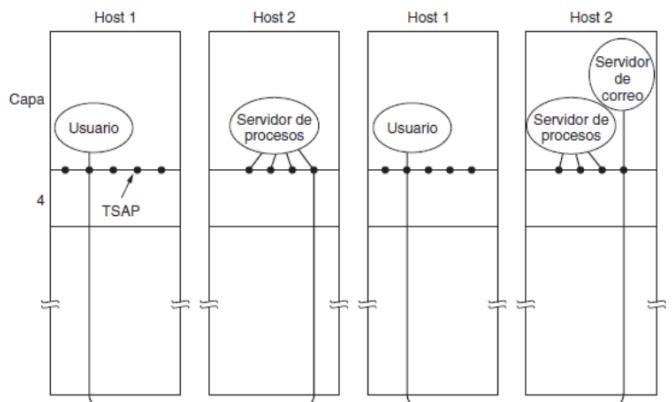
5. Se libera la conexión de transporte.

Algunos servicios tienen direcciones TSAP estándares. Cuando un cliente no sabe la dirección TSAP del servicio al que se quiere conectar, porque no es estándar o pq la cambiaron, se conecta con el proceso "port mapper" o "asignador de puertos". Pasos:

1. Usuario envía un mensaje en el que especifica el nombre del servicio y el asignador de puertos le regresa la dirección TSAP.
2. Usuario libera la conexión con el asignador de puertos y establece una nueva conexión con el servicio deseado.

Cuando se crea un nuevo servicio, se debe registrar con el asignador de puertos para proporcionarle su nombre de servicio y su TSAP. El asignador de puertos registra en su base de datos de manera que cuando empiecen a llegar las consultas, conozca las respuestas. Es imprescindible que la dirección del TSAP bien conocido que utilice el asignador de puertos sea en realidad bien conocida.

Los procesos que no se llaman muy seguidos no están esperando una conexión al pedo. Se usa el Protocolo de conexión inicial; es un servidor de procesos especial, actúa como proxy de los servidores que se usan menos. Se llama inetd en los sistemas UNIX, escucha a un conjunto de puertos al mismo tiempo, en espera de una solicitud de conexión. Los usuarios comienzan emitiendo una solicitud CONNECT, especifican la dirección TSAP del servicio que desean. Si no hay ningún servidor, consiguen una conexión al servidor de procesos. El servidor de procesos genera el servidor solicitado y le permite heredar la conexión existente con el usuario. El nuevo servidor hace el trabajo solicitado, mientras que el servidor de procesos regresa a escuchar nuevas solicitudes, se puede aplicar cuando los servidores se crean bajo demanda.



Establecimiento de una conexión

Para establecer la conexión se usan paquetes de datagramas durante el handshake. Por lo que el sistema debe soportar tiempos de respuestas muy largos, no enviar paquetes duplicados o poder descartarlos.

Soluciones:

- Para tiempos de respuestas muy largos y paquetes duplicados o perdidos: se coloca un mecanismo de tiempo de vida, que cuenta los saltos que hace cada paquete, se descarta cuando llega a cero. Para garantizar que un paquete y sus confirmaciones de recepción estén eliminadas, en el receptor o la red, se espera un tiempo “T”, múltiplo entero del tiempo de vida máxima admitido.
- Para detectar paquetes duplicados que llegan al receptor: se le coloca a cada paquete un identificador o número de secuencia. El número inicial de esta secuencia se negocia al establecer la conexión. Se establece inicialmente como “k” bits del reloj de cada host, es aleatorio. El espacio de secuencia (cuántos números incrementales puedo mandar) debe ser lo suficientemente grande como para no repetir un número de secuencia antes de que pase el tiempo “T”. Acordar el número de secuencia evita tener que sincronizar relojes para contar tiempo y soporta fallas, porque el reloj sigue contando.

Técnicas para restringir el tiempo de vida de un paquete:

1. Un diseño de red restringido. Es difícil, dado que el alcance de las interredes puede variar.
2. Colocar un contador de saltos en cada paquete. Consiste en inicializar el contador de saltos con un valor apropiado y decrementarlo cada vez que se reenvíe. El protocolo de red simplemente descarta cualquier paquete cuyo contador de saltos llega a cero. Se usa este.
3. Marcar el tiempo en cada paquete. Requiere que cada paquete lleve la hora en la que fue creado, y que los enrutadores se pongan de acuerdo en descartar un paquete que haya rebasado cierto tiempo, requiere que los relojes de los enrutadores estén sincronizados, no es una tarea fácil.

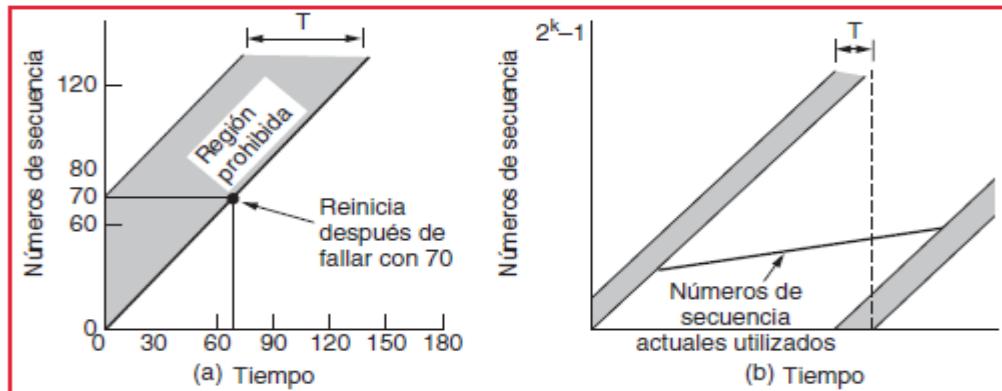
El tiempo de vida máximo del paquete es una constante conservadora para una red; en Internet se toma como 120 segundos. El múltiplo tiene el efecto de hacer a T más largo. Si esperamos T segundos después de enviar un paquete, podemos estar seguros de que todos sus rastros ya han desaparecido. Al limitar los tiempos de vida, es posible rechazar segmentos duplicados con retardo. Este método se utiliza en TCP.

La base del método es que el origen etiquete los segmentos con números de secuencia que no se vayan a reutilizar durante T segundos. T y la tasa de paquetes por segundo determinan el tamaño de los números de secuencia. Sólo un paquete con un número de secuencia específico puede estar pendiente en cualquier momento dado. Puede haber duplicados, en cuyo caso el destino debe descartarlos. Ya no se da el caso en que un duplicado con retardo de un paquete pueda vencer a un nuevo paquete con el mismo número de secuencia y que el destino lo acepte. Para resolver el problema de una máquina que pierde toda la memoria acerca de su estado, es exigir a las entidades de transporte que estén inactivas durante T segundos después de una recuperación. Permitirá que todos los segmentos antiguos expiren, el emisor podrá empezar de nuevo con cualquier número de secuencia. En una interred compleja el periodo T puede ser grande, esta estrategia no es muy atractiva.

Tomlinson propuso equipar cada host con un reloj, no necesitan estar sincronizados, cada reloj tiene la forma de un contador binario que se incrementa a sí mismo a intervalos uniformes, la cantidad de bits del contador debe ser igual o mayor que la cantidad de bits en los números de secuencia y el reloj continúa operando, aunque el host falle. Cuando se establece una conexión, los k bits de menor orden del reloj se usan como número de secuencia inicial. El espacio de secuencia también debe ser lo bastante grande para que, cuando se reinicie, los segmentos antiguos con el mismo número hayan desaparecido. En la figura, la región prohibida muestra los tiempos en donde los números de secuencia de los segmentos son ilegales. Si se envía cualquier segmento en esta región, se podría retrasar y hacerse pasar por un paquete distinto con el mismo número, si el host falla y se reinicia en 70 segundos, utilizará los números de secuencia iniciales con

base en el reloj para continuar a partir de donde se quedó; el host no empieza con un número de secuencia inferior en la región prohibida.

Una vez que ambas entidades han acordado el número de secuencia inicial, se puede usar cualquier protocolo de ventana deslizante para el control de flujo de datos. Este encontrará y descartará exactamente los paquetes duplicados después de que éstos hayan sido aceptados.

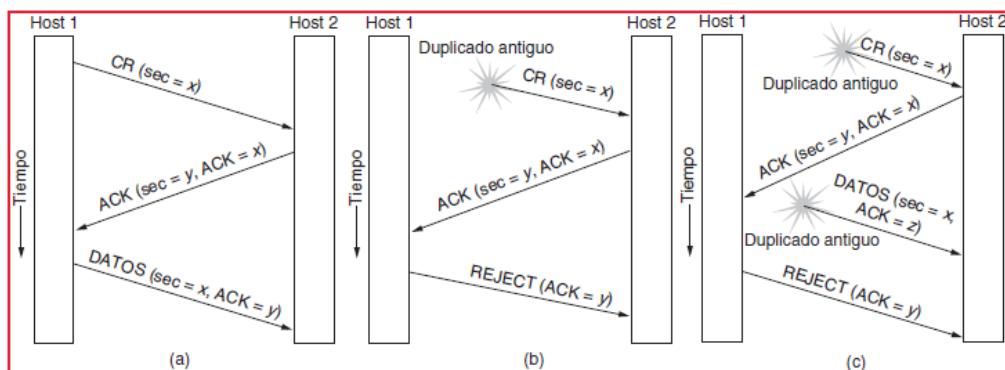


Para mantener los números de secuencia fuera de la región prohibida, necesitamos tener cuidado. Podemos meternos en problemas de dos maneras distintas. Si un host envía muchos datos con demasiada rapidez, el número de secuencia actual contra la curva de tiempo puede subir en forma más pronunciada que el número de secuencia inicial contra la curva de tiempo, provoca que el número de secuencia entre a la región prohibida. Para evitar esto, la tasa máxima de datos es de un segmento por cada pulso de reloj. La entidad de transporte debe esperar hasta que el reloj emita un pulso antes de abrir una nueva conexión después de un reinicio, no sea que el mismo número se utilice dos veces. El reloj no puede pulsar demasiado rápido en relación con el número de secuencia. Para un reloj C y un espacio de números de secuencia de tamaño S, $S/C > T$ para que los números de secuencia no se reinicen con tanta rapidez. A cualquier tasa de datos menor que la tasa de reloj, la curva de números de secuencia actuales utilizados vs el tiempo entrará en un momento dado a la región prohibida desde la izquierda, mientras los números de secuencia se reinician. Entre mayor sea la pendiente de los números de secuencia actuales, más se retardará este evento. Al evitar esta situación se limita el grado de lentitud.

El método basado en reloj resuelve el problema de no poder diferenciar los segmentos duplicados con retardo de los segmentos nuevos, hay un inconveniente, por lo general no recordamos los números de secuencia de una conexión a otra, no tenemos forma de saber si un segmento CONNECTION REQUEST que contiene un número de secuencia inicial es un duplicado. Este inconveniente no existe durante una conexión, ya que la ventana deslizante sí recuerda el número de secuencia actual.

Acuerdo de 3 vías

Para resolver este Tomlinson desarrolló el acuerdo de tres vías (three-way handshake), implica que un igual verifique con el otro que la solicitud de conexión sea realmente actual.



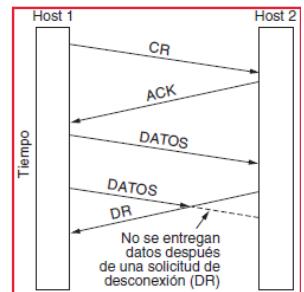
- a) Host 1 escoge un número de secuencia, 'x', y envía al host 2 un segmento CONNECTION REQUEST. El host 2 responde con un segmento ACK para confirmar la recepción de 'x' y anunciar su propio número de secuencia inicial. Finalmente, el host 1 confirma la recepción del número de secuencia inicial seleccionado por el host 2 en el primer segmento de datos que envía.
- b) El primer segmento es un CONNECTION REQUEST duplicado con retardo, llega al host 2 sin el conocimiento del host 1. El host 2 reacciona y envía al host 1 un segmento ACK, para solicitar la comprobación de que el host 1 haya tratado realmente de establecer una nueva conexión. El host 1 rechaza el intento del host 2 por establecer una conexión, el host 2 se da cuenta de que fue un duplicado con retardo y abandona la conexión.
- c) El peor caso, host 2 recibe un CONNECTION REQUEST con retardo y lo contesta, el host 2 ha propuesto usar 'y' como número de secuencia inicial para el tráfico del host 2 al host 1, cuando llega el segundo segmento con retardo al host 2, se confirmó la recepción de 'z' en lugar de 'y', indica al host 2 que éste también es un duplicado antiguo. No hay una combinación de segmentos antiguos que puedan provocar la falla del protocolo y permitan establecer una conexión accidental.

TCP usa este acuerdo de tres vías para establecer las conexiones, se utiliza una estampa de tiempo para extender el número de secuencia de 32 bits, de manera que no se reinicie en un intervalo menor al tiempo de vida máxima del paquete. Esto es una corrección al TCP. Se describe en el RFC 1323 y se llama PAWS (Protección Contra el Reinicio de Números de Secuencia, del inglés Protection Against Wrapped Sequence Numbers). TCP utilizaba originalmente el esquema basado en reloj que describimos antes, se detectó una vulnerabilidad de seguridad: mediante el reloj, un atacante podía predecir con facilidad el siguiente número de secuencia inicial y enviar paquetes que engañaran al acuerdo de tres vías para establecer una conexión falsificada.

Liberación de una conexión

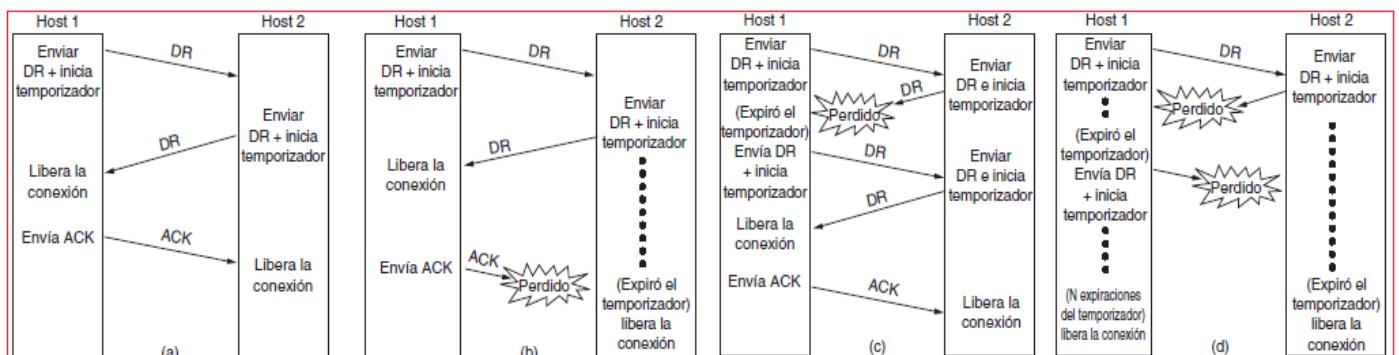
Es más fácil liberar una conexión que establecerla, hay dos estilos para terminar una conexión:

- Asimétrica: una de las partes cuelga, se interrumpe la conexión. Es abrupta y puede provocar la pérdida de datos. Se requiere un protocolo de liberación, para evitar la pérdida de los datos.
- Simétrica: trata la conexión como dos conexiones unidireccionales distintas y requiere que cada una se libere por separado. Cada dirección se libera en forma independiente de la otra, un host puede continuar recibiendo datos, aún después de haber enviado un segmento DISCONNECT. Es ideal cuando cada proceso tiene una cantidad fija de datos por enviar y sabe con certeza cuándo los ha enviado.



El problema de los dos ejércitos: Se basa, informáticamente hablando, en que los hosts no pueden asegurar que, luego de que uno de ellos solicite la desconexión, el otro host recibió esta solicitud. Al agregar mensajes de solicitud y de confirmación, se vuelve una historia de nunca acabar, ya que los hosts esperarían una confirmación de su confirmación, y nunca podrían estar finalmente de acuerdo si se desconectarán o no.

Podemos evitar este dilema al eludir la necesidad de un acuerdo y pasar el problema al usuario de transporte, cada lado puede decidir por su cuenta si se completó o no la comunicación, es un problema más fácil de resolver, mediante el uso de un acuerdo de tres vías. Aunque no es infalible, es adecuado.



- a) Caso normal, uno de los usuarios envía un segmento DR de solicitud de desconexión (DISCONNECTION REQUEST), el receptor devuelve también un segmento DR e inicia un temporizador, por si acaso se pierde su DR. Cuando este DR llega, el emisor original envía de regreso un segmento ACK y libera la conexión. Cuando llega el segmento ACK, el receptor también libera la conexión. La entidad de transporte remueve la información sobre la conexión de su tabla de conexiones abiertas y avisa al dueño de la conexión. Esta acción es diferente en la que el usuario de transporte emite una primitiva DISCONNECT.
- b) Se pierde el ACK, expira el temporizador, la conexión se libera de todos modos.
- c) Se pierde el segundo DR, el usuario que inicia la desconexión no recibirá la respuesta esperada, su temporizador expirará y todo comenzará de nuevo, suponiendo que la segunda vez no se pierden segmentos, y que todos se entregan correctamente y a tiempo.
- d) Todos los intentos repetidos de retransmitir el segmento DR también fallan debido a los segmentos perdidos. Después de N reintentos, el emisor simplemente se da por vencido y libera la conexión. Mientras tanto, expira el temporizador del receptor y también se sale. En teoría puede fallar si se pierden el DR inicial y N retransmisiones. El emisor se dará por vencido y liberará la conexión, pero el otro lado no sabrá nada sobre los intentos de desconexión y seguirá plenamente activo. Esta situación provoca una conexión semiabierta. Pudimos haber evitado este problema obligándolo a seguir insistiendo hasta recibir una respuesta. Si permitimos que expire el temporizador en el otro lado, el emisor continuará por siempre. Si no permitimos que expire el temporizador en el lado receptor, el protocolo queda suspendido.

Para eliminar las conexiones semiabiertas, si no han llegado segmentos durante ciertos segundos, se libera automáticamente la conexión. Esto también se encarga del caso donde se rompe la conexión. Es necesario que cada entidad de transporte tenga un temporizador que se detenga y se reinicie cada vez que se envíe un segmento. Si expira, se transmite un segmento ficticio, para evitar que el otro lado se desconecte. Si se usa la regla de desconexión automática y se pierden demasiados segmentos ficticios, primero se desconectará de forma automática un lado y después el otro.

La liberación de una conexión sin pérdida de datos no es tan simple, el usuario de transporte debe estar involucrado a la hora de decidir cuándo desconectarse; no se puede resolver mediante las entidades de transporte. Mientras TCP realiza un cierre simétrico, muchos servidores web envían al cliente un paquete RST que provoca un cierre repentino de la conexión, es más parecido a un cierre asimétrico. Esto funciona sólo porque el servidor web conoce el patrón del intercambio de datos. Primero recibe una solicitud del cliente, envía una respuesta al cliente. Cuando termina con su respuesta, se han enviado todos los datos en ambas direcciones. El servidor puede enviar una advertencia al cliente y cerrar en forma repentina la conexión. Si el cliente recibe esta advertencia, liberará su estado de conexión. Si no recibe la advertencia, en algún momento detectará que el servidor ya no se está comunicando con él y liberará el estado de la conexión. En cualquiera de los casos, los datos se transfieren con éxito.

Control de errores y almacenamiento en búfer

Control de errores ⇒ Asegura Confiabilidad

Control de Flujo ⇒ Evita Saturación

Usa mecanismos similares a capa de Enlace:

1. Una trama transporta un código de detección de errores para comprobar que la información se haya recibido de manera correcta.
2. Una trama transporta un número de secuencia para identificarse a sí misma; el emisor la retransmite hasta que reciba una confirmación de recepción exitosa por parte del receptor, se le conoce como ARQ (Solicitud Automática de Repetición, del inglés Automatic Repeat reQuest).
3. Hay un número máximo de tramas pendientes que el emisor permitirá, y se detendrá si el receptor no envía confirmaciones de recepción de las tramas. Si este máximo es de un paquete, el protocolo se denomina parada y espera. Las ventanas más grandes permiten canalizaciones y mejoran el desempeño.
4. El protocolo de la ventana deslizante combina estas características y se utiliza para soportar la transferencia de datos bidireccional.

Diferencias entre mecanismos de suma de verificación entre capa de Enlace y Transporte:

Capa de Enlace	Capa de Transporte
Protege una trama mientras atraviesa un solo enlace. Es Valiosas para mejorar el desempeño.	Protege un segmento mientras atraviesa una trayectoria de red completa. Es punto a punto. Esencial para la precisión en la integridad de datos.

Diferencias de grado entre capa de Enlace y Transporte:

Considere las retransmisiones y el protocolo de ventana deslizante, en enlaces inalámbricos, el producto de ancho de banda-retardo es tan pequeño que ni siquiera se puede almacenar una trama completa, un tamaño de ventana pequeño es suficiente para un buen desempeño. Usa un protocolo de parada y espera, transmite o retransmite cada trama y espera una confirmación antes de pasar a la siguiente trama. Un tamaño de ventana más grande aumentaría la complejidad sin mejorar el desempeño.

Para los enlaces cableados, la tasa de errores es tan baja que se pueden omitir las retransmisiones de la capa de enlace, porque las retransmisiones punto a punto repararán la pérdida de tramas residuales. Muchas conexiones TCP tienen un producto de ancho de banda-retardo mucho mayor, una conexión a 1 Mbps y de ida y vuelta de 100 mseg, para esta conexión, se almacenarán 200 Kbits de datos en el receptor, en el tiempo requerido para enviar un segmento y recibir una confirmación. Para estos casos se debe usar una ventana deslizante grande. El protocolo de parada y espera paralizaría el desempeño. Los protocolos de transporte usan ventanas deslizantes más grandes, un host puede tener muchas conexiones, cada una se trata por separado, puede requerir una cantidad considerable de búferes para las ventanas deslizantes. Se necesitan tanto en el emisor como en el receptor. En el emisor para contener todos los segmentos transmitidos, para los cuales todavía no se ha enviado una confirmación de recepción, debido a que se pueden perder y tal vez necesiten retransmitirse. Como el emisor está usando búferes, tal vez el receptor pueda o no dedicar búferes específicos a conexiones específicas, el receptor puede mantener un solo grupo de búferes compartido por todas las conexiones. Cuando entre un segmento, se hará un intento por adquirir un nuevo búfer en forma dinámica. Si hay uno disponible, se acepta el segmento; en caso contrario, se descarta, el emisor está preparado para retransmitir los segmentos perdidos. El emisor simplemente sigue intentando hasta que recibe una confirmación.

La mejor solución entre usar búferes en el origen o usarlos en el destino depende del tipo de tráfico. Para el tráfico en ráfagas con bajo ancho de banda, es razonable no dedicar ningún búfer, sino adquirirlos en forma dinámica en ambos extremos. Para la transferencia de archivos y tráfico de alto ancho de banda, es mejor si el receptor dedica una ventana completa de búferes para permitir que los datos fluyan a la máxima velocidad. Ésta es la estrategia que utiliza TCP. Otra cosa es cómo organizar el grupo de búferes, si la mayoría de los segmentos tienen el mismo tamaño, es natural, un grupo de búferes de tamaño idéntico, con un segmento por búfer, si hay una variación amplia en cuanto al tamaño de los segmentos, un grupo de búferes de tamaño fijo presenta problemas. Si se selecciona de manera que sea igual al segmento más grande, se desperdiciará espacio cada un segmento corto. Si el tamaño menor al tamaño máximo de segmento, se requerirán varios búferes para los segmentos largos. Otra forma es usar búferes de tamaño variable, es un mejor uso de la memoria, aunque el costo de una administración de búferes es complicada. Una tercera posibilidad es dedicar un solo búfer circular grande por conexión, es simple, no depende de los tamaños de los segmentos, pero hace buen uso de la memoria sólo cuando todas las conexiones están muy cargadas. A medida que se abren y cierran conexiones y cambia el patrón del tráfico, el emisor y el receptor necesitan ajustar en forma dinámica sus búferes. El protocolo debe permitir que un host emisor solicite espacio de búfer en el otro extremo. Se podrían asignar búferes por conexión entre los dos hosts. Al conocer su situación de uso de búferes, el receptor podría decir al emisor "Reservé X búferes para tí". Si el número de conexiones aumentara, tal vez sería necesario reducir una asignación.

Una manera de administrar la asignación dinámica es desacoplar los búferes de las confirmaciones de recepción, significa una ventana de tamaño variable. El emisor solicita cierto número de búferes, con base en sus necesidades esperadas. Después el receptor otorga tantos de éstos como pueda. Cada vez que el emisor transmite un segmento debe disminuir su asignación, y cuando ésta llegue a cero debe detenerse por completo. El receptor superpone por separado las confirmaciones de recepción y las asignaciones de búfer en el tráfico inverso. TCP usa este esquema y transmite las asignaciones de búfer en un campo de encabezado llamado Tamaño de ventana.

Acá muestra un ejemplo de la administración dinámica de ventanas en una red de datagramas⇒

Si el espacio de búfer ya no limita el flujo máximo, aparecerá otro cuello de botella: la capacidad de transporte de la red. Si los enrutadores pueden intercambiar k paquetes/seg y hay k trayectorias, no hay manera de que hosts puedan intercambiar más de kx segmentos/seg, sin importar la cantidad de espacio de búfer. Si el emisor presiona demasiado, la red se congestionará. Se necesita un mecanismo que limite las transmisiones del emisor con base en la capacidad de transporte de la red. El emisor ajusta en forma dinámica el tamaño de la ventana para igualarla a la capacidad de transporte de la red. Una ventana deslizante dinámica puede implementar tanto el control de flujo como el de congestión. Si la red puede manejar c segmentos/seg y el tiempo de ida y vuelta es de r , la ventana del emisor debe ser cr . Con una ventana de este tamaño, el emisor opera con el canal a su máxima capacidad. La capacidad de red varía con el tiempo, hay que ajustar el tamaño de ventana con frecuencia.

Conclusión: los errores se controlan con las retransmisiones, la capa de enlace controla con sus sumas de verificación la integridad de datos entre enrutadores, mientras que la capa de transporte lo hace de punto a punto a través de la red.

Por otro lado, según las características del medio de transmisión quizás no es necesario que la capa de enlace pida retransmisiones o quizás sí. En wifi, la capa de enlace transmite de a una trama por vez, se llama protocolo de parada y espera, por si llego rota la trama la vuelve a pedir, en este caso el tamaño de la ventana y del buffer es de una trama, por la naturaleza del medio de transmisión; ya que si manda más tramas de una se rompen más y se vuelve muy ineficiente.

En cambio, por cable, no es necesario mandar de a una trama por vez y si hay pérdida de dato alcanza con que pida retransmisiones la capa de transporte, ahora como podemos mandar más tramas seguidas y hasta en paralelo, los tamaños de buffers, tanto de transmisor como receptor se pueden asignar dinámicamente según lo permita el hardware de los hosts y la capacidad de transmisión de la red. El control de flujo, para no saturar el enlace, lo hace la capa de transporte, ajustando el tamaño de la ventana móvil con la que hace el control de los datos y buffers.

Multiplexión

La multiplexión de varias conversaciones a través de conexiones, circuitos virtuales y enlaces físicos, son importante en varias capas de la arquitectura de red. En la capa de transporte puede surgir la necesidad de usarla, si sólo hay una dirección de red disponible en un host, todas las conexiones tendrán que utilizarla. Con k conexiones de red abiertas, el ancho de banda efectivo se podría incrementar. El SCTP (Protocolo de Control de Transmisión de Flujo, del inglés Stream Control Transmission Protocol) puede operar mediante múltiples interfaces de red. En contraste, TCP utiliza un solo punto terminal de red. La multiplexión inversa también se encuentra en la capa de enlace, cuando se usan varios enlaces de baja velocidad en paralelo como uno solo.

A	Mensaje	B	Comentarios
1	→ < solicita 8 búferes >	→	A quiere 8 búferes.
2	← <ack = 15, buf = 4>	→	B sólo otorga los mensajes 0 a 3.
3	→ <seq = 0, data = m0>	→	A tiene 3 búferes libres ahora.
4	→ <seq = 1, data = m1>	→	A tiene 2 búferes libres ahora.
5	→ <seq = 2, data = m2>	...	Se perdió el mensaje, pero A piensa que le queda 1 libre.
6	← <ack = 1, buf = 3>	→	B confirma la recepción de 0 y 1, permite 2-4.
7	→ <seq = 3, data = m3>	→	A tiene un búfer libre.
8	→ <seq = 4, data = m4>	→	A tiene 0 búferes libres y debe detenerse.
9	→ <seq = 2, data = m2>	→	El temporizador de A expira y retransmite.
10	→ <ack = 4, buf = 0>	→	Todo confirmado, pero A sigue bloqueado.
11	→ <ack = 4, buf = 1>	→	A puede enviar el 5 ahora.
12	→ <ack = 4, buf = 2>	→	B encontró un nuevo búfer en alguna parte.
13	→ <seq = 5, data = m5>	→	A tiene 1 búfer libre.
14	→ <seq = 6, data = m6>	→	A está bloqueado nuevamente.
15	→ <ack = 6, buf = 0>	→	A sigue bloqueado.
16	... <ack = 6, buf = 4>	→	Interbloqueo potencial.

Recuperación de fallas

Si los hosts y los enrutadores están sujetos a fallas o conexiones de larga duración, la recuperación de estas se vuelve un tema importante. Si la entidad de transporte está totalmente dentro de los hosts, la recuperación de fallas de la red y de los enrutadores es sencilla. Las entidades esperan la pérdida de segmentos todo el tiempo y saben cómo lidiar con las retransmisiones. Un problema es recuperarse de las fallas del host, es conveniente que los clientes sean capaces de continuar trabajando cuando los servidores fallan; supongamos que un host envía un archivo grande al servidor de archivos, mediante el protocolo parada y espera. La capa de transporte en el servidor pasa los segmentos al usuario de transporte, uno por uno; a mitad de la transmisión falla. Al reactivarse, sus tablas se reinicializan, ya no sabe precisamente en dónde se encontraba. En un intento por recuperar su estado, el servidor envía un segmento de difusión para anunciar que acaba de fallar y solicitar que le informen sobre el estado de todas las conexiones abiertas. Cada cliente puede estar en uno de dos estados: un segmento pendiente (S1) o ningún segmento pendiente (S0). Con base en esta información de estado, el cliente debe decidir si retransmitirá o no el segmento más reciente. El cliente debe retransmitir sólo si tiene un segmento pendiente al momento de enterarse de la falla. Sin embargo, con esta metodología, la situación en la que servidor envía primero una confirmación de recepción y luego, escribe en el proceso de aplicación, son dos eventos diferentes que no se pueden hacer al mismo tiempo. Si ocurre una falla después de enviar la confirmación, pero antes de que la escritura se complete, el cliente recibirá la confirmación de recepción y estará en el estado S0 cuando llegue el anuncio de recuperación, el cliente no retransmitirá, pues pensará (en forma errónea) que llegó el segmento. Esta provoca que falte un segmento.

Podría pensar: Todo lo que hay que hacer es que primero haga la escritura y luego envíe la confirmación; imagine que se ha hecho la escritura pero que la falla ocurre antes de enviar la confirmación, el cliente se encontrará en el estado S1, y retransmitirá, lo que provocará un segmento duplicado sin detectar en el flujo de salida. Sin importar cómo se programen, siempre hay situaciones que no se puede recuperar de manera apropiada. Podemos programar el servidor en una de dos formas: enviar confirmación de recepción primero o escribir primero; y el cliente de cuatro formas: siempre retransmitir el último segmento, nunca retransmitir el último segmento, retransmitir sólo en el estado S0 o retransmitir sólo en el estado S1. Nos da ocho combinaciones, para cada combinación existe cierto conjunto de eventos que hacen fallar al protocolo.

Son posibles tres eventos en el servidor: enviar una confirmación (A), escribir al proceso de salida (W) y fallar (C). Pueden ocurrir en seis órdenes diferentes: AC(W), AWC, C(AW), C(WA), WAC y WC(A), los paréntesis se usan para indicar que una vez que el servidor falla, así se queda. Hacer más elaborado el protocolo no sirve de nada, el cliente no tiene manera de saber si ha ocurrido una falla justo antes o justo después de la escritura. Es inevitable que la falla de un host y su recuperación no pueden hacerse transparentes a las capas superiores. “La recuperación de una falla en la capa N sólo se puede llevar a cabo en la capa N + 1”, sólo si la capa superior retiene suficiente información del estado como para reconstruir la condición antes de que ocurriera el problema. El protocolo de transporte es de extremo a extremo y no está encadenado como en las capas inferiores. Considere el caso de un usuario que introduce solicitudes de transacciones para una base de datos. Suponga que está programada para pasar primero los segmentos a la siguiente capa superior y luego emitir las confirmaciones, el hecho de que la máquina de un usuario reciba una confirmación de recepción no significa necesariamente que el host remoto actualizó la base de datos. Es imposible lograr una verdadera confirmación de recepción de extremo a extremo.

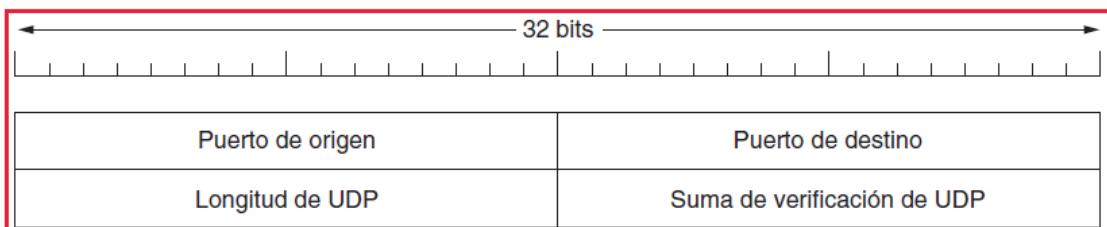
Conclusión: entidad de transporte en hosts, espera que la red falle. Servidor falla, envía un segmento de difusión para anunciar que acaba de fallar y solicitar que le informen sobre el estado de todas las conexiones abiertas. Los clientes tendrán segmentos pendientes de enviar S1 o S0. Como el informe de fallo y recepción de segmentos son asíncronos entre sí, el cliente puede pensar que el segmento enviado se recibió correctamente y falta un segmento. Sin importar cómo se programen, siempre hay situaciones que no se pueden recuperar de manera apropiada. Las combinaciones de fallas dan el tipo de error que hace fallar el protocolo.

LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: UDP

El protocolo sin conexión es UDP, no hace nada más que enviar paquetes entre aplicaciones. El protocolo orientado a conexión es TCP. Realiza las conexiones y agrega confiabilidad mediante las retransmisiones, junto con el control de flujo y el control de congestión. Dos aplicaciones de UDP: un protocolo de capa de transporte que por lo general se ejecuta en el sistema operativo y los protocolos que utilizan UDP por lo general se ejecutan en el espacio de usuario, estos usos se podrían considerar como aplicaciones.

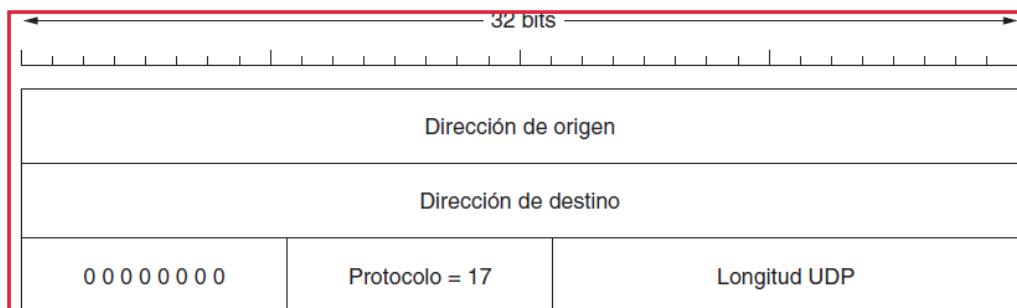
Introducción a UDP

UDP (Protocolo de Datagrama de Usuario, del inglés User Datagram Protocol). UDP proporciona una forma para que las aplicaciones envíen datagramas IP encapsulados sin establecer una conexión. UDP transmite segmentos que consisten en un encabezado de 8 bytes seguido de la carga útil. Los dos puertos sirven para identificar los puntos terminales dentro de las máquinas de origen y destino; cuando llega un paquete UDP, su carga útil se entrega al proceso que está conectado al puerto de destino. El enlace ocurre cuando se utiliza la primitiva BIND. El valor principal de contar con UDP es la adición de los puertos de origen y destino. Sin los puertos, la capa de transporte no sabría qué hacer con cada paquete entrante.



- Puerto de origen se necesita principalmente cuando hay que enviar una respuesta al origen. Al copiar el campo Puerto de origen en el campo Puerto de destino del segmento que sale, el proceso que envía la respuesta puede especificar cuál proceso va a recibirla.
- El campo Longitud incluye el encabezado de 8 bytes y los datos. La longitud mínima es de 8 bytes. La máxima 65 515 bytes (es menor al número que cabe en 16 bits debido al límite de tamaño en los paquetes IP).
- Campo Suma de verificación opcional. Se realiza una suma de verificación para el encabezado, los datos y un pseudo encabezado IP conceptual. Al hacer esto, el campo Suma de verificación se establece en cero y el campo de datos se rellena con un byte cero adicional si su longitud es un número impar. El algoritmo consiste en sumar todas las palabras de 16 bits en complemento a uno y sacar el complemento a uno de la suma. Cuando el receptor realiza el cálculo en todo el segmento, el resultado debe ser 0. Si no se calcula la suma se almacena como 0.

Pseudo Encabezado para el caso de IPv4 contiene las direcciones IPv4 de 32 bits de las máquinas de origen y de destino, el número de protocolo para UDP y la cuenta de bytes para el segmento. Es análogo a IPv6. Es útil incluir el pseudo encabezado en el cálculo de la suma para detectar paquetes mal entregados, pero se viola la jerarquía de protocolos debido a que las direcciones IP pertenecen a la capa IP, no a la capa UDP. TCP usa el mismo pseudo encabezado para su suma.

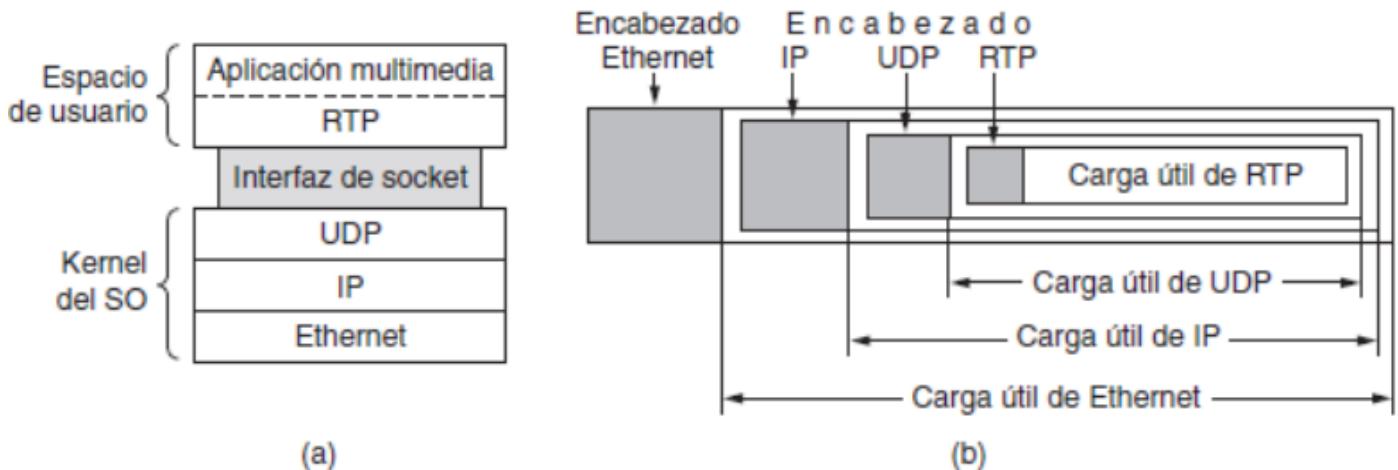


UDP no realiza control de flujo, control de congestión o retransmisión cuando se recibe un segmento erróneo, esto les corresponde a los procesos de usuario, pero sí proporcionar una interfaz para el protocolo IP con la característica agregada de demultiplexar varios procesos mediante puertos y la detección de errores extremo a extremo opcional. Para aplicaciones que sí necesitan control sobre el flujo de paquetes, errores o temporización, TCP es lo que se necesita. UDP es especialmente útil en las situaciones cliente-servidor, el cliente envía una solicitud corta al servidor y espera una respuesta corta. Si se pierde simplemente puede intentar de nuevo. El código requiere menos mensajes en comparación con TCP.

Una aplicación que utiliza UDP es DNS, un programa quiere buscar la dirección IP de algún host, puede enviar al servidor DNS un paquete UDP. El servidor responde con un paquete UDP que contiene la dirección IP. No se necesita configuración ni tampoco una liberación. Sólo dos mensajes.

Protocolo de transporte en Tiempo Real [supuestamente no lo toma, pero ya estamos grandes para creernos esas]

El RTP es un protocolo cliente servidor, se ejecuta en el espacio usuario sobre UDP. Es un protocolo genérico, independiente de la aplicación que lo quiera utilizar que se implementa en capa de aplicación como una biblioteca de servicio cualquiera. Su función básica es multiplexar flujos de datos de tiempo real en un solo flujo de paquetes UDP. Cada paquete de RTP está numerado de manera creciente y es encapsulado en un paquete UDP. Estos luego se pasan a la capa de red que los mete en un paquete IP. El receptor hace el proceso inverso.



Por la propia naturaleza del uso, no es posible realizar retransmisiones (de acá a que llega el paquete que faltaba ya es tarde). Por esto no ofrece ningún mecanismo para pedir retransmisiones. Lo que llega es lo que hay.

Permite asignar “perfiles” a lo que está transmitiendo y asignarle formatos con su correspondiente codificación (mp3, mp4, etc). También permite asignar una estampa de tiempo para que el receptor reproduzca dicho paquete RTM en el momento exacto, temporalmente hablando, desde el inicio de la transmisión, sin importar el orden en el que llegaron los mensajes. (por ejemplo si estás estremeciendo un mp3, reproduce cada muestra de la música en el orden correcto. [si no sería todo ruido]).

LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: TCP [hipermegateraMuchoTexto]

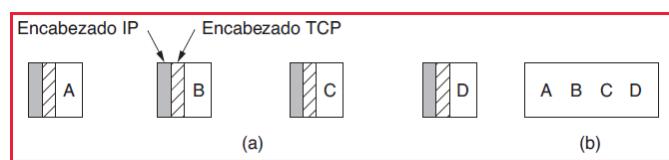
Introducción a TCP

TCP (Protocolo de Control de Transmisión, del inglés Transmission Control Protocol) se diseñó para proporcionar un flujo de bytes confiable de extremo a extremo a través de una interred no confiable, se diseñó para adaptarse de manera dinámica a las propiedades de la interred y sobreponerse a muchos tipos de fallas. Cada máquina que soporta TCP tiene una entidad de transporte TCP, ya sea un procedimiento de biblioteca, un proceso de usuario o sea parte del kernel. Maneja flujos TCP e interactúa con la capa IP, acepta flujos de datos de usuario de procesos locales, los divide en fragmentos que no excedan los 64 KB, por lo general son 1460 bytes, para ajustarlos en una sola trama Ethernet con los encabezados IP y TCP, y envía cada pieza como un datagrama IP independiente. Cuando llegan a una máquina, se pasan a la entidad TCP y reconstruye los flujos de bytes originales. La capa IP no ofrece ninguna garantía de que se entregarán de manera apropiada, ni qué tan rápido. Corresponde a TCP enviar los datagramas con la suficiente rapidez sin provocar una congestión; le corresponde terminar los temporizadores y retransmitir los datagramas. Estos podrían hacerlo en el orden incorrecto, por lo que corresponde a TCP re ensamblarlos en mensajes con la secuencia apropiada. Debe proporcionar un buen desempeño con la confiabilidad que las aplicaciones desean y que IP no proporciona.

El modelo del servicio TCP

TCP se obtiene al hacer que tanto el servidor como el receptor creen puntos terminales, llamados sockets. Tiene un número (dirección) que consiste en la dirección IP del host y un número de 16 bits que es local para ese host, llamado puerto. Es el nombre TCP para un TSAP. Para obtener el servicio TCP, hay que establecer de manera explícita una conexión entre un socket en una máquina y un socket en otra. Podemos usar un socket para múltiples conexiones, dos o más conexiones pueden terminar en el mismo socket, se identifican mediante los identificadores de socket de los dos extremos (en el código se identifican por los descriptores que devuelve la llamada ACCEPT). Los números de puerto menores que 1024 están reservados para los servicios estándar, se llaman puertos bien conocidos.

Se pueden registrar otros puertos del 1024 hasta el 49151 para las aplicaciones. Podría ser posible que el demonio (daemon en inglés) [ES EL SERVIDOR DE PROCESOS QUE VIMOS ANTES], se conecte por sí solo, podría llenar la memoria con demonios que están inactivos. Lo que se hace por lo general es que un solo demonio, llamado demonio de Internet (inetd) en UNIX, se conecte por sí solo a múltiples puertos y espere la primera conexión entrante. Cuando eso ocurre, inetd bifurca un nuevo proceso y ejecuta el demonio apropiado en él, para dejar que ese demonio maneje la solicitud. De esta forma, los demonios distintos a inetd sólo están activos cuando hay trabajo para ellos. Inetd consulta un archivo de configuración para saber cuál puerto utilizar. En consecuencia, el administrador del sistema puede configurar el sistema.



Puerto	Protocolo	Uso
20, 21	FTP	Transferencia de archivos.
22	SSH	Inicio de sesión remoto, reemplazo de Telnet.
25	SMTP	Corre electrónico.
80	HTTP	World Wide Web.
110	POP-3	Acceso remoto al correo electrónico.
143	IMAP	Acceso remoto al correo electrónico.
443	HTTPS	Acceso seguro a web (HTTP sobre SSL/TLS).
543	RTSP	Control del reproductor de medios.
631	IPP	Compartición de impresoras.

Todas las conexiones TCP son full dúplex y de punto a punto, el tráfico puede ir en ambas direcciones al mismo tiempo. TCP no soporta la multidifusión ni la difusión, es un flujo de bytes, no un flujo de mensajes. Los límites de los mensajes no se preservan de un extremo a otro. No hay manera de que el receptor detecte la(s) unidad(es) en la(s) que se escribieron los datos. Los archivos de UNIX también tienen esta propiedad, TCP no tiene idea de lo que significan los bytes, un byte es sólo un byte. Cuando una aplicación pasa datos a TCP, éste decide entre enviarlos de inmediato o almacenarlos en el búfer, algunas veces la aplicación necesita que se envíen de inmediato y no se almacenen en el búfer. Para forzar la salida de los datos, TCP tiene la noción de una bandera PUSH que se transporta en los paquetes.

El protocolo TCP

Cada byte de una conexión TCP tiene su propio número de secuencia de 32 bits, se transmiten en paquetes para la posición de ventana deslizante en una dirección, y para las confirmaciones de recepción en la dirección opuesta. La entidad TCP emisora y receptora intercambian datos en forma de segmentos, este consiste en un encabezado fijo de 20 bytes seguido de cero o más bytes de datos. El software de TCP decide qué tan grandes deben ser los segmentos. Hay dos límites que restringen el tamaño:

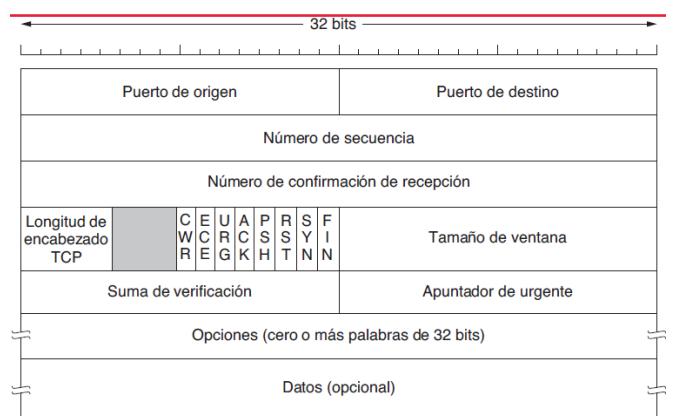
- Cada segmento, incluido el encabezado, debe caber en la carga útil de 65 515 bytes del IP.
- Cada segmento debe caber en la MTU en el emisor y el receptor, de modo que se pueda enviar y recibir en un solo paquete sin fragmentar. El MTU es por lo general de 1500 bytes, es la carga útil en Ethernet y define el límite superior en el tamaño de segmento.

Es posible que los paquetes IP se fragmenten al pasar por una trayectoria de red que tenga una MTU pequeña, degradará el desempeño y provocará otros problemas, las implementaciones modernas de TCP realizan el descubrimiento de MTU de la ruta. Esta técnica usa mensajes de error de ICMP para encontrar la MTU más pequeña. Después, TCP ajusta el tamaño del segmento para evitar la fragmentación. El protocolo que utilizan las entidades TCP es el protocolo de ventana deslizante con un tamaño dinámico. Cuando un emisor transmite un segmento, inicia un temporizador. Cuando llega el segmento al destino, la entidad TCP receptora devuelve un segmento que contiene un número de confirmación igual al siguiente número de secuencia que espera recibir, junto con el tamaño de la ventana remanente. Si el temporizador del emisor expira antes de recibir la confirmación de recepción, el emisor transmite de nuevo el segmento. Las retransmisiones podrían incluir rangos de bytes diferentes a los de la transmisión original si requiere una administración cuidadosa para llevar el control.

El encabezado del segmento TCP

Cada segmento comienza con un encabezado de formato fijo de 20 bytes, puede ir seguido de encabezado de opciones. Pueden continuar 65 495 bytes de datos, donde los primeros 20 se refieren al encabezado IP y los segundos al encabezado TCP. Los segmentos sin datos se usan por lo común para confirmaciones de recepción y mensajes de control.

- Puerto de origen y Puerto de destino identifican los puntos terminales locales de la conexión. Un puerto TCP más la dirección IP forman un punto terminal. Los puntos terminales de origen y de destino en conjunto identifican la conexión. Se denomina 5-tupla, ya que consiste en el protocolo (TCP), IP de origen y puerto de origen, IP de destino y puerto de destino.
- Número de secuencia y número de confirmación de recepción desempeñan sus funciones normales. El segundo especifica el siguiente byte en el orden esperado, no el último byte de manera correcta recibido. Es una confirmación de recepción acumulativa debido a que sintetiza los datos recibidos con un solo número. No va más allá de los datos perdidos. Ambos tienen 32 bits de longitud.
- Longitud del encabezado indica la cantidad de palabras de 32 bits contenidas en el encabezado, es necesaria porque el campo Opciones es de longitud variable. Este campo indica el comienzo de los datos en el segmento, medido en palabras de 32 bits.
- Campo de 4 bits que no se usa.
- Ocho banderas de 1 bit. CWR y ECE para indicar congestión. Cuando se usa ECN (Notificación Explícita de Congestión), ECE se establece para indicar a un emisor que reduzca su velocidad. CWR se establece para indicar una Ventana de congestión reducida del emisor TCP al receptor TCP, de modo que sepa que el emisor redujo su velocidad y puede dejar de enviar la Repetición de ECN.
- URG se establece en 1 si está en uso el Apuntador urgente. El Apuntador urgente se usa para indicar un desplazamiento en bytes a partir del número de secuencia actual en el que se deben encontrar los datos urgentes. Se usa muy poco.



- ACK se establece en 1 para indicar que el número de confirmación de recepción es válido. Éste es el caso para casi todos los paquetes. Si ACK es 0, el segmento no contiene una confirmación, por lo que se ignora el campo Número de confirmación de recepción.
- PSH indica datos que se deben transmitir de inmediato (PUSHed data) y no los almacene en búfer.
- RST se usa para restablecer una conexión que se ha confundido debido a una falla. También se usa para rechazar un segmento no válido o un intento de abrir una conexión.
- SYN se usa para establecer conexiones. La solicitud de conexión tiene SYN = 1 y ACK = 0, la respuesta de conexión sí lleva una confirmación SYN = 1 y ACK = 1. El bit SYN se usa para denotar CONNECTION REQUEST y CONNECTION ACCEPTED, y el bit ACK sirve para distinguir entre ambas.
- FIN se usa para liberar una conexión y especifica que el emisor no tiene más datos que transmitir. Sin embargo, puede continuar recibiendo datos de manera indefinida. SYN y FIN tienen números de secuencia y, se garantiza su procesamiento en el orden correcto.

El control de flujo se maneja mediante una ventana deslizante de tamaño variable.

- Tamaño de ventana indica la cantidad de bytes que se pueden enviar, comenzando por el byte cuya recepción se ha confirmado. Un campo de Tamaño de ventana de 0 es válido e indica que se han recibido los bytes hasta Número de confirmación de recepción – 1, y ya no desea más datos por el momento. El receptor puede otorgar permiso más tarde para enviar mediante la transmisión de un segmento con el mismo Número de confirmación de recepción y un campo Tamaño de ventana distinto de cero.
- Suma de verificación para agregar confiabilidad. Realiza una suma del encabezado, los datos y un pseudo encabezado conceptual exactamente de la misma forma que UDP, sólo que el pseudo encabezado tiene el número de protocolo para TCP y es obligatoria.
- Opciones ofrece una forma de agregar las características adicionales. Se han definido muchas opciones, llenan un múltiplo de 32 bits mediante la técnica de relleno con ceros y se pueden extender hasta 40 bytes, se transmiten cuando se establece una conexión para negociar o informar al otro extremo.
 - MSS (Tamaño Máximo de Segmento, del inglés Maximum Segment Size) que está dispuesto a aceptar. Durante el establecimiento de la conexión, cada lado puede anunciar su máximo y ver el de su compañero. Si un host no usa esta opción, tiene una carga útil predeterminada de 536 bytes. Se requiere que todos los hosts de Internet acepten segmentos TCP de $536 + 20 = 556$ bytes. No es necesario que el tamaño máximo de segmento en ambas direcciones sea el mismo.
 - Escala de ventana permite al emisor y al receptor negociar un factor de escala de ventana al inicio de una conexión. Ambos lados utilizan el factor de escala para desplazar el campo Tamaño de ventana hasta un máximo de 14 bits a la izquierda, con lo cual se permiten ventanas de hasta 230 bytes.
 - Estampa de tiempo transmite una estampa de tiempo enviada por el emisor y repetida por el receptor. Se incluye en todos los paquetes y se utiliza para calcular muestras del tiempo de ida y vuelta para estimar cuándo se ha perdido un paquete. También se utiliza como extensión lógica del número de secuencia de 32 bits. En una conexión rápida, el número de secuencia se puede reiniciar muy rápido, puede provocar una confusión entre los datos viejos y los nuevos. El esquema PAWS (Protección Contra el Reinicio de Números de Secuencia, del inglés Protection Against Wrapped Sequence Numbers) descarta los segmentos entrantes que tengan estampas de tiempo viejas.
 - SACK (Confirmación de Recepción Selectiva, del inglés Selective ACKnowledgement) permite a un receptor indicar al emisor los rangos de números de secuencia que ha recibido. Se utiliza después de haber perdido un paquete y de la llegada de los datos subsiguientes. Los nuevos datos no se reflejan mediante el campo Número de confirmación de recepción en el encabezado debido a que sólo proporciona el siguiente byte en el orden que se espera. Con SACK, el emisor está consciente de los datos que tiene el receptor y, puede determinar qué datos se deben retransmitir.

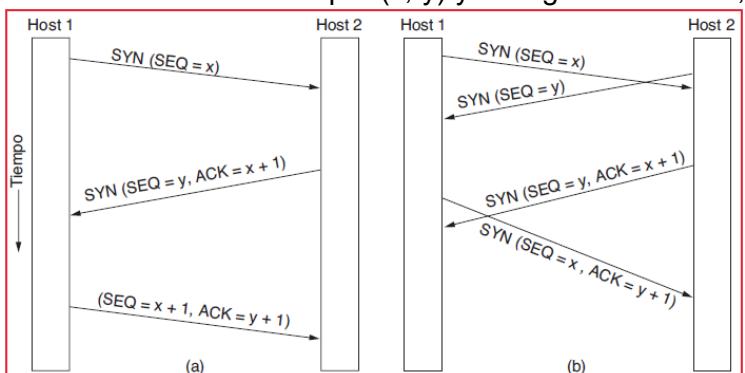
En TCP, las confirmaciones de recepción y los permisos para enviar datos adicionales son por completo independientes.

Establecimiento de una conexión TCP

Las conexiones se establecen mediante el acuerdo de tres vías. Para establecer una conexión, el servidor espera en forma pasiva una conexión entrante mediante la ejecución de las primitivas LISTEN y ACCEPT. El cliente ejecuta una primitiva CONNECT en la que especifica la dirección y el puerto con el que se desea conectar, el tamaño máximo de segmento a aceptar y de manera opcional algunos datos de usuario. CONNECT envía un segmento TCP con el bit SYN encendido y el bit ACK apagado, y espera una respuesta.

Cuando este segmento llega al destino, la entidad TCP de ahí revisa si hay un proceso que haya ejecutado una primitiva LISTEN en el puerto que se indica en el campo Puerto de destino. Si no lo hay, envía una respuesta con el bit RST encendido para rechazar la conexión. Si algún proceso está escuchando en el puerto, recibe el segmento TCP entrante y puede aceptar o rechazar la conexión. Si la acepta, se devuelve un segmento de confirmación. Un segmento SYN consume 1 byte de espacio de secuencia, por lo que se puede reconocer sin ambigüedades. En el caso en que dos hosts intenten establecer al mismo tiempo una conexión entre los mismos dos sockets. El resultado de estos eventos es que sólo se establece una conexión y no dos, si el primer establecimiento resulta en una conexión identificada por (x, y) y el segundo también, sólo se hace una entrada de tabla; para (x, y) .

El número de secuencia inicial elegido por cada host se debe reiniciar con lentitud en vez de ser una constante tal como 0, es para protegerse contra los paquetes duplicados. En un principio se lograba basado en reloj, emitía un pulso cada 4 μ seg. Ahora se hace mediante PAWS (protección contra reinicio de números de secuencia).



Liberación de una conexión TCP

Aunque las conexiones TCP son full dúplex, es mejor visualizarlas como un par de conexiones simplex. Cada conexión simplex se libera de manera independiente de su igual. Cualquiera de las partes puede enviar un segmento TCP con el bit FIN establecido, significa que no tiene más datos por transmitir. Al confirmarse la recepción de FIN, se apaga ese sentido. Sin embargo, los datos pueden seguir fluyendo de manera indefinida por el otro sentido. Cuando se apagan ambos sentidos, se libera la conexión. Requieren cuatro segmentos TCP para liberar una conexión: un FIN y un ACK para cada sentido, es posible que el primer ACK y el segundo FIN estén contenidos en el mismo segmento, con lo cual se reduce la cuenta total a tres.

Ambos extremos pueden enviar segmentos FIN al mismo tiempo. No hay diferencia entre la liberación secuencial o simultánea. Para evitar el problema de los dos ejércitos se usan temporizadores.

Modelado de administración de conexiones TCP

Los pasos requeridos para establecer y liberar conexiones. En cada estado son legales ciertos eventos. Al ocurrir un evento legal, debe emprenderse alguna acción. Si ocurre algún otro evento, se reporta un error. Cada conexión comienza en el estado CLOSED y deja ese estado cuando hace una apertura pasiva (LISTEN) o activa (CONNECT). Si el otro lado realiza la acción opuesta, se establece una conexión y se vuelve ESTABLISHED. La liberación de la conexión se puede iniciar desde cualquiera de los dos lados. Al completarse, regresa a CLOSED. El caso común de un cliente que se conecta activamente a un servidor pasivo (se indica con líneas gruesas, punteadas para el servidor). Las líneas delgadas son secuencias de eventos no usuales. Cada línea se marca mediante un par evento/acción. Puede ser una llamada de sistema (CONNECT, LISTEN, SEND o CLOSE), la llegada de un segmento (SYN, FIN, ACK o RST) o, una expiración de temporizador. El envío de un segmento de control (SYN, FIN o RST) o nada, se indica mediante (—).

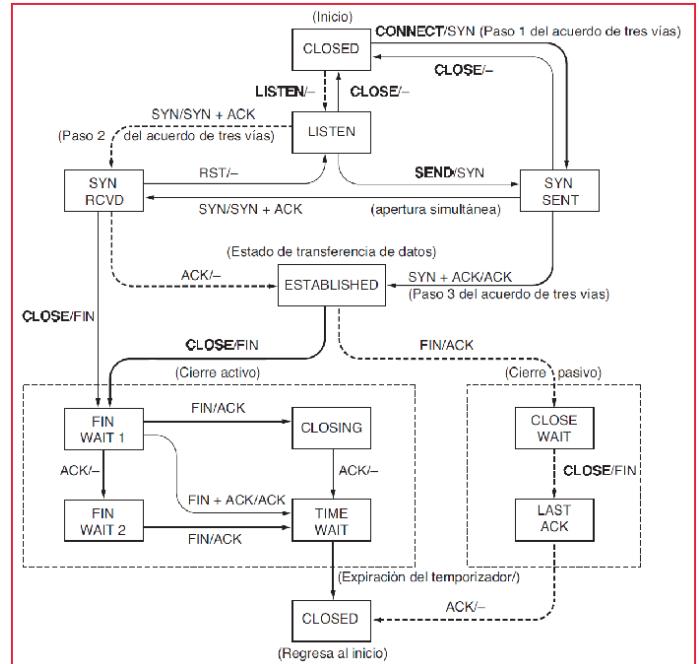
Seguimos primero la trayectoria de un cliente y luego la de un servidor. Un programa emite una solicitud CONNECT, la entidad TCP local crea un registro de conexión, está en el estado SYN SENT y envía un

segmento SYN. Muchas conexiones pueden estar abiertas al mismo tiempo el estado es por conexión y se graba en el registro de conexiones. Al llegar el SYN + ACK, TCP envía el último ACK del acuerdo de tres vías y cambia al estado ESTABLISHED. Ahora se pueden enviar y recibir datos.

Cuando una aplicación ha terminado, ejecuta una primitiva CLOSE; envíe un segmento FIN y espere el ACK. Al llegar el ACK, se hace una transición al estado FIN WAIT 2 y se cierra un sentido de la conexión. Cuando cierra también el otro lado llega un FIN, para el cual se envía una confirmación de recepción. Ahora ambos lados están cerrados, pero el TCP espera un tiempo para garantizar que todos los paquetes de la conexión hayan expirado, sólo por si acaso se perdió la confirmación de recepción. Al expirar el temporizador, TCP borra el registro de la conexión.

Desde el punto de vista del servidor, emite una solicitud LISTEN y se detiene a esperar a que aparezca alguien. Cuando llega un SYN, se envía una confirmación de recepción y el servidor pasa al estado SYN RCVD. Cuando llega la confirmación de recepción del SYN del servidor, el acuerdo de tres vías se completa y el servidor pasa al estado ESTABLISHED. Ahora puede ocurrir la transferencia de datos.

Cuando el cliente termina, emite una solicitud CLOSE; esto provoca la llegada de un FIN al servidor, se envía una señal al servidor. Cuando éste también emite una solicitud CLOSE, se envía un FIN al cliente. Al llegar la confirmación de recepción del cliente, el servidor libera la conexión y elimina el registro de conexión.



Ventana deslizante de TCP [el tafe suele dar hasta acá, el resto lo anotó el manija el fede porque le pegaban de chiquito]

Suponga que el receptor tiene un búfer de 4 096 bytes, el emisor transmite un segmento de 2 048 bytes que se recibe correctamente, el receptor enviará la confirmación de recepción del segmento. Dado que ahora sólo tiene 2 048 bytes de espacio de búfer anunciará una ventana de 2048 comenzando con el siguiente byte esperado. El emisor envía otros 2 048 bytes, el receptor envía la confirmación de recepción, pero la ventana anunciada tiene un tamaño de 0. El emisor debe detenerse hasta que el receptor retire algunos datos del búfer, TCP podrá anunciar una ventana más grande y se podrán enviar más datos.

Cuando la ventana es de 0, el emisor no puede enviar segmentos, salvo en dos situaciones, se pueden enviar datos urgentes o el emisor puede enviar un segmento de 1 byte para hacer que el receptor vuelva a anunciar el siguiente byte esperado y el tamaño de la ventana. Este se denomina sonda de ventana, para evitar un interbloqueo si llega a perderse una actualización de ventana. No se requiere que los emisores transmitan datos tan pronto como llegan de la aplicación. Tampoco que los receptores envíen confirmaciones de recepción tan pronto como sea posible. Considere una conexión a una terminal remota; mediante SSH o telnet, que reacciona con cada pulso de tecla. En conjunto se usa ancho de banda, y este escasea.

Un enfoque que usa para optimizar esta situación es el de las confirmaciones de recepción con retardo. La idea es retrasar las confirmaciones de recepción y las actualizaciones de ventana por hasta 500 mseg, con la esperanza de que lleguen algunos datos. Suponiendo que la terminal hace eco en un lapso de 500 mseg, ahora el lado remoto sólo necesita enviar de vuelta un paquete, con lo cual se recorta a la mitad la cuenta

de paquetes y el uso de ancho de banda. Aunque las confirmaciones de recepción con retardo reducen la carga en la red, un emisor que envía varios paquetes cortos, aún opera de manera ineficiente. El algoritmo de Nagle es sencillo: Si la aplicación envía muchas piezas de datos, el algoritmo colocará todas las diversas piezas en un segmento, con lo cual se reducirá de manera considerable el ancho de banda utilizado. El algoritmo establece que se debe enviar un nuevo segmento si se acumularon suficientes datos como para llenar un segmento máximo. Se usa mucho en las implementaciones de TCP, pero hay veces en que es mejor deshabilitarlo, en los juegos interactivos que operan a través de Internet. Un problema es que en ocasiones el algoritmo de Nagle puede interactuar con las confirmaciones de recepción con retardo para provocar un interbloqueo temporal: el receptor espera los datos sobre los cuales superponer una confirmación de recepción, y el emisor espera la confirmación de la recepción para enviar más datos. Puede retardar las descargas, el algoritmo de Nagle se puede deshabilitar con TCP_NODELAY.

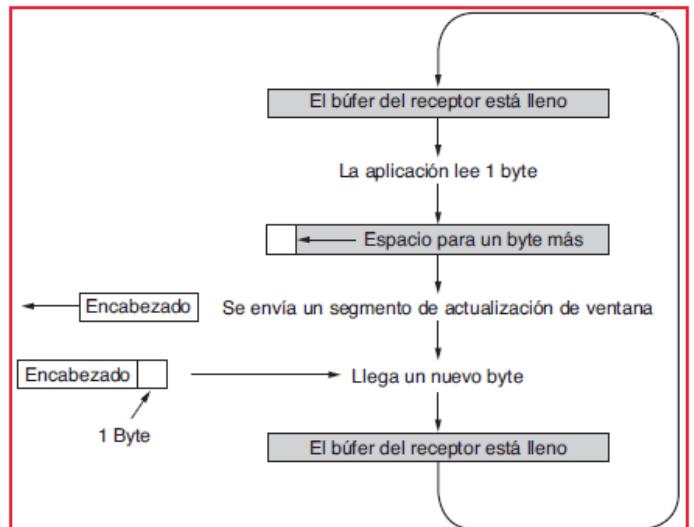
Otro problema de TCP es el síndrome de ventana tonta, cuando se pasan datos a la entidad TCP emisora en bloques grandes, pero una aplicación interactiva del lado receptor lee datos sólo a razón de 1 byte a la vez. El búfer TCP del lado receptor está lleno y el emisor lo sabe que la aplicación interactiva lee un carácter del flujo TCP. El receptor TCP envía una actualización de ventana al emisor para que envíe 1 byte. El emisor accede y envía 1 byte. El búfer ahora está lleno, por lo que el receptor confirma la recepción del segmento de 1 byte y establece la ventana a 0. Este comportamiento puede continuar por siempre. La solución es evitar que el receptor envíe una actualización de ventana para 1 byte. Se le obliga a esperar tener una cantidad decente de espacio, hasta que pueda manejar el tamaño máximo de segmento que anunció al establecerse la conexión o a la mitad de capacidad. El emisor puede ayudar al no enviar segmentos muy pequeños, hasta que pueda enviar un segmento completo o la mitad del tamaño del búfer del receptor.

El objetivo es que el emisor no envíe segmentos pequeños y que el receptor no los pida. El receptor TCP puede bloquear una solicitud READ de la aplicación hasta que pueda proporcionarle un bloque grande de datos. Se reduce la cantidad de llamadas a TCP, aumenta el tiempo de respuesta, pero en las aplicaciones no interactivas, la eficiencia puede ser más importante. Otro problema es que los segmentos pueden llegar fuera de orden. El receptor colocará los datos en el búfer hasta que se puedan pasar en orden a la aplicación. Las confirmaciones de recepción se pueden enviar sólo después de haber recibido todos los datos. Se le conoce como confirmación de recepción acumulativa. Si el receptor recibe los segmentos 0, 1, 2, 4, 5, 6 y 7, puede enviar una confirmación de recepción de todos los bytes hasta el último byte del segmento 2, inclusive. Al expirar el temporizador del emisor, éste retransmitirá el segmento 3. Como el receptor ha puesto en el búfer los segmentos 4 a 7, al recibir el segmento 3 puede enviar una confirmación de recepción de todos los bytes hasta el final del segmento 7.

Administración de temporizadores de TCP

TCP usa varios temporizadores, el más importante es el RTO (Temporizador de Retransmisión, del inglés Retransmission TimeOut). Cuando se envía un segmento, se inicia un temporizador de retransmisión. Si la confirmación llega antes de que expire, se detiene. Si el temporizador termina antes se retransmite y se inicia de nuevo. Surge la pregunta: ¿qué tan grande debe ser el intervalo? Este problema es mucho más difícil en la capa de transporte que en los protocolos de enlace de datos.

Las confirmaciones de recepción pocas veces se retardan en la capa de enlace de datos. La función de densidad de probabilidad del tiempo de confirmación de recepción TCP es grande y variable. Es complicado determinar el tiempo de ida y vuelta, es difícil decidir sobre el intervalo de expiración. Si se establece demasiado corto, ocurrirán retransmisiones innecesarias. Si se establece demasiado largo, el desempeño sufrirá debido al largo retardo. La solución es usar un algoritmo dinámico que ajuste de manera constante el intervalo de expiración del temporizador, con base en mediciones continuas del desempeño de la red. Funciona de la siguiente manera: TCP mantiene una variable llamada SRTT (Tiempo de Ida y Vuelta Suavizado), que es la mejor estimación actual del tiempo de ida y vuelta, se inicia un temporizador, TCP



mide el tiempo y luego actualiza el SRTT. Este tipo de fórmula es un EWMA (Promedio Móvil Ponderado Exponencialmente, del inglés Exponentially Weighted Moving Average). Cuando la carga se acerca a la capacidad máxima, el retardo se hace grande y muy variable. Para corregir este problema, se propuso hacer que el valor de expiración del temporizador fuera sensible a la diferencia en los tiempos de ida y vuelta, así como al tiempo de ida y vuelta suavizado. Hay que llevar el registro RTTVAR (Variación de Tiempo de Ida y Vuelta), no es la desviación estándar, pero es la desviación media. Con esto se calcula el RTO que es el tiempo de expiración de retransmisión. Un problema que ocurre con la recopilación de las muestras, R, es qué hacer cuando expira el temporizador. Cuando llega la confirmación de recepción, no está claro si ésta se refiere a la primera transmisión o a una posterior. Se hizo una propuesta sencilla: no actualizar las estimaciones sobre ninguno de los segmentos retransmitidos. El otro temporizador que TCP utiliza es el temporizador de persistencia, es el segundo de ellos. Está diseñado para evitar el siguiente interbloqueo. El receptor envía una confirmación de recepción con un tamaño de ventana de 0 para indicar al emisor que espere. Después actualiza la ventana, pero se pierde el paquete. Cuando expira el temporizador de persistencia, el emisor transmite un sondeo al receptor. La respuesta proporciona el tamaño de la ventana. Si aún es cero, se inicia el temporizador de persistencia una vez más, si es diferente se pueden enviar datos.

Un tercer temporizador, es el temporizador de seguir con vida (keepalive). Cuando una conexión ha estado inactiva durante demasiado tiempo, el temporizador puede expirar para que compruebe que el otro aún está ahí. Si no se recibe respuesta, se termina la conexión. El último temporizador es el que se usa en el estado TIME WAIT durante el cierre, durante el doble del tiempo máximo de vida de paquete para asegurar que todos los paquetes creados hayan desaparecido.

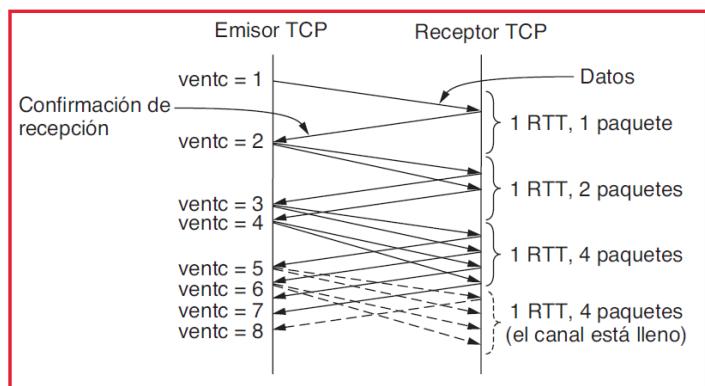
Control de congestión en TCP

Función clave de TCP: el control de congestión. Cuando la carga red es mayor que la que puede manejar, se genera una congestión. La capa de red detecta la congestión cuando las colas crecen demasiado. Es responsabilidad de la capa de transporte recibir la retroalimentación de la capa de red y reducir la tasa del tráfico, TCP desempeña el control de la congestión, así como en el transporte confiable. Un protocolo de transporte, que utiliza una ley de control AIMD (Incremento Aditivo/Decremento Multiplicativo) en respuesta a las señales de congestión binarias provenientes de la red, converge hacia una asignación de ancho de banda equitativa y eficiente. TCP se basa en una ventana y con la pérdida de paquetes como la señal binaria. TCP mantiene una ventana de congestión cuyo tamaño es el número de bytes que puede tener el emisor. La tasa es el tamaño de ventana dividido entre el tiempo de viaje. TCP ajusta el tamaño de la ventana, de acuerdo con la regla AIMD. La ventana de congestión se mantiene además de la ventana de control de flujo, la cual especifica el número de bytes que el receptor puede colocar en el búfer, el número de bytes que se puede enviar es el menor de las dos ventanas. TCP dejará de enviar datos si la ventana de congestión o la de control de flujo está temporalmente llena. Si el receptor dice "envía 64 KB" pero el emisor sabe que las ráfagas mayores de 32 KB obstruyen la red, enviará 32 KB. Por otro lado, si el receptor dice "envía 64 KB" y el emisor sabe que las ráfagas de hasta 128 KB pasan sin problema, enviará los 64 KB. Para usar la pérdida de paquetes como una señal de congestión es necesario que los errores de transmisión sean relativamente raros. No es así para 802.11, incluyen su propio mecanismo. TCP en Internet suponen que los paquetes perdidos se deben a la congestión, por lo cual monitorean las expiraciones de los temporizadores. Se requiere corregir este temporizador mediante la inclusión del factor de variación fue un paso importante en el trabajo realizado por Jacobson. Si se tiene un buen temporizador de retransmisión, el emisor de TCP analiza la diferencia entre los números de secuencia que se transmiten y los números de secuencia cuya confirmación de recepción se ha enviado. Tenemos que rastrear la ventana de congestión mediante el uso de los números de secuencia y los que ya se han confirmado, y ajustarla mediante el uso de una regla AIMD. Una de las primeras consideraciones es que la forma en que se envían los paquetes a la red debe coincidir con la trayectoria de red. En caso contrario, el tráfico provocará una congestión. Ejemplo, un host con una ventana de congestión de 64 KB conectado a una red Ethernet comutada de 1

Gbps, envía toda la ventana a la vez, esta ráfaga puede viajar a través de una línea ADSL lenta de 1 Mbps. Este podría provocar congestión.

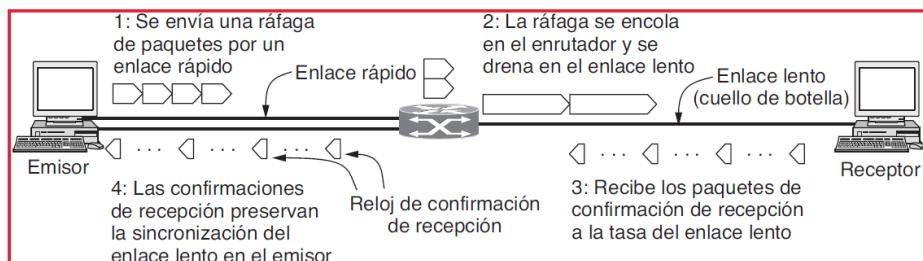
El reloj de confirmación de recepción (ack clock). Mediante el uso de un reloj de confirmación de recepción, TCP regula el tráfico y evita las colas innecesarias en los enrutadores. Una segunda consideración es que la regla AIMD tardará mucho tiempo para llegar a un buen punto de operación en las redes rápidas si la ventana de congestión se inicia a partir de un tamaño pequeño. Podríamos reducir este tiempo de inicio si empezáramos con una ventana inicial más grande, pero esta ventana sería demasiado grande para los enlaces lentos o cortos. Provocaría congestión. La solución para manejar ambas consideraciones es una mezcla de incremento lineal y de incremento multiplicativo. El emisor inicializa la ventana de congestión con un pequeño valor inicial, de cuatro segmentos: envía la ventana inicial. Los paquetes tardarán un tiempo de ida y vuelta para que se confirme su recepción. Para cada segmento cuya recepción se confirme, el emisor agrega a la ventana de congestión una cantidad de bytes equivalente a un segmento. Cada segmento confirmado permite enviar dos segmentos más. La ventana de congestión se duplica cada tiempo de ida y vuelta. Este algoritmo se conoce como inicio lento (slow start), su crecimiento es exponencial.

Funciona bien sobre un rango de velocidades de enlaces y tiempos de ida y vuelta. Cuando el emisor recibe una confirmación de recepción, incrementa en uno la ventana de congestión y envía de inmediato dos paquetes a la red, estos no llegarán necesariamente con un espaciamiento tan estrecho como cuando se enviaron. Si la trayectoria de la red es lenta, las confirmaciones de recepción llegarán con lentitud. Si la red es rápida, las confirmaciones llegarán con rapidez. En un momento dado enviará demasiados paquetes con mucha rapidez, las colas



se acumularán en la red. Cuando las colas estén llenas se perderán uno o más paquetes, el temporizador del emisor TCP expirará. El inicio lento aumenta con demasiada rapidez, después de tres RTT, hay cuatro paquetes en la

un RTT en receptor, es ventana de cuatro paquetes correcto para A medida que recepción, el



red, tardan todo llegar al decir, una congestión de es el tamaño esta conexión. se confirma la inicio lento

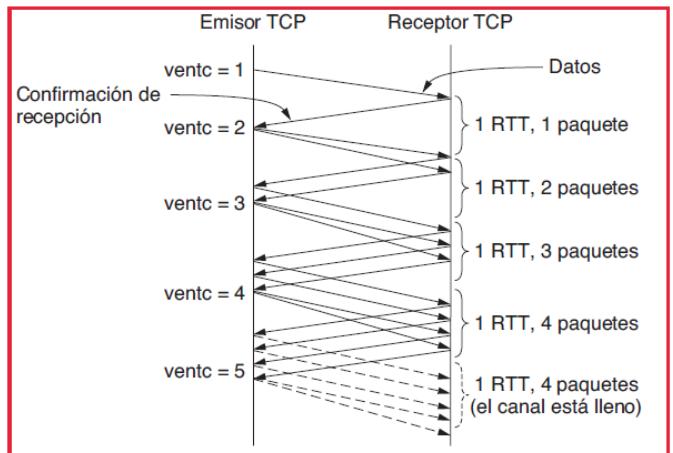
continúa aumentando y llega a ocho paquetes. Sólo cuatro de estos paquetes pueden llegar al receptor en un RTT, sin importar cuántos se envíen. El canal de la red está lleno. Los paquetes adicionales se acumularán en las colas de los enrutadores, ya que no pueden entregarse ocurrirá una congestión y se perderán paquetes.

Para mantener el inicio lento bajo control, el emisor mantiene un umbral para la conexión, conocido como umbral de inicio lento. Se establece en un nivel arbitrariamente alto. TCP continúa incrementando la ventana de congestión en el inicio lento hasta que expira un temporizador. Cada vez que se detecta la pérdida de un paquete, el umbral de inicio lento se establece a la mitad de la ventana de congestión y se reinicia todo el proceso. La ventana actual es demasiado grande, puesto que antes provocó una congestión. Quizá la mitad de la ventana sea una mejor estimación. Cada vez que se atraviesa el umbral de inicio lento, TCP cambia del inicio lento al incremento aditivo. La ventana de congestión se incrementa un segmento por cada tiempo de ida y vuelta, por cada segmento cuya recepción se ha confirmado. La idea general es que una conexión TCP pase mucho tiempo con su ventana de congestión cerca del valor óptimo: no tan pequeño como para que la tasa de transferencia real sea baja y no tan grande como para que ocurra una congestión. La tasa lineal de crecimiento es mucho menor.

Después de perder un paquete, el receptor no puede confirmar la recepción más allá de ese paquete, el número de confirmación de recepción permanecerá fijo y el temporizador se dispare y se retransmita el

paquete perdido. TCP vuelve a empezar con el inicio lento. Hay una forma de que el emisor reconozca que se ha perdido uno de sus paquetes. A medida que llegan al receptor los paquetes subsiguientes al perdido, activan confirmaciones de recepción que regresan al emisor. Estas contienen el mismo número de confirmación de recepción, se denominan confirmaciones de recepción duplicadas. Es probable que haya llegado otro paquete al receptor y que el paquete perdido todavía no aparezca.

Los paquetes pueden tomar distintas trayectorias por la red, pueden llegar fuera de orden. Esto activará confirmaciones de recepción duplicadas, la mayor parte del tiempo esto es poco común en Internet. Los paquetes recibidos no se reordenan demasiado. TCP asume que tres confirmaciones de recepción duplicadas indican que se ha perdido un paquete. La identidad del paquete perdido se puede inferir que es el siguiente en la secuencia. Se puede retransmitir antes de que se dispare el temporizador. Esta se denomina retransmisión rápida. Una vez que se dispara, el umbral de inicio lento sigue establecido a la mitad de la ventana, justo como cuando expira un temporizador. Para volver a comenzar con el inicio lento, hay que establecer la ventana de congestión a un paquete.



Ejemplo, el tamaño máximo de segmento es de 1 KB. En un principio la ventana de congestión era de 64 KB, ocurrió una expiración, el umbral se estableció a 32 KB y la ventana de congestión a 1 KB. La ventana crece en forma exponencial hasta que llega al umbral (32 KB). Se incrementa cada vez que llega una nueva confirmación. Una vez que se traspasa el umbral, crece en forma lineal. El TCP Tahoe resolvió el problema del colapso por congestión. Es posible hacerlo aún mejor. Al momento de la retransmisión rápida, la conexión está operando con una ventana de congestión demasiado grande, pero aún sigue operando con un reloj de confirmación de recepción funcional. Cada vez que llega otra confirmación de recepción duplicada, es probable que otro paquete haya dejado la red. Al usar las confirmaciones de recepción duplicadas para contabilizar los paquetes en la red, es posible dejar que algunos paquetes salgan de la red y continúen enviando uno nuevo para cada confirmación de recepción duplicada adicional.

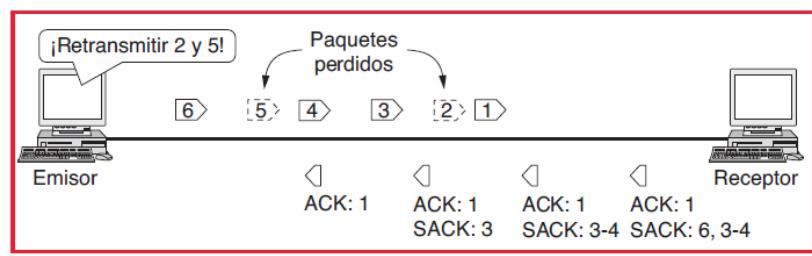
La recuperación rápida implementa este comportamiento, busca mantener el reloj de confirmación de recepción en operación con una ventana de congestión que es el nuevo umbral, se contabilizan las confirmaciones de recepción duplicadas (incluyendo las tres que desencadenaron la retransmisión rápida) hasta que el número de paquetes en la red disminuye a un valor equivalente al nuevo umbral.

El resultado es que TCP evita el inicio lento, excepto cuando la conexión se inicia por primera vez y cuando expira un temporizador. Puede ocurrir cuando se pierde más de un paquete sigue un patrón de diente de sierra. En esencia, es el TCP Tahoe más la recuperación rápida. Linux usa una variante llamada CUBIC TCP, Windows Compound TCP. Hay dos cambios que han afectado las implementaciones de TCP. En primer lugar, la complejidad de inferir mediante un flujo de confirmaciones de recepción duplicadas qué paquetes han llegado y cuáles se han perdido. El número de confirmación de recepción acumulativa no provee esta información. Una corrección simple es el uso de SACK (Confirmaciones de Recepción Selectivas, del inglés Selective ACKnowledgements), que lista hasta tres rangos de bytes que se hayan recibido.

Cuando el emisor y el receptor establecen una conexión, envía la opción SACK permitido de TCP para indicar que comprenden las confirmaciones. Funciona como se muestra en la figura. Un receptor usa el campo Número de confirmación de recepción de TCP de la manera usual. Cuando recibe el paquete 3 fuera de orden, envía una opción SACK para los datos recibidos, junto con la confirmación de recepción acumulativa (duplicada) para el paquete 1. La opción SACK proporciona los rangos de bytes que se han recibido por encima del número proporcionado por la confirmación de recepción acumulativa, se utilizan hasta tres rangos. De cada opción SACK que recibe, el emisor puede decidir qué paquetes retransmitir. SACK es información estrictamente de asesoría. La detección real mediante el uso de confirmaciones

duplicadas y ajustes a la ventana de congestión se lleva a cabo de la misma manera que antes. SACK se implementa ampliamente. Se describe en el RFC 2883 y 3517. El segundo cambio es el ECN (Notificación Explícita de Congestión, del inglés Explicit Congestion Notification), es un mecanismo de la capa de IP para notificar a los hosts sobre la congestión, el receptor TCP puede recibir señales de congestión de IP.

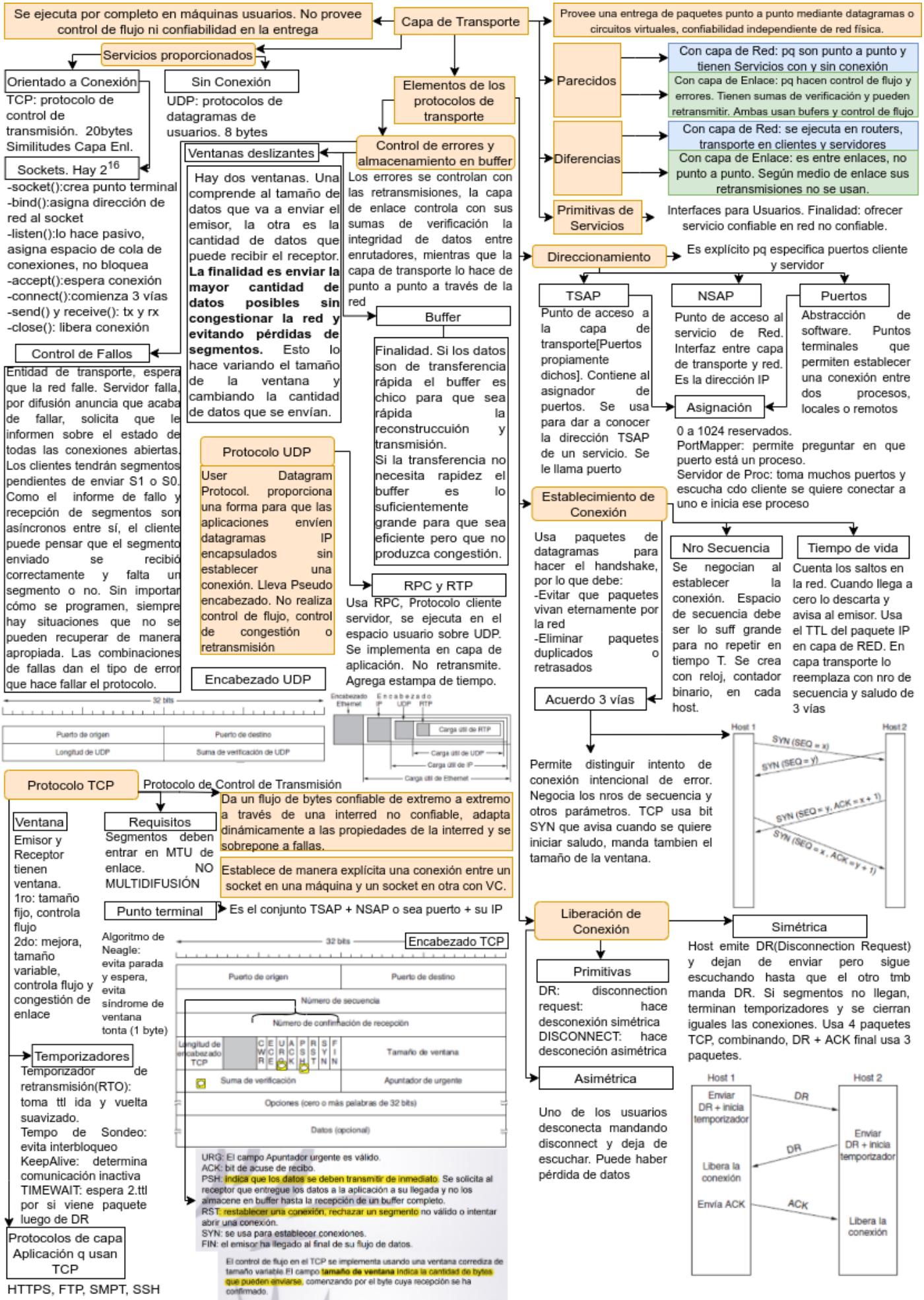
Se habilita para una conexión TCP cuando tanto el emisor como el receptor indican usar ECN y activan los bits ECE y CWR al momento de establecer la conexión. Cada paquete que transporte un segmento TCP se señala con banderas en el encabezado IP para mostrar que puede transportar una señal ECN. Se informa al receptor TCP si cualquier paquete que llegue transporta una señal de congestión de ECN. Luego el receptor utiliza la bandera ECE (ECN Echo) para indicar al emisor TCP que sus paquetes han experimentado una congestión. Para indicar al receptor que escuchó la señal, el emisor usa la bandera CWR (Ventana de Congestión Reducida). El emisor TCP reacciona a estas通知aciones exactamente de la misma forma que reacciona con la pérdida de paquetes mediante confirmaciones duplicadas. Sin embargo, la situación es mucho mejor. Se ha detectado la congestión y ningún paquete resultó dañado. ECN se describe en el RFC 3168. Requiere soporte tanto del host como de los enrutadores, el conjunto completo de control de congestión está en el RFC 5681.



El futuro de TCP

Dos aspectos específicos: El primero es que TCP no proporciona la semántica de transporte que desean todas las aplicaciones. Algunas aplicaciones desean enviar mensajes o registros cuyos límites hay que preservar, otras desean un mejor control sobre las trayectorias de red que utilizan. La aplicación tiene la responsabilidad de lidiar con cualquier problema que TCP no resuelva. Esto ha conducido a propuestas de nuevos protocolos SCTP (Protocolo de Control de Transmisión de Flujo, del inglés Stream Control Transmission Protocol) y SST (Transporte Estructurado de Flujo, del inglés Structured Stream Transport). Sin embargo, cada vez que alguien propone cambiar algo hay una enorme batalla.

El segundo aspecto es el control de la congestión. Se basa en las pérdidas de paquetes como señal de congestión. La tasa de pérdida de paquetes debe disminuir con rapidez con base en el aumento de velocidad. Es muy poco frecuente como para que sirva como una buena señal de congestión puede limitar la tasa de transferencia real. Una posibilidad es usar un control de congestión alternativo, podría ser el tiempo de ida y vuelta, se utiliza en FAST TCP.



Capa de Aplicación: DNS, HTTP

En esta se encuentran las aplicaciones. Todas las capas inferiores dan servicios a esta capa. Usan la capa de transporte para acceder a la red.

También tiene sus protocolos que permiten el funcionamiento de las apps.

DNS: Sistema de Nombres de Dominio

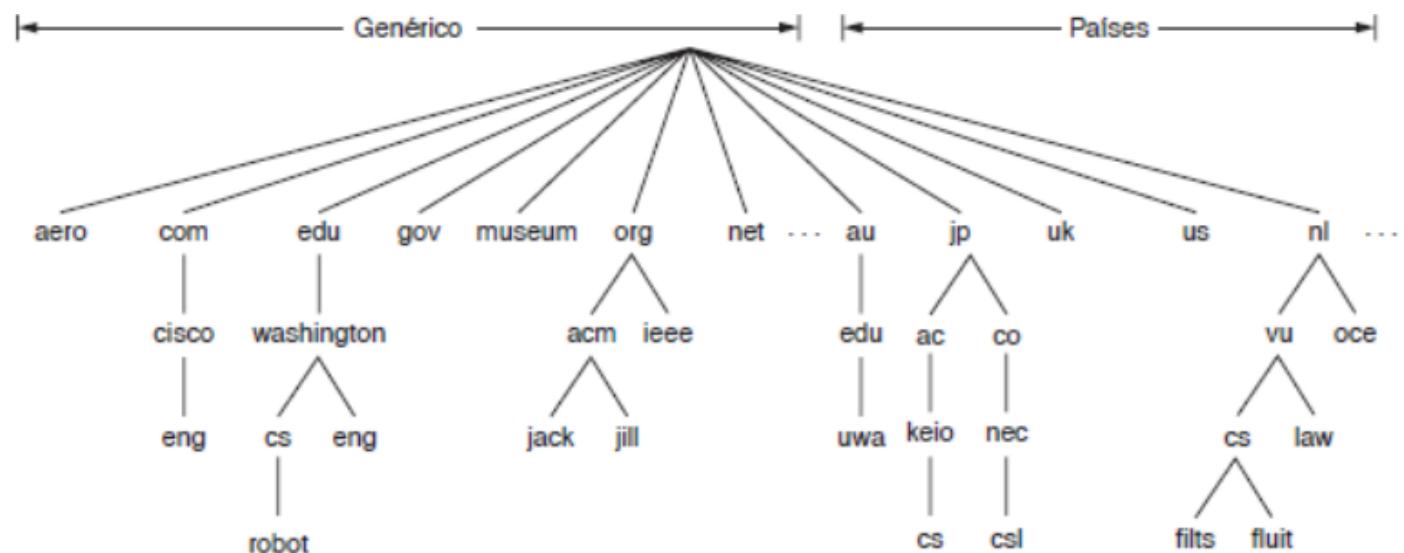
Las direcciones IP son difíciles de recordar, a los servicios web se les da un nombre fácil de recordar y hace falta un mecanismo para convertir estos nombres en las direcciones IP donde se encuentran.

Para esto se inventó el DNS. Asocia los nombres de los hosts con sus direcciones IP. Consiste en un esquema jerárquico de nombres basados en dominios y bases de datos distribuidas por el mundo.

Para asociar un nombre con su IP, un programa llama al procedimiento de biblioteca llamado “Resolvedor” y le pasa el nombre como parámetro. El resovedor envía una consulta por UDP a un servidor DNS y este responde con la IP.

Espacios de Nombres

Los dominios de Nivel Superior (hay 250) abarcan muchos host y subdominios, que a su vez se subdividen y así sucesivamente. Los dominios de nivel superior se dividen en genéricos y de países.



Los genéricos son operados por ICAN. Los de países los operan entidades locales. Las componentes que conforman un nombre de dominio se separan por puntos.

Registros de Recursos

Además del nombre y la IP, el DNS puede tener más datos almacenados para poder acceder al servicio que se pide la IP. Estos son:

Nombre_Dominio Tiempo_De_Vida Clase Tipo Valor

Todo eso se lo manda al Resolvedor que lo pida.

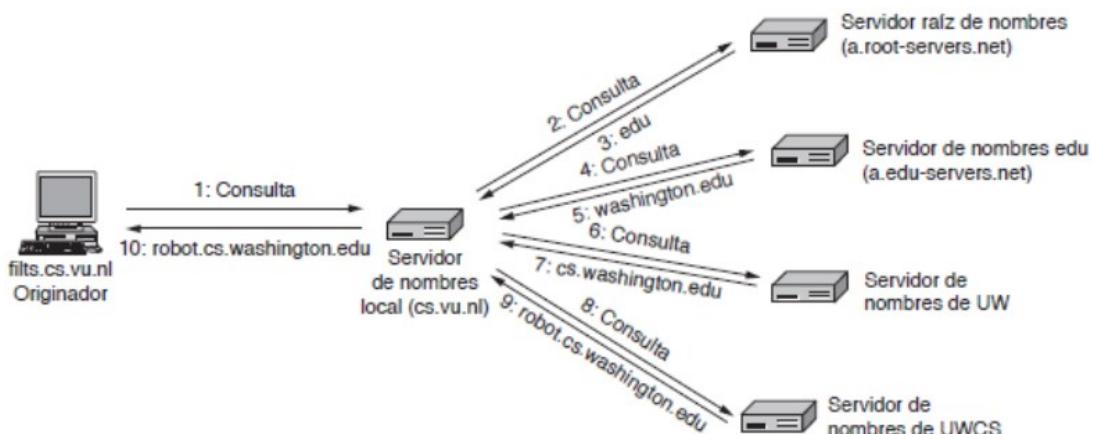
Servicios de Nombres

Un solo servidor para todo el mundo estaría sobrecargado y sería ineficiente por la distancia. Por eso el espacio de nombres se divide en zonas que no se solapan entre partes del árbol. Cada zona se asocia con 1 o más servidores DNS.

- Resolución de Nombres: proceso de buscar la IP de un dominio
- Registro Autorizado: servidor que administra el registro y siempre se toma como legítimo, siempre su información es correcta o verdadera.

Cuando un servidor DNS local no sabe la dirección de un dominio comienza una consulta remota. Hay dos tipos de consultas:

- Recursiva: obliga a un servidor DNS a responder a una solicitud con un error o una respuesta de éxito. Con una consulta recursiva, el servidor DNS debe ponerse en contacto con otros los servidores DNS que necesita para resolver la solicitud. Tipo de consulta típica usado por una resolución de consultas a un servidor DNS y por un servidor DNS consulta su reenviador, que es otro servidor DNS configurado para procesar solicitudes reenviadas a él.
- Iterativa: el servidor DNS responda con la mejor información local que tiene. Un servidor DNS no tiene ninguna información local que puede responder la consulta, simplemente envía una respuesta negativa. Un servidor DNS realiza este tipo de consulta al intentar encontrar nombres fuera de su dominio local



WORLD WIDE WEB

Es una arquitectura para acceder a contenido distribuido en computadoras que son parte de la interred.

Arquitectura

La web consiste de una colección de páginas webs. Cada una puede contener vínculos a otras páginas. Esto se conoce como hipertexto. Las páginas se ven mediante un navegador, este obtiene el código de la página, lo interpreta y la muestra en la pantalla. Los hipervínculos le dicen al navegador que tiene que obtener partes de una página web desde otros servidores.

Navegadores solicitan páginas por TCP, mediante un protocolo de capa de aplicación llamado HTTP.

- Página estática: si es el mismo documento cada vez que se descarga
- Página dinámica: si se genera bajo demanda mediante un programa en el servidor o porque contiene código que se ejecuta en el navegador del cliente. Suele usar cookies.

Cliente

Se deben solucionar 3 aspectos para desplegar una página web:

1. Cómo se llama la página
2. Donde está ubicada
3. Cómo se accede a ella

Solución: A cada página se le da una URL (Localizador Uniforme de Recursos) ⇒ Es el nombre de la pág.
La URL consta de 3 partes: ej: <http://www.cs.washington.edu/index.html>

- Protocolo o esquema
- Nombre DNS del host
- Nombre de la ruta: la ruta sigue el esquema de árbol de archivos de unix, pq la página se suele almacenar en un archivo dentro del servidor al que se conecta.

Protocolos: http es el protocolo nativo de la red para páginas webs, después tenés ftp, mailto (protocolo de correos), rtsp y sip son para establecer flujos continuos de bytes en capa de aplicación.

Pasos del navegador para acceder a página web

1. Determinar URL
2. Si no la tiene en su caché, pide la IP de esa URL al DNS
3. DNS responde con IP
4. Navegador realiza conexión TCP a la IP mediante el puerto 80 (estándar http)
5. Establecida la conexión TCP, manda una solicitud HTTP
6. Servidor envía la página como respuesta
7. Si la página incluye hipervínculos el navegador hace lo mismo con otros servidores para tener todos los recursos que conforman la página
8. Navegador despliega la página
9. Cierra la conexión TCP.

Servidor

1. Acepta conexión TCP entrante
2. Obtiene la ruta de la página, que es el nombre del archivo solicitado donde está la página
3. Obtiene el archivo
4. Envía el archivo
5. Libera la conexión TCP.

Los servidores son multi-hilos. Un hilo front-end acepta las solicitudes de conexión. Crea hilos de procesamiento que atienden dichas solicitudes. Dichos hilos acceden a un caché con los archivos más pedidos, si no, a los HDD más lentos. Ventaja: los hilos bloqueados no afectan a otros clientes.

Problema: capacidad de cómputo limitada. Los hilos se asignan a medida que se liberan conexiones. Estos hilos se dividen en sub-hilos que atienden distintas tareas con el cliente al que están conectados.
Pasos:

1. Resolver el nombre de la página solicitada
2. Control de acceso
3. Verifica caché
4. Obtener del disco o caché la página, si es dinámica se crea para el cliente.
5. Determinar la respuesta
6. Enviar rta al cliente.
7. Registrar la transacción internamente en un registro.

Cookies

Es una cadena de bytes, con nombre único, de pequeño tamaño que el servidor genera para identificar al cliente en subsecuentes conexiones. Se almacena en el navegador del cliente y éste se la envía al servidor junto con sus peticiones para que server sepa quien verga le está hablando. Porque hasta ahora nada de lo que vimos permite identificar usuarios luego de que la conexión se termina.

Páginas estáticas

No cambian con el tiempo, a menos que se modifiquen manualmente. Pueden ser estáticas y aún así contener videos. Se escriben en HTML. Los formularios a llenar siguen siendo contenido estático.

HTML

Lenguaje de Marcado de Hipertexto. Permite escribir páginas webs que incluyen texto, gráficos, video, hipervínculos, etc. Usa marcado, con caracteres y sintaxis especiales se formato del texto y gráficos.

Páginas Dinámicas y Apps Web

Se ejecutan en el navegador, los datos se guardan en servidor. Con protocolos web el navegador muestra la interfaz de usuario y se accede a los datos remotamente. Ventaja: usuarios no necesitan instalar nada, se puede acceder a los datos desde cualquier lado.

Para que las páginas actúen como aplicaciones no pueden ser estáticas. El contenido se puede generar por código que corre en el cliente o en el servidor o ambos.

Generación del lado del Servidor

Hay APIS para que los servidores invoquen programas:

- a) CGI: interfaz de puerta de enlace común. CGI provee una interfaz para permitir que los servidores web se comuniquen con programas de soporte y secuencias de comandos que pueden aceptar entrada (por ejemplo, de los formularios) y generar páginas HTML en respuesta.
- b) PHP: preprocesador de hipertexto. incrusta secuencias de comandos en las páginas hechas con html y hace que las ejecute el mismo servidor para generar la página.

Generación del lado del Cliente

CGI y PHP solucionan el problema de manejar interacciones con bases de datos en el servidor. No pueden responder o interactuar directamente con usuarios. Para esto se usan etiquetas <script> incrustadas en las páginas html que se ejecutan en la máquina del cliente.

El lenguaje más popular para esto es JavaScript. Similar a PHP pero se ejecuta de manera distinta.

PHP manda solicitud al servidor con la interacción del usuario, servidor responde ejecutando el código PHP incrustado en la página y generando una nueva versión que envía como respuesta al cliente.

JavaScript se ejecuta en el cliente y modifica la página directamente. Hace la interacción mucho más rápida.

PHP y java se usan para cosas distintas.

HTTP

Es de solicitud-respuesta. Trabaja sobre TCP. Es de capa de aplicación. Se está volviendo más como un protocolo de transporte con su evolución.

Conexiones

Las primeras versiones no soportaban sesiones y no recordaban usuario. Luego implementaron conexiones persistentes que mantienen el canal TCP abierto para mandar más solicitudes, ahorra conexiones al pedo. Permiten mandar más de una solicitud a la vez. Aprovechan mejor el enlace y recursos, ahorra tiempo.

Métodos

Se diseñó lo más genérico posible para hacerlo future proof. Los métodos son formas diferentes de solicitar cosas. Cada solicitud consiste de una o más líneas de texto ASCII y obtienen respuestas que dan el estado de la solicitud y quizás datos pedidos.

Encabezados de Mensajes

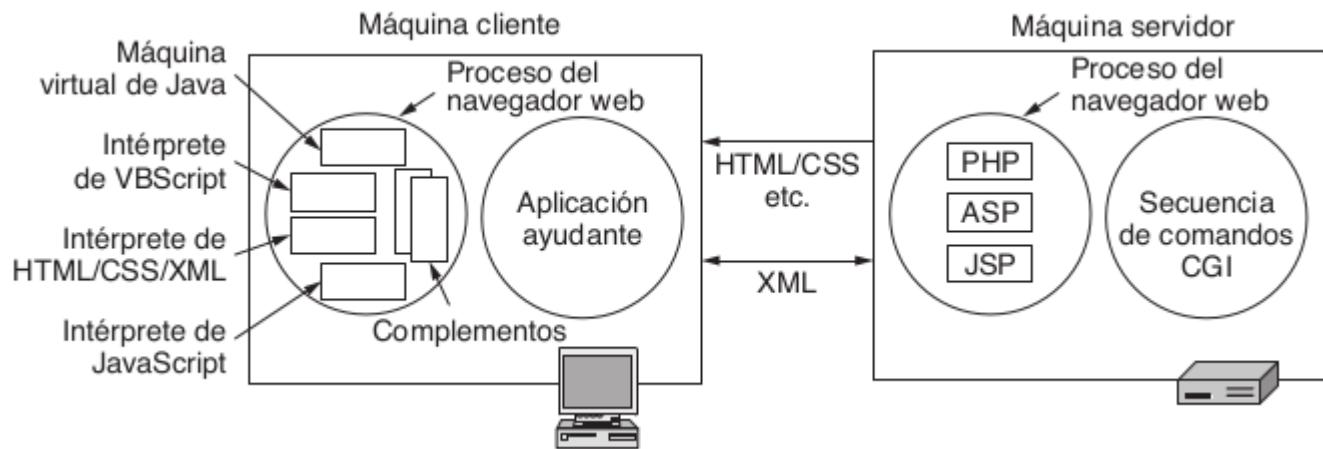
Son líneas adicionales a los métodos que informan al servidor datos extras que le pueden ser de utilidad. Se llaman encabezados de solicitud. Por ejemplo

User-agent: permite al cliente informar al servidor sobre la implementación para que este cree la página correctamente.

Caché

Guarda páginas accedidas con frecuencia en el usuario. ¿Cómo determina que copia mostrar? Mediante:

- Validación de páginas: la página guardada tiene una etiqueta de tiempo que indica cuando expira para que pida una nueva al server.
- Preguntar al server: hace un get condicional, si la guardada en caché es válida la respuesta es corta y rápida, si no, manda la página nueva.



Teoría de Tps de Ana de Redes

Redes TP1: Introducción

Modelos de Capas:

- Modelo OSI: 7 capas que definen las diferentes fases por las que deben pasar los datos para viajar de un dispositivo a otro sobre una red de comunicaciones.
- Modelo TCP/IP: es un modelo de descripción de protocolos de red que se encuentran dentro del conjunto TCP/IP.

Modelo OSI		Modelo TCP/IP
7	Aplicación	Aplicación
6	Presentación	
5	Sesión	
4	Transporte	Transporte
3	Red	Internet
2	Enlace de Datos	Acceso a la red
1	Física	

Acá podemos ver los equivalentes entre capas dentro de los dos modelos.

Donde se colocan los dispositivos dentro de estas capas? Dispositivos a conectar:

- Conmutador o Switch
- Concentrador o HUB
- Puente o Bridge
- Repetidor o Repeater
- NIC
- Router o dispositivo de enrutamiento
- Gateway o pasarela

7	Aplicación	.Gateway
6	Presentación	
5	Sesión	
4	Transporte	
3	Red	.Routers
2	Enlace de Datos	Switch, Bridge, NIC
1	Física	Repetidor, HUB

HUB (concentrador)



Su propósito es regenerar y retemporizar señales de red. Tiene la función de interconectar las computadoras de una red local. Se suele denominar como repetidor multipuerto. Recibe los datos procedentes de una pc y los retransmite a las demás, en ese momento ninguna otra puede enviar una señal. Pueden tener 8, 16, 24... puertos. Se los utiliza para realizar redes de tipo estrella. Si una pc o cable tiene problemas con su conexión las demás seguirán funcionando. SOLO ACTÚAN EN CAPA 1 OSEA EN CAPA FÍSICA.

Se los puede clasificar en dos tipos:

1. Activos: casi todos los modernos. Regeneran electrónicamente las señales de red.
2. Pasivos: solo dividen la señal entre múltiples usuarios y por lo tanto no permiten extender el alcance de un cable.

Switch (Conmutador)



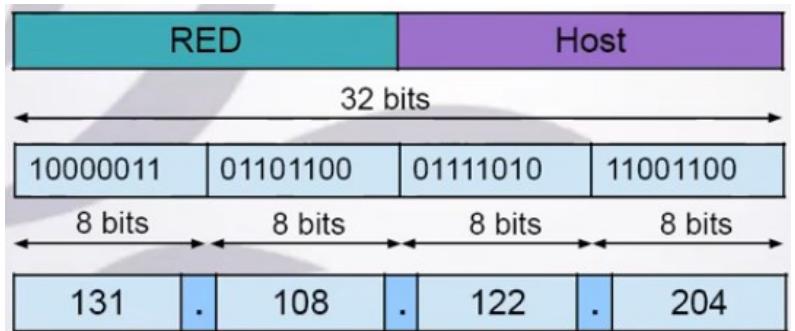
Puente Multipuerto: interconecta segmentos de red, pasando segmentos de uno a otros según la dirección de control de acceso al medio MAC. Concentran la actividad haciendo que la transmisión de datos sea más eficiente.

A diferencia del hub, los switches toman decisiones según las direcciones mac. Están en la capa de enlace de datos del modelo OSI. Se consideran como un HUB "Inteligente". Cuando se enciende, reconoce las mac enviadas que más se envían por cada

puerto, al no mandar toda la información por todos los puertos, ahorra carga en el resto de puertos que no requieren los paquetes. Cuando no tiene armada la tabla de ARP, la cual mantiene una lista de qué direcciones MAC están en cada puerto, se comporta como un hub, retransmitiendo todos los mensajes por todos los puertos. Una vez tiene la lista armada, se comporta como switch, canalizando los paquetes.

Direcciones IP (modelo IP V4)

Son de 32 bits, agrupados de a 8 bits y separados por puntos. Se colocan en forma decimal separados por puntos. Una parte de los bits depende de la dirección de la red y otra de la dirección del host específico dentro de esa red. La dirección del HOST va a depender de algo llamado "Máscara de Subred".



Máscara de Subred

Esta permite distinguir los bits que identifican una red y los que identifican un host. Al realizar una operación lógica **AND** entre la dirección IP y la máscara se obtiene que los bits que identifican la red se ponen en 1 y los que identifican el host en cero.

IP	131	.	108	.	122	.	204	IP = 131.108.122.204 / 24
Máscara	255	.	255	.	255	.	0	RED = 131.108.122.0
RED							Host	Host1 = 131.108.122.1
								Hostn = 131.108.122.254
								Broadcast=131.108.122.255
IP	10000011	01101100	0111 1010	110001100	IP = 131.108.122.204 / 20 ↗			
Máscara	255	255	240	0	RED = 131.108.112.0			
Máscara	11111111	11111111	1111 0000	00000000	Host1 = 131.108.112.1			
					Hostn = 131.108.127.254			
					Broadcast=131.108.127.255			
RED							Host	

Para una misma dirección de red pero con distintas cantidades de 1's en su máscara de subred obtenemos distintas cantidades de direcciones de hosts posibles. Hasta donde tenemos 1 en la máscara es el nombre de red y el resto es el nombre del host dentro de esa red. La dirección de broadcast la obtenemos poniendo todos los bits de host en 1.

Siempre que colocamos una IP a mano también tenemos que colocar la máscara. La dirección MAC de broadcast es de todos unos.

Consulta de redes

¿Qué funcionalidad cumple el broadcast?

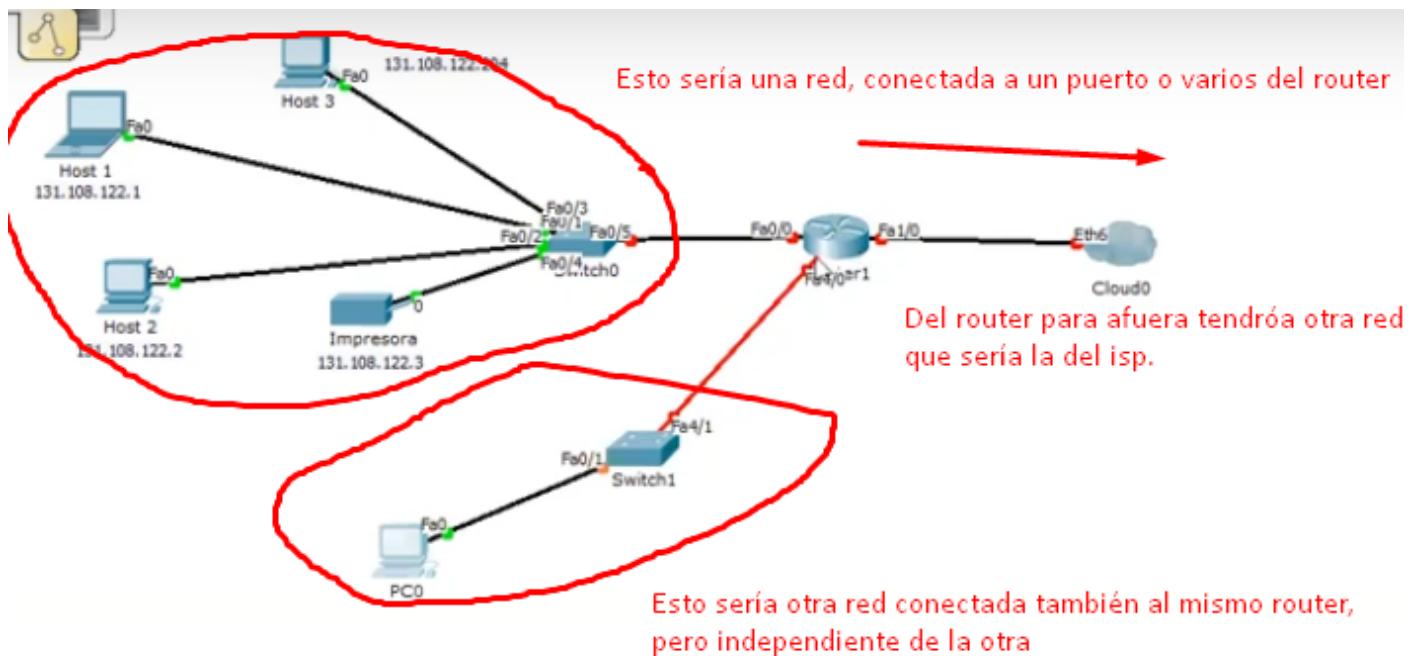
El broadcast se hace cuando se manda un mensaje ICMP, que sabe la dirección de origen y destino, pero el switch no tiene la lista de ARP armada todavía, no sabe que MAC está en cada puerto, entonces el broadcast es un mensaje que se envía a todos los host conectados al switch, el cual solo es contestado por el destinatario que le coincide la dirección IP.

La máscara se hizo pq las direcciones ip están limitadas, entonces los dispositivos que se conectan a las redes tienen su ip variable, esta ip se la da un servidor DHCP cuando se conectan a la red. Esto es así pq la ip de cada host debe pertenecer a una red específica.

Las direcciones ip se dividen en privadas y públicas. Éstas son las que tienen el router de casa y es con la que sale del router al proveedor de internet.

El gateway por defecto también debe pertenecer a la misma red. El gateway es para poder salir de la red. Los únicos que dividen en redes son los routers. El tendrá dos puertos con dos direcciones ip distintas, el puerto WAN que se conecta al proveedor tendrá la ip pública y el otro puerto será el puerto al que conectaremos los host.

Los isp nos dan un modem-router que adentro tiene un switch. La dirección de ip pública nos la da el isp, mientras que las direcciones ip dentro de la red las podemos dar nosotros.



La dirección del gateway por defecto es por la que debo enviar los paquetes afuera de la red. Es decir, la dirección del gateway es la del puerto por el lado de adentro de la red del router.

Redes Tp2: Capa de Enlace

Es la capa 2 del modelo OSI. Su principal tarea es tomar un medio de transmisión y transformarlo en una línea que parezca libre de errores. Funciones:

- Proporcionar una interfaz de servicio bien definida con la capa de red. Considerando:
 1. El medio físico
 2. El control de acceso al medio y
 3. Direccionamiento físico de la información
 4. Manejar los errores de transmisión
 5. Regular flujo de datos para no saturar receptores lentos.
- Por el lado del receptor, convierte el flujo de bits enviado por la capa física en un flujo de tramas para que la capa de red los desencapsule. Métodos de entramado:
 1. Conteo de caracteres
 2. Relleno de bytes
 3. Relleno de bits
 4. Banderas

Los protocolos de la capa de enlace pueden proporcionar control de errores y control del flujo de datos. La IEEE divide la capa de enlace en DOS SUBCAPAS:

Control de enlace lógico (LLC): es la interfaz con las demás capas. Participa en el encapsulamiento. Soporta los "servicios orientados a conexión y los sin conexión".

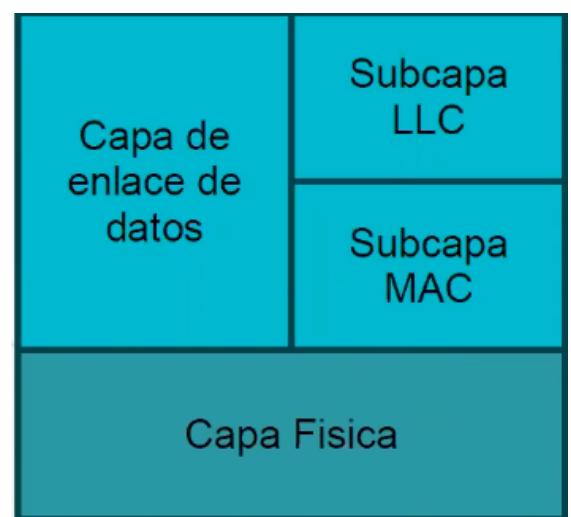
Control de acceso al medio (MAC): se usa para determinar a quien se le deja transmitir por un medio donde hay muchos host queriendo transmitir.

Esta capa utiliza protocolos orientados a hardware como:

1. Ethernet: en estos entornos direcciona por MAC
2. Protocolo Punto a Punto: MPLS
3. Control de enlace de datos de alto nivel (HDLC)
4. Frame relay
5. Modo de transferencia asincrónico (ATM)

Otros protocolos que operan en esta capa son: CSMA/CD y CDP

Los dispositivos en esta capa son: Bridges y Switches



Fragmenta utilizando:

1. Ethernet
2. Ethernet II
3. 802.5 (Token ring)
4. 802.3 y
5. 802.11

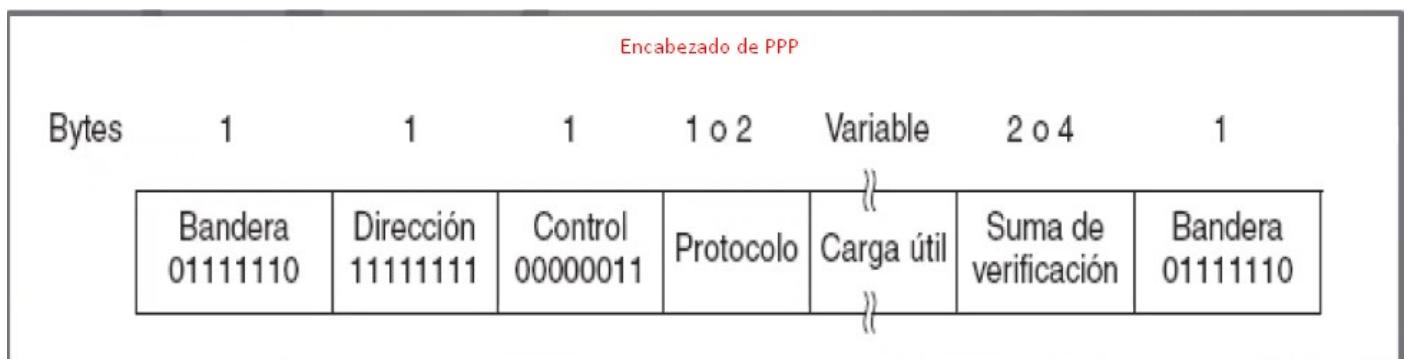
ProtocoloPPP

Protocolo de capa de enlace, usado para establecer comunicaciones directas entre dos nodos. Puede proveer autenticación y encriptación. Soporta detección de errores. Facilita dos funciones importantes:

- Autenticación: generalmente mediante claves de acceso mediante PAP y CHAP
- Asignación dinámica de IP y negociación de la misma en el momento de la conexión.

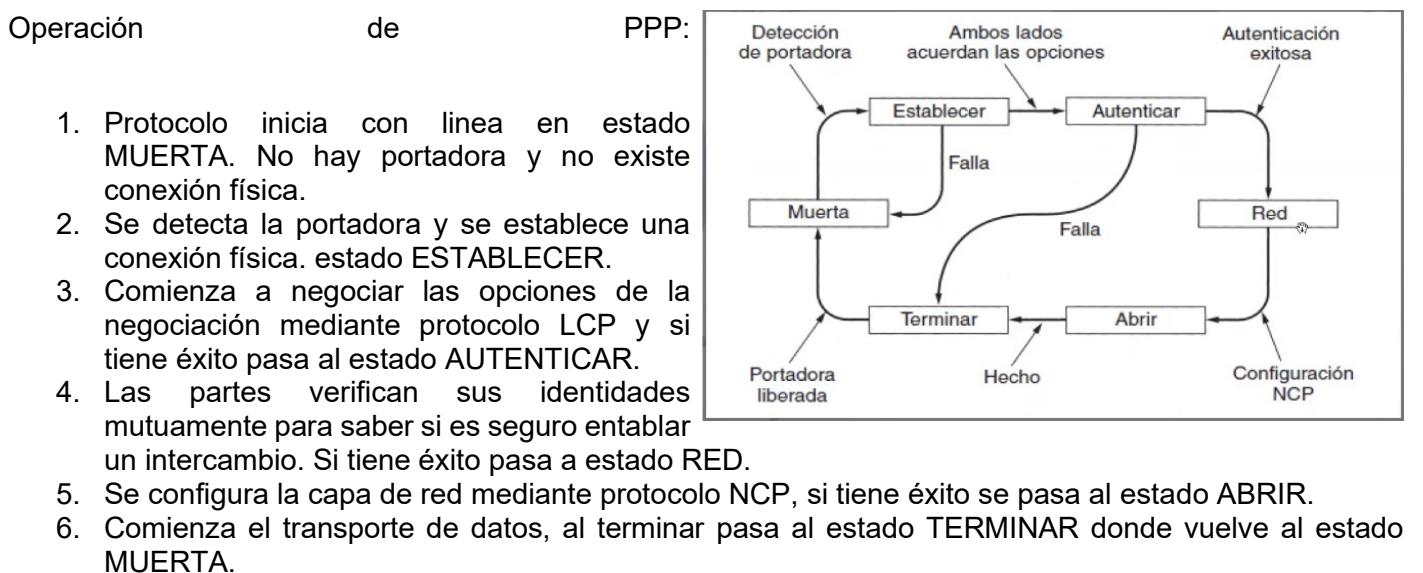
Características importantes que da este protocolo:

1. Entramado sin ambigüedades entre inicio y final de tramas y con sistema de detección de errores incluído.
2. Control de enlace para activar, probar, negociar y desactivar líneas de comunicación. Este protocolo se llama LCP(Link Control Protocol) y permite autenticar, compresión y control de errores. Soporta circuitos síncronos y asíncronos.
3. Un mecanismo para negociar opciones de capa de red con independencia del protocolo. Se tiene un NCP(Network Control Protocol) para cada protocolo de capa de red soportado.



El formato de las tramas está orientado a caracteres o Bytes. Los campos del encabezado son:

- Bandera: marca el inicio y final de la trama
- Dirección: se pone todo en 1 e indica que todas las estaciones deben aceptar la trama. Esto evita asignar las direcciones en la capa de enlace de datos.
- Control: indica el tipo de trama, en este caso es NO NUMERADA.
- Protocolo: Indica cómo está codificado el payload, es decir, en qué protocolo.
- Carga útil: es de longitud variable hasta un máx negociado. Predeterminado es 1500 bytes.
- Suma de verificación: 2 bytes puede negociarse hasta 4 bytes.



Redes Tp3: Capa de enlace Ethernet

Ethernet

- Es una red de difusión basada en bus con control descentralizado.
- Es la más popular de las redes LAN
- Utiliza el control de acceso al medio CSMA/CD
- Ocupa las capas 1 y 2 del modelo OSI

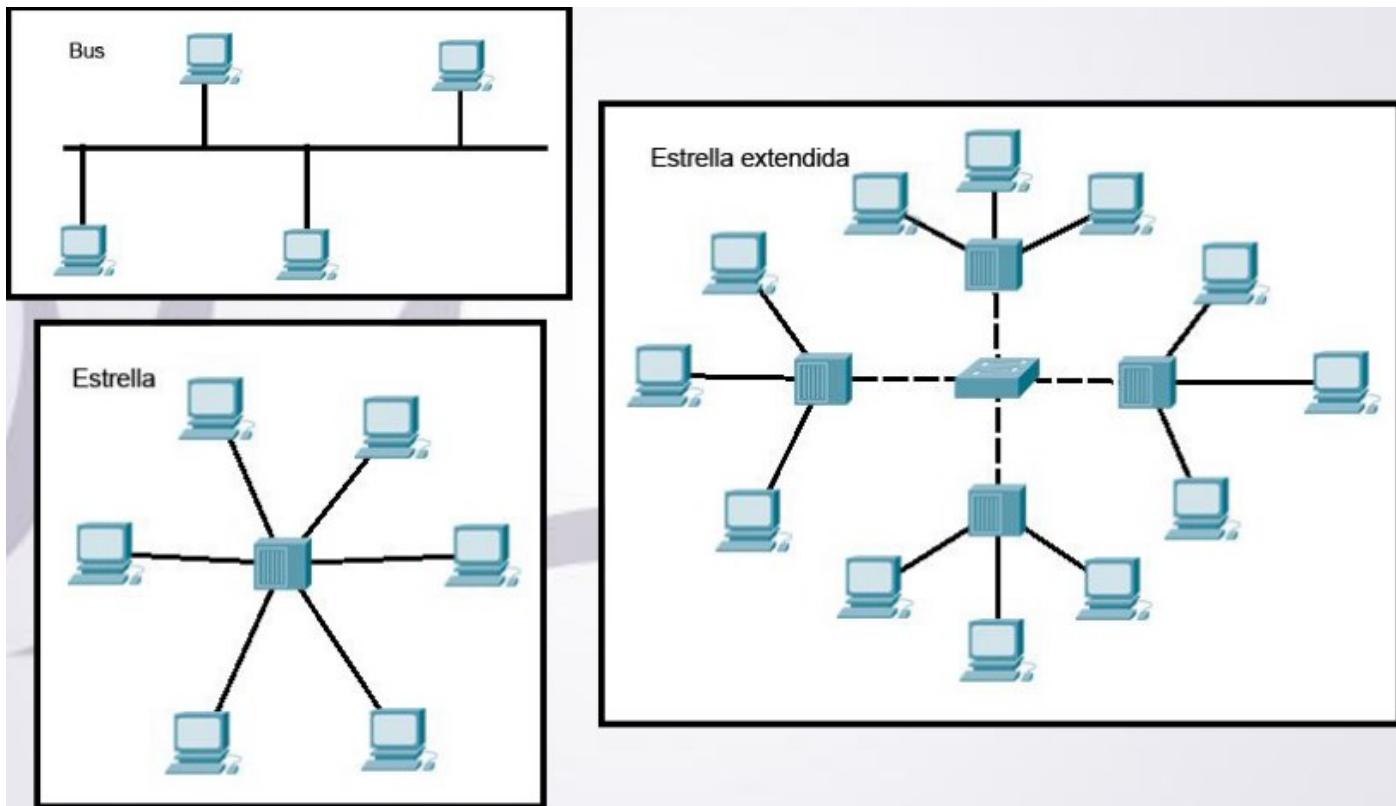
La IEEE tiene varios estándares de LAN, uso de ellos es Ethernet. La 802.3 (Ethernet) y la 802.11 Wifi, tienen distintas capas físicas y diferentes subcapas MAC pero convergen en la misma subcapa de control lógico LLC, con lo cual su interfaz a la capa de red es la misma.

Tipos de Ethernet según el cableado:

Nombre	Cable	Longitud máxima de segmento	Ancho de Banda máximo	Topología Física	Topología lógica
10Base5	Coaxial grueso	500 m	10 Mbps	Bus	Bus
10Base-T	UTP categoría 5	100 m	10 Mbps	Estrella, estrella extendida	Bus
10Base-FL	Fibra óptica multimodo	2000 m	10 Mbps	Estrella	Bus
100Base-TX	UTP categoría 5	100 m	100 Mbps	Estrella	Bus
100Base-FL	Fibra óptica multimodo	2000 m	100 Mbps	Estrella	Bus
1000Base-T	UTP categoría 5	100 m	1000 Mbps	Estrella	Bus

- Número: es la velocidad en Mbps
- Base: es la tecnología de transmisión
- Letras: es el medio físico que se usa para el transporte (Ej: T es par trenzado)

Topologías



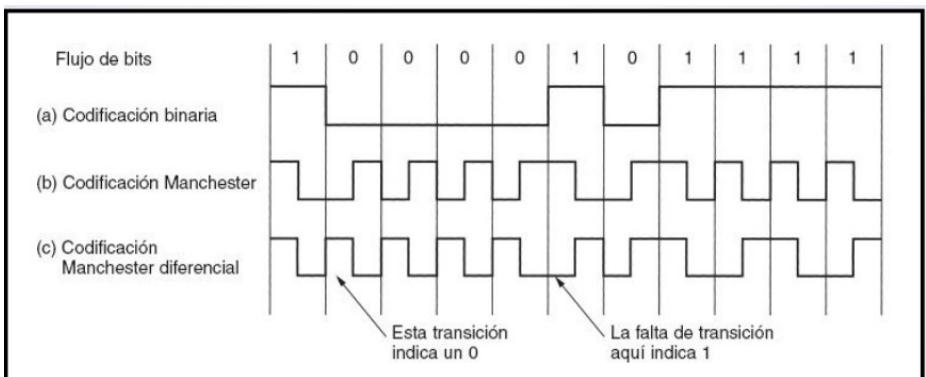
Codificación: usa Manchester y Manchester Diferencial

Ethernet no utiliza codificación binaria directa, las diferencias de velocidades de reloj pueden causar que el receptor y el emisor pierdan la sincronía.

En la codificación Manchester, cada periodo de bit se divide en dos intervalos iguales. Un bit 1 binario se envía teniendo el voltaje alto durante el primer intervalo y bajo durante el segundo. Un 0 binario es justo lo inverso: primero bajo y después alto.

La codificación Manchester diferencial, un bit 1 se indica mediante la ausencia de una transición al inicio del intervalo. Un bit 0 se indica mediante la presencia de una transición al inicio del intervalo. En ambos casos también hay una transición a la mitad.

Una desventaja de la codificación Manchester y Manchester diferencial es que requiere el doble de ancho de banda que la codificación binaria directa, pues los pulsos son de la mitad de ancho.



Trama Ethernet

Trama 802.3

8 bytes	6 bytes	6 bytes	2 bytes	0-1500 b	0-46 b	4 bytes
Preámbulo	Dirección destino	Dirección Origen	Tipo	Datos	Relleno	Suma de verificación

8 bytes	6 bytes	6 bytes	2 bytes	0-1500 b	0-46 b	4 bytes
Preámbulo	Dirección destino	Dirección Origen	Longitud	Datos	Relleno	Suma de verificación

Cambia el byte de tipo por el byte de longitud, el tipo indica cómo interpretar lo que llega, la longitud puede ser menor o iguales a 1500.

Protocolo ARP: Protocolo de Resolución de Direcciones

No se pueden usar las direcciones IP que tienen los host para enviar paquetes porque el hardware de la capa de enlace no entiende las direcciones IP. Entiende de direcciones físicas que son las MAC. Estas son las únicas de cada dispositivo. Las tramas se envían en base a estas MAC y no las IP.

Entonces el protocolo es el responsable de encontrar a qué MAC le corresponde una determinada IP. Es decir, que la dirección Ethernet corresponde a una IP dada.

El protocolo ARP depende de la capa 2 y 3 del modelo OSI.

Funcionamiento:

1. Se envía un paquete (ARP request) a la dirección de difusión de la red (broadcast → MAC = FF FF FF FF FF FF) que contiene la dirección IP por la que se pregunta.
2. Se espera a que esa máquina (u otra) responda (ARP reply) con la dirección Ethernet que le corresponde.

Cada máquina mantiene una tabla con las direcciones traducidas para reducir el retardo y la carga. ARP permite a la dirección de Internet ser independiente de la dirección Ethernet. Para permitir que las direcciones físicas cambien, por ejemplo, cuando una tarjeta Ethernet falla y se reemplaza con una nueva (nueva dirección Ethernet), las entradas en la tabla ARP deben expirar en unos cuantos minutos.

Protocolo RARP: Protocolo de Resolución de dirección de Retorno

Permite que un host que está recientemente iniciado transmita su dirección Ethernet y pregunte su dirección IP. Capa 2 y 3 del modelo OSI

Redes VLAN

Método de crear redes lógicas e independientes dentro de una misma red física.

- Varias VLAN pueden coexistir en un único switch físico.
- Separan en segmentos lógicos una LAN, de modo que otros integrantes de otro segmento no puedan intercambiar datos
- Se configuran por soft, las hace muy flexibles
- Cuando se traslada físicamente un host a otra ubicación, puede permanecer en la misma VLAN sin cambiar la configuración IP de la misma.

Redes Tp4: Capa de Red - Ruteo Estático

7	Aplicación	Es la capa 3 del modelo OSI. Proporciona direccionamiento y selección de ruta.
6	Presentación	Lleva los paquetes de origen a destino. Para esto debe conocer la topología de la subred de comunicación y elegir las rutas adecuadas dentro de ella. Protocolos dentro de la capa 3:
5	Sesión	
4	Transporte	
3	Red	
2	Enlace de Datos	
1	Física	

PDU = Paquete | Dispositivos -> Routers

Los servicios que proporciona a la capa de transporte deben cumplir:

- Ser independientes de la tecnología del enrutador.
- Ser independientes de la cantidad, tipo y tipología de los enrutadores.
- Las direcciones de red deben seguir un plan de numeración uniforme, aun a través de varias LANs y WANs.

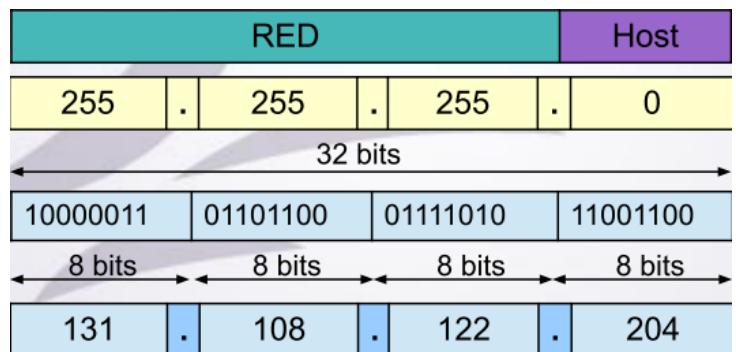
Direcciones IP

El direccionamiento del Protocolo IP usa direcciones IP.

Puerta de enlace predeterminada o Gateway

Una puerta es un dispositivo que sirve como enlace entre dos o más redes informáticas. Es el que conecta y dirige el tráfico de entre de datos entre éstas. En un host debemos configurar la dirección

IP propia del host, la máscara de subred y la puerta de enlace predeterminada (por la que los mensajes salen de la red).



Ruteo

Es la tarea de mandar los paquetes de forma que lleguen a destino. Es necesario que todos los routers conozcan las distintas redes que pueden alcanzar y por donde. Estas son las tareas del router. Se usan protocolos de enrutamiento. Lo que está en el router son tablas(capa 3), si hay algún algoritmo para crear esas tablas se encuentra en capa 7 de aplicación.

Ruteo estático:

El ruteo estático es la forma más sencilla y que menos conocimientos exige para configurar las tablas de ruteo en un dispositivo. Es un método manual en el que se indica explícitamente en cada equipo las redes que puede alcanzar y por qué camino hacerlo.

Ventajas:

- Simple de configurar
- No genera sobrecarga adicional en el router
- Trabajo engorroso
- Se cometen errores
- No hay redundancia, las rutas se deberán modificar manualmente.

Desventajas:

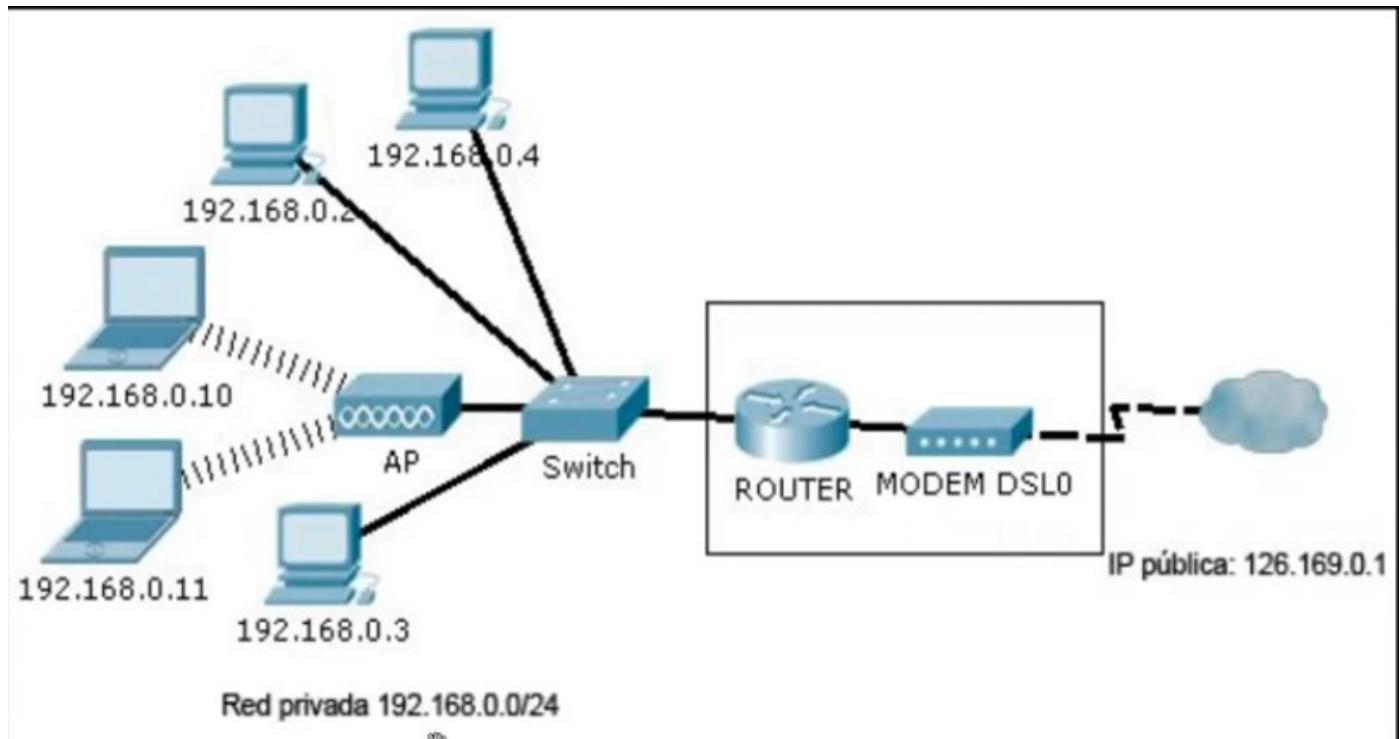
Redes Tp5: Capa de Red y Transporte - NAT y DHCP

NAT (Network Address Translation)

Las direcciones IP están divididas en rangos, públicas y privadas. La idea de nat es asignar una sola dirección IP a una entidad. En la entidad habrán otras pcs con direcciones IP únicas dentro de la red, sin embargo cuando un paquete sale de la red se realiza una traducción de dirección.

NAT se encarga de traducir las ips privadas de una red local a una ip pública. Permitiendo enviar y recibir de internet y luego traduce la IP pública a la del equipo que envió el paquete dentro de la red privada. [también traduce los puertos de la capa de transporte, por esto se dice que nat es una “violación” a las capas, porque trabaja con elementos de capas distintas].

NAT se encuentra entre las capas de red y de transporte del modelo OSI.



Todos los paquetes que los hosts envían al router para salir a internet, saldrán con la ip pública.

Funcionamiento de NAT:

En una red LAN, cada máquina tiene una dirección IP privada única (por ej. 192.168.0.x). Cuando un paquete sale de red LAN, pasa a través de un NAT que convierte la dirección IP interna de origen a una dirección IP pública (por ej. 126.169.0.1). Cuando la respuesta vuelve, se dirige a la IP pública de origen, ¿cómo sabe ahora NAT con qué IP privada se reemplaza? En el encabezado IP no quedan bytes sin usar. La mayoría de los paquetes llevan cargas útiles en capa 4 TCP o UDP. Estos dos protocolos tienen un encabezado que contiene un puerto origen y un puerto destino. El puerto es un número de 16 bits. NAT utiliza este campo.

NAT usa un campo de protocolo de transporte y un campo de direcciones IP, por eso decimos que está entre las capas de transporte y de red, porque ocupa funciones de los protocolos de ambas capas.

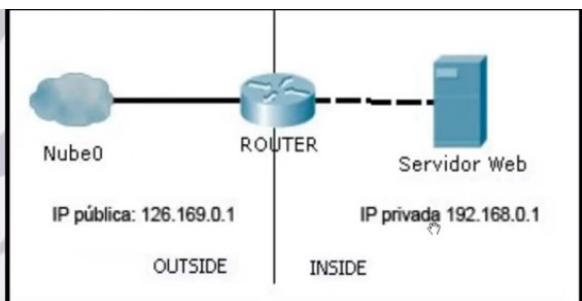
Reemplaza la dirección IP privada por una IP pública y además se toma el valor del puerto, ya sea TCP o UDP y se copia en una tabla de traducción. Cada entrada de la tabla contiene el puerto de origen y la dirección IP privada originales. Finalmente recalcula las sumas de verificación de los encabezados.

Cuando el paquete llega desde afuera, la NAT extrae el campo "puerto de origen" del encabezado TCP o UDP y lo usa como índice en la tabla de traducción. En la tabla encuentra la IP privada y el puerto de origen y los vuelve a insertar en el paquete.

Tipos de NAT:

- NAT estático: o NAT 1:1 la dirección IP privada se traduce a una dirección IP pública que es siempre la misma. Se usa en servidores Web los cuales pueden tener una dirección ip privada dentro del centro de datos pero también estar expuestos a internet.

Config: Se debe configurar por cual interfaz sale al lado público o privado.



NAT estático

Configuración del Router:

- 1) Asociar la dirección IP privada a la pública, comando a utilizar:

Router(config)#ip nat inside source static [IP LOCAL] [IP EXTERNA]

- 2) Indica qué interfaz es la que está conectada a la red interna. Hay que ingresar dentro de la interfaz y e indicarle que es interna

Router(config)#interface [INTERFAZ]

Router(config-if)#ip nat inside

Router(config-if)#exit

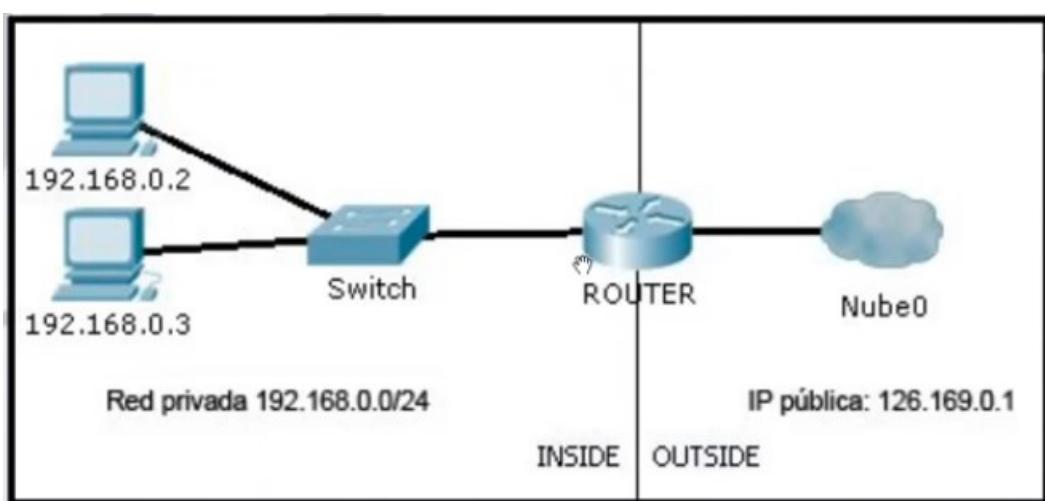
- 3) Indica qué interfaz es la que está conectada a la red externa. Hay que ingresar dentro de la interfaz y e indicarle que es externa

Router(config)#interface [INTERFAZ]

Router(config-if)#ip nat outside

Router(config-if)#exit

- NAT dinámico: se usa cuando tenemos un rango de direcciones IP privadas y una sola IP pública. Usa la tabla de mapeo. El router mantiene una tabla con las direcciones IP registradas y cuando un host nuevo con su ip privada requiera acceso a internet, el router colocará esa ip privada en la tabla.



Config:

NAT dinámico

Configuración del Router:

1) Definimos la lista de direcciones IP externas que va a tener el router para asignarle a los paquetes salientes:

**Router(config)#ip nat pool [NOMBRE DE LISTA DE IPS] [PRIMERA IP]
[ULTIMA IP] netmask [MASCARA DE RED]**

2) Configurar la lista de acceso para que sepa el rango de direcciones a las que le tiene que aplicar NAT:

Router(config)#access-list [NUMERO DE LISTA DE ACCESO] permit [IP DE RED] [WILDCARD]

3) Decirle a NAT con qué lista de acceso va a controlar las IPs que tiene que convertir:

Router(config)#ip nat inside source list [NUMERO DE LISTA DE ACCESO] pool [NOMBRE DE LISTA DE IPS] overload

4) Indica qué interfaz es la que está conectada a la red interna. Hay que ingresar dentro de la interfaz y e indicarle que es interna

Router(config)#interface [INTERFAZ]

Router(config-if)#ip nat inside

Router(config-if)#exit

5) Indica qué interfaz es la que está conectada a la red externa. Hay que ingresar dentro de la interfaz y e indicarle que es externa

Router(config)#interface [INTERFAZ]

Router(config-if)#ip nat outside

Router(config-if)#exit

1) La lista de direcciones externas define las direcciones IP públicas que puede tener ese router para asignar a los paquetes salientes.

2) La lista de acceso configura el rango de direcciones a las que se le tiene que aplicar NAT. Luego en lugar de colocar la máscara de red (donde dice Wildcard) se coloca la máscara de red invertida, porque al router le queda más cómodo para hacer los cálculos nada más.

3) Luego le decimos a NAT con que lista va controlar las direcciones de IP. Se enlazan las listas que colocamos en los pasos 1 y 2.

4) Quedan por configurar las interfaces públicas y privadas.

DHCP (Dynamic Host Configuration Protocol)

Es un protocolo que se encuentra en la capa de aplicación del modelo OSI, porque es una aplicación. Es un protocolo tipo cliente servidor que permite a los clientes de una red obtener los parámetros de configuración de la misma automáticamente. El server posee una lista con direcciones privadas de ip dinámicas las cuales las va asignando a los clientes conforme éstas van quedando libres. 3 métodos de asignación:

1. Manual o estática: asigna una dirección ip fija a una máquina determinada dentro de la red.
2. Automática: asigna una ip de manera permanente mientras el cliente esté conectado. Se usa cuando el número de clientes no varía demasiado. Cada vez que el cliente se vuelve a conectar le asigna una ip nueva.
3. Dinámica: permite la reutilización dinámica de las direcciones IP disponibles. El administrador determina un rango de direcciones IP y cada dispositivo conectado a la red está configurado para solicitar su dirección IP al server cuando la interfaz de red se inicializa.

El protocolo DHCP puede proveer un montón de configuraciones:

Un servidor DHCP puede proveer de una configuración opcional al dispositivo cliente (definidas en RFC 2132)

- Dirección del servidor DNS
- Nombre DNS
- Puerta de enlace de la dirección IP
- Dirección de broadcast
- Máscara de subred
- Tiempo máximo de espera del ARP
- MTU (Unidad de Transferencia Máxima) para la interfaz
- Servidores NIS (Servicio de Información de Red)
- Dominios NIS
- Servidores NTP (Protocolo de Tiempo de Red)
- Servidor SMTP
- Servidor TFTP
- Nombre del servidor WINS

1) Indicar rango de IP excluido del pool (conjunto)

Router(config)#ip dhcp excluded-address [IP inicial] [IP final]

2) Asignar un nombre al conjunto de direcciones que serán asignadas.

Router(config)#ip dhcp pool [nombre]

3) Definir los parámetros.

Router(dhcp-config)#network [IP RED] [MASCARA DE RED]

Router(dhcp-config)#default-router [IP routes]

Router(dhcp-config)#dns-server [IP servidor DNS]

Para mostrar tabla de asignación:

Router#show ip dhcp binding



No es necesario que se configure todo. Alcanza con que se configure:

- Dirección IP con su máscara
- La puerta de enlace, para poder salir de la red
- Si queremos traducir URL en IP necesitamos saber la dirección del server DNS

Config de DHCP:

1) Las ip excluidas se guardan para direccionar routers o máquinas con IP fija dentro de la red.

2) Asigna al conjunto de direcciones que vamos a permitir configurar un nombre.

3) Definimos parámetros: primero la dirección IP de la red y su máscara de red, luego la dirección del gateway y finalmente la del DNS.

Config en router hogareño:

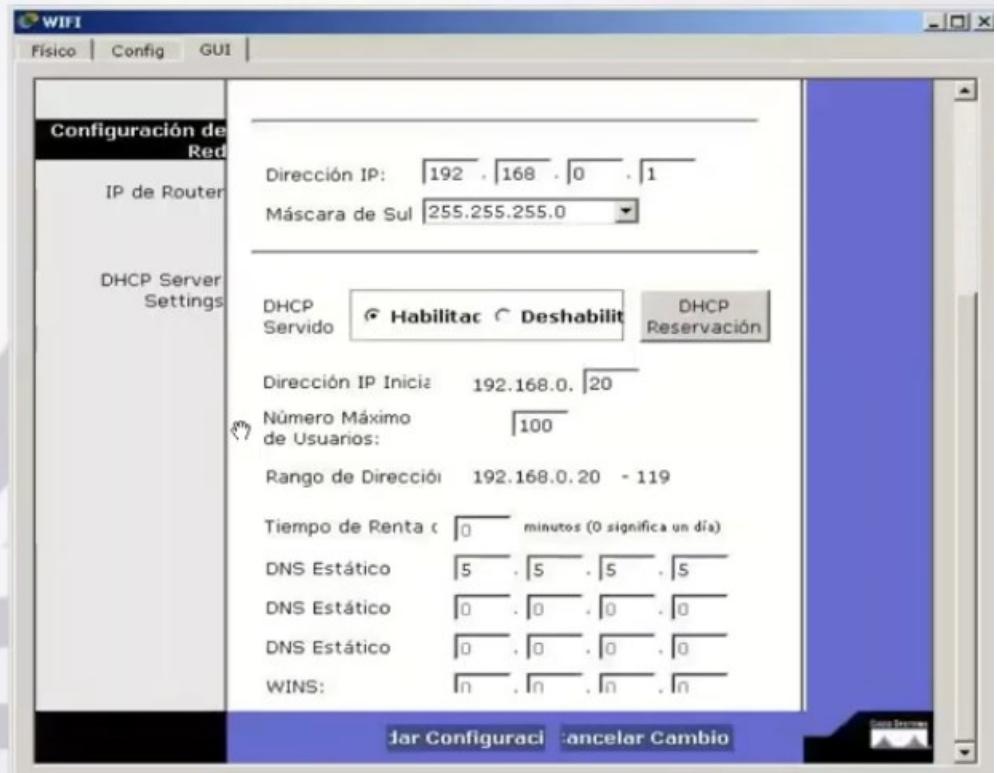
Tenemos varias partes que configurar:

- Una es el puerto WAN o Ethernet que será el puerto que está en otro color, a ese puerto le decimos que tome direcciones por DHCP.
- En puerto inalámbrico que es el puerto de WIFI le decimos cuál es su dirección IP y máscara.
- El puerto lan es el switch interno.

Por último puedo configurar al router como servidor DHCP.

Configuración servidor DHCP en router Linksys

- 1) Habilitar servidor DHCP
- 2) Guardar configuración
- 3) Verificar la dirección IP y máscara del router
- 4) Colocar la primer dirección IP a ser asignada
- 5) Colocar el número máximo de usuarios
- 6) Colocar servidor DNS
- 7) Guardar configuración



Al colocar que empiece con la dirección 192.168.20 se reserva las primeras 19 direcciones para direccionamiento estático, porque al configurar el puerto Ethernet como DHCP le asigne la dirección 192.168.0.1.

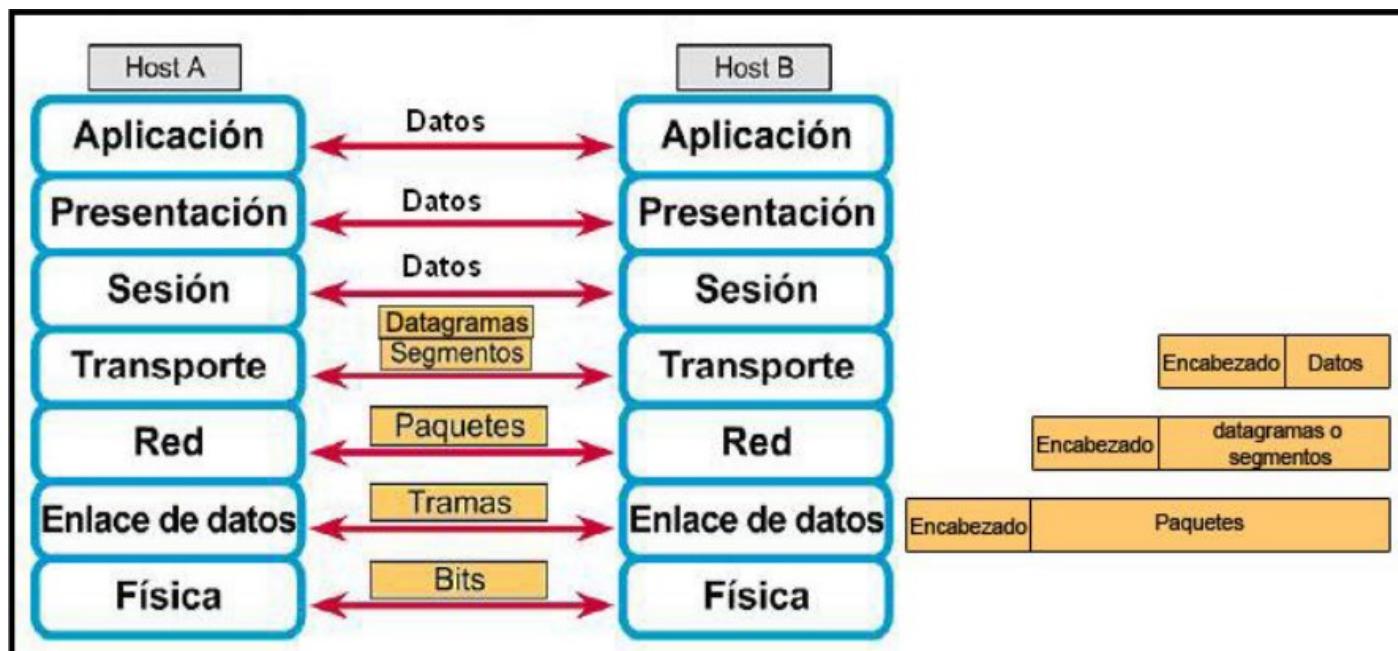
Luego le puedo colocar el server dns para que envíe esa información al cliente que se quiere conectar.

Redes Tp6: Capa de Transporte - UDP y TCP

Capa de transporte (primer video):

Función básica: aceptar datos enviados por capas superiores, fragmentarlos si es necesario y pasarlo a la capa de red. Su servicio de transporte de datos aísla a las capas superiores de la implementación del transporte. Multiplexa comunicación entre diferentes apps entre hosts mediante puertos lógicos.

Modelo OSI		Modelo TCP/IP	
7	Aplicación	Aplicación	Se diseña para que pares de nodos [origen - destino] lleven a cabo una comunicación. Se asemeja al modelo OSI.
6	Presentación		
5	Sesión		
4	Transporte		Recibe el flujo de datos desde una aplicación y lo divide en segmentos o datagramas según el protocolo usado.
3	Red		
2	Enlace de Datos		
1	Física		



Tenemos dos hosts A y B. Se quiere establecer comunicación B → A. Los datos del modelo híbrido que corresponden a Aplicación, Presentación y Sesión, son los que se quieren mandar de B hacia A. Se pasan a la capa de transporte y esta los envuelve agregando un encabezado. Dicho conjunto se llama datagrama o segmento según el protocolo. Se pasa a la capa de red y se agrega encabezado, se llama paquete. Se pasa a capa de enlace de datos y se agrega otro encabezado, se llama trama, se pasa a la capa física y se transmite.

La aplicación y cada capa ven como se comunican con las otras del otro host, no ven las capas inferiores.

El que recibe va sacando de cada conjunto el encabezado hasta obtener los datos y pasarlo a la aplicación destino.

Puertos:

Se utilizan para hacer direccionamiento en capa de transporte. Permiten la comunicación entre aplicaciones. Puerto es un campo de 16bits.

- Puerto 0: reservado pero permitido. Se usa como puerto de origen cuando el emisor no espera recibir respuestas.
- Puertos 1 a 1023: son puertos "bien conocidos" y tienen usos estandarizados.
- Puertos 1024 a 49151: pueden ser usados por cualquier aplicación del usuario
- Puertos restantes: son puertos efímeros, se asignan aleatoriamente para comunicaciones temporales entre clientes y servidores.

Ejemplos de puertos bien conocidos:

Aplicación	FTP	Telnet	SMTP	DNS	TFTP	HTTP	POP3	Telnet: acceso remoto
Puerto	20/21	23	25	53	69	80	110	Los puertos son independientes del protocolo de transporte usado, son estándares de la capa.
Protocolo	TCP / UDP							

Protocolos:

- TCP: orientado a conexión. SEGMENTOS.
- UDP: no orientado a conexión. DATAGRAMAS.

Todo lo que definimos como función de la capa de transporte lo tienen que cumplir todos sus protocolos. Como UDP no tiene confirmación de entrega ni control de flujo, la capa no hace esas tareas, sino que son funciones específicas del protocolo TCP.

Protocolo UDP:

Protocolo de nivel de transporte basado en intercambio de datagramas. No necesita conexión previa ya que el datagrama lleva la información necesaria para el direccionamiento. Solo añade mux de apps y verificación de cabecera y carga útil. Todo lo que respecta a confiabilidad lo tiene que implementar capa superior.

Desventajas

Ventajas

Sin confirmación de entrega ni control de flujo.

No introduce retardos. Permite soportar más clientes a un servidor que con TCP pq tiene menos requerimientos. Bajo costo de transmisión, solo 8 bytes.

Encabezado UDP

Puerto origen (2 bytes)	Puerto destino (2 bytes)
Longitud total (2 bytes)	Suma de verificación (2 bytes)
Datos (longitud variable)	

Protocolo TCP:

Orientado a conexión, fiable a nivel de capa de transporte y proporciona un flujo de bytes confiable de extremo a extremo mediante una interred no conciable.

Este protocolo garantiza que los datos se entregan sin errores y en el orden correcto.

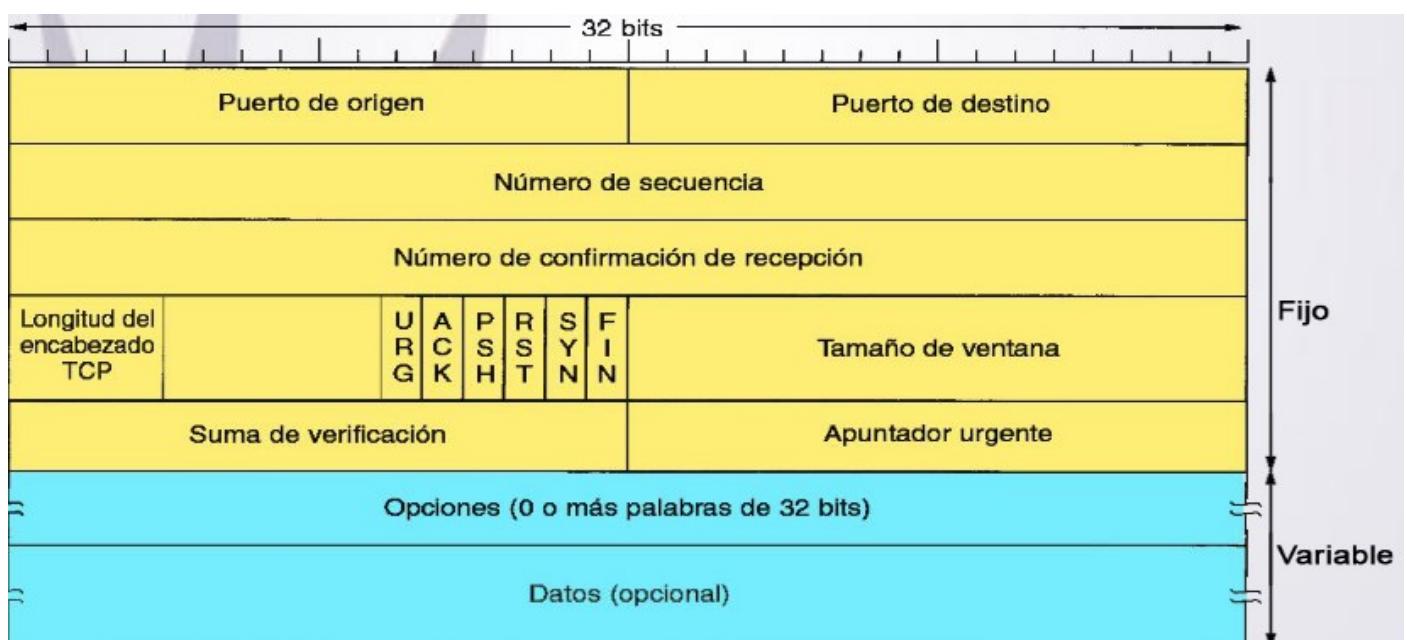
Entidad TCP -> proceso local que acepta flujos de datos de usuarios o procesos locales, los fragmenta, encapsula, y envía como paquetes IP independientes. Las entidades Tx y Rx intercambian datos como segmentos.

TCP soporta otros protocolos de capa de aplicación populares como HTTP, SMTP, SSH y FTP. Mediante el concepto de puertos multiplexa la salida de aplicaciones.

Como se obtiene un servicio TCP? Se obtiene mediante la creación de sockets en cliente y servidor, cada socket tiene una dirección IP y un puerto asignados. Se establece de manera explícita una conexión entre estos sockets. Full duplex, punto a punto. Sin multicast.

Segmento TCP:

El segmento es [ENCABEZADO FIJO + OPCIONES + DATOS]



La longitud de encabezado indica la cantidad de palabras de 32 bits que hay en el encabezado.

Banderas de 1 bit:

- URG: El campo Apuntador urgente es válido.
- ACK: bit de acuse de recibo.
- PSH: indica que los datos se deben transmitir de inmediato. Se solicita al receptor que entregue los datos a la aplicación a su llegada y no los almacene en buffer hasta la recepción de un buffer completo.
- RST: restablecer una conexión, rechazar un segmento no válido o intentar abrir una conexión.
- SYN: se usa para establecer conexiones. FIN: el emisor ha llegado al final de su flujo de datos.

Puertos origen y destino: puntos terminales locales de la conexión.

Número de secuencia: posición de los datos del segmento en el flujo total transmitido.

Nro confirmación: valor del segmento siguiente que espera el receptor.

Control de flujo: se implementa mediante una ventana corrediza de tamaño variable indicado por el campo "tamaño de ventana" en el encabezado. Tamaño de ventana 0 indica que no puede recibir más datos por el momento. Período de envío lo hace mandando el nro de conformación nuevo con un tamaño de ventana distinto de cero.

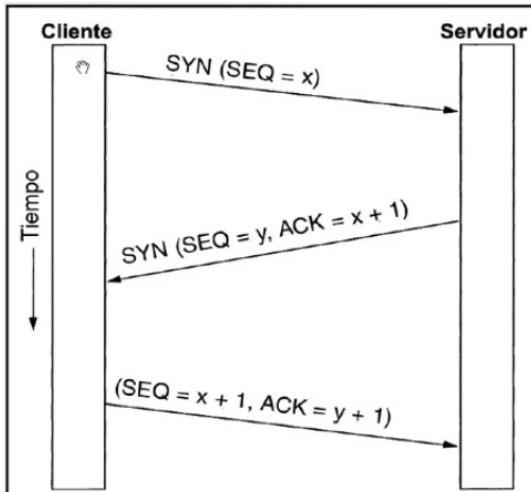
Suma de verificación: para comprobar errores en cabecera y datos.

Puntero urgente: indica la posición en bytes donde están los datos urgentes.

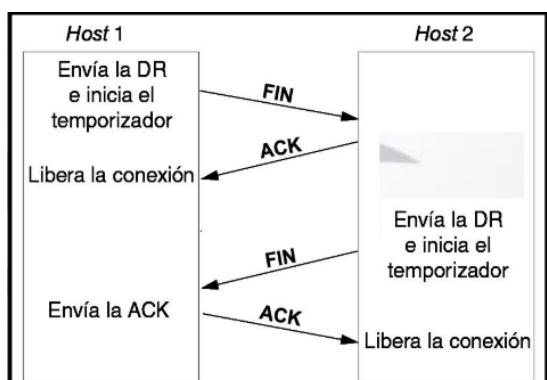
Opciones: permite agregar cosas

Campo relleno: asegura que la cabecera tenga un tamaño múltiplo de 32 bits.

Establecimiento de conexión -> saludo de 3 vías.



Cliente inicia la conexión con parámetro $SYN = 1$ y nro de seq = X, el servidor contesta con su $SYN = 1$, su secuencia propia = y más la secuencia recibida más 1 (que sería la próxima secuencia que espera enviar el cliente). Luego la conexión queda establecida cuando el cliente contesta con los ACK, nros de secuencia correspondientes y los datos que desea enviar.



Liberación de conexión -> método de 4 pasos.

Las conexiones terminan independientemente de ambos lados. Quien desea terminar envía segmento con bit $FIN = 1$, el otro contesta con ACK. Así pasa en ambas direcciones.

Requiere 4 mensajes. Puede ser 3 mensajes y evita el problema de los dos ejercicios con temporizadores

Diferencias principales entre TCP y UDP

UDP

- Sin conexión
- Sin confirmación
- Transporte no fiable de datagramas
- Sin retardos (por no hacer conexión)
- Más liviano para un servidor = más clientes activos en paralelo
- No retransmite

TCP

- Orientado a conexión
- Con confirmación
- Transporte fiable de segmentos
- Confidabilidad agrega costo en datos transmitidos, carga del servidor y retardo en tiempo
- Retransmite si no llega un segmento

FTP: Protocolo de Transferencia de Archivos

Protocolo para transferir archivos entre sistemas mediante una red TCP. El servicio FTP lo ofrece la capa de aplicación usando TCP de capa de transporte. Usa puertos 20 y 21

Comandos de FTP

Comando	Acción que realiza
open servidor	Inicia una conexión con un servidor FTP.
close o disconnect	Finaliza una conexión FTP sin cerrar el programa cliente.
quit	Finaliza una conexión FTP y la sesión de trabajo con el programa cliente.
delete archivo	Borra un archivo en el servidor.
put archivo	Envía un archivo al directorio activo del servidor.
rename archivo	Cambia el nombre a un archivo en el servidor.
get archivo	Obtiene un archivo.
dir	Muestra el contenido del directorio en el que estamos en el servidor.
mkdir	Crea el directorio indicado de forma remota.