Arquitectura de Redes y Servicios

Tema 8: El protocolo ICMP (v1.1)

Dr. Diego R. Llanos
Dpto. Informática, Universidad de Valladolid
diego@infor.uva.es

1. Introducción

Necesidad de un protocolo de control

- Los protocolos TCP e IP permiten la comunicación entre dos máquinas conectada a la red... cuando la red funciona correctamente.
- Sin embargo, si hay un problema de encaminamiento en un router, o si un nodo no funciona por cualquier motivo, la comunicación simplemente no se produce.
- El protocolo ICMP (*Internet Control Message Protocol*) permite diagnosticar gran parte de estos problemas, y además optimizar el funcionamiento de la red.

2. ICMP en la pila de protocolos

Lugar de ICMP en la pila de protocolos

- ICMP es un protocolo de control, y por lo tanto no encaja exactamente en las funcionalidades de la pila de protocolos OSI.
- Los *datagramas* ICMP pertenecen formalmente a la capa de transporte, ya que para enviarlos hay que añadir una cabecera de la capa IP que codifique el origen y el destino.
    - Al crear el socket con `socket()`, hay que indicar el tipo como `SOCK_RAW`, y el "protocol" como `IPPROTO_ICMP`.

- Se envían con `sendto()`.

- Sin embargo, cuando se recibe un datagrama ICMP, la capa IP no retira la cabecera antes de entregarlo en destino (tenerlo en cuenta!).

Lugar de ICMP en la pila de protocolos

- Otra diferencia importante con TCP es que la comunicación no sigue el modelo cliente-servidor.
- Cualquier aplicación (que cuente con permisos para ello) puede enviar un datagrama ICMP, pero su recepción corre por cuenta de la implementación de la capa IP de la máquina destino.
- En otras palabras, el destino no es una aplicación de las capas superiores que actúa como "servidor", sino la propia pila TCP/IP.
- Por cierto, si intentamos enviar un paquete ICMP a nuestra propia IP, nuestra capa TCP/IP no lo procesa, tan sólo nos lo devuelve.

Formato de los datagramas ICMP

- El formato de los datagramas ICMP es muy sencillo:

1 byte	1 byte	2 bytes
Type	*Code*	*Checksum*
"Payload" del datagrama ICMP (tam. variable, mult. de 4)		

- Los campos *Type* y *Code* sirven como códigos de operación.
- El *Payload*, de tamaño variable, depende del tipo de mensaje ICMP que se esté enviando.
- El campo *Checksum* sirve para garantizar la integridad del datagrama ICMP. (La capa IP comprueba la integridad de su cabecera, no de la del resto del datagrama.)

Cálculo del checksum

- Para poder mandar un datagrama ICMP, además de rellenarlo, tenemos que calcular correctamente su checksum (o la capa IP de destino lo descartará).
- El checksum se define como "*el complemento a uno sobre 16 bits de la suma en complemento a uno de todas las palabras de 16 bits a partir del campo Type, inclusive.*"
- Para calcular este checksum, el campo de checksum debe valer cero.

Refrescando conceptos: el complemento a 1

- El complemento a 1 es un sistema de numeración binario con signo en donde:
  - Los números positivos se representan en binario natural.
  - Los números negativos se representan como la negación lógica de la representación en binario natural del número positivo correspondiente.
- La suma en complemento a uno de dos números de X bits se hace sumando bit a bit (como en binario natural), pero si se produce un acarreo a partir del bit X+1, ese acarreo hay que acumularlo al resultado.
- Ejemplo: suma en complemento a uno a 16 bits:

$$F321_{16} + 1000_{16} = \mathbf{1}\ 0321_{16} \rightarrow 0322_{16}$$

Cálculo del checksum en C (primera parte)

1. Se inicializa a 0 los dos bytes del campo de checksum en el datagrama.
2. Se define una variable entera **numShorts**, inicializándola al tamaño en bytes del datagrama, dividido entre dos (esto da el tamaño en "half words").
3. Se define un **puntero** a `unsigned short int` (para recorrer los elementos de 16 bits del datagrama ICMP).
4. Se define un **acumulador** de tipo `unsigned int` = 0, (variable de 32 bits sin signo para ir acumulando los resultados parciales).
5. Se inicializa el puntero al inicio del datagrama ICMP.
6. Para i desde 0 hasta (numShorts-1):

- acumulador = acumulador + (unsigned int) *puntero
- puntero++ (como puntero apunta a elementos de 16 bits, este incremento lo deja apuntando al siguiente elemento)

Cálculo del checksum en C (segunda parte)

- Ya tenemos la suma de todos los elementos de 16 bits del datagrama en una variable que tiene 32 bits.
- Para que la suma sea correcta a 16 bits en complemento a uno, tenemos que sumar la parte alta del acumulador a la parte baja:

```
acum = (acum >> 16) + (acum & 0x0000ffff)
```

- Esto hay que hacerlo dos veces (¿por qué?)
- Una vez hecho, simplemente hay que hacer un *not* del resultado, para obtener su complemento a 1 (esto se hace con `acum = ~acum`), y guardar los 16 bits de menos peso en el campo de checksum.

¿Hemos calculado bien el checksum?

- Para comprobarlo, podemos volver a calcularlo tras guardar el valor del checksum, y el resultado tiene que dar cero.
- Ejemplo: si la suma de todos los elementos nos dio 100, el algoritmo de checksum dice que el checksum ha de valer -100.
- En consecuencia, si colocamos el -100 en el campo de checksum y lo volvemos a calcular, el checksum ha de valer cero.

Algunos usos de los mensajes ICMP

- **Redirect** (type 5, code variable) Permite que un router le *sugiera* a un host que use otro router distinto para el envío de paquetes a un destino concreto.
- **Time exceeded** (type 11, code variable) Se usa para que un router le indique al origen de un datagrama que éste se ha descartado porque su TTL ha alcanzado el valor cero.
- **Source Quench** (type 4, code 0). Se usaba para que los routers le pidieran a los hosts que enviaran los datos más despacio. No era un mecanismo justo para el control de la congestión: se abandonó en 2012.

Algunos usos de los mensajes ICMP

- **Timestamp** (type 13 solicitud, 14 respuesta) Se usa para sincronizar dos hosts. Se envía el tiempo local en milisegundos, y el host de destino envía a su vez el suyo. Los respectivos tiempos no cambian, pero de este modo se conoce el *offset*.
- **Address mask** (type 17 solicitud, 18 respuesta) Se usa para que un host pida la máscara de subred asociada a un host. Es una forma de obtención remota de información que puede usarse para atacar una red: los routers ya no suelen responder.
- **Destination unreachable** (type 3, code variable) Se usa para informar al origen que el paquete no ha podido entregarse.
- **Ping** (type 8 solicitud, 0 o 3 respuesta) Se usa para ver si un host remoto está funcionando. Será el objeto de nuestra práctica.

Algunos usos de los mensajes ICMP

- **Ping** (type 8 solicitud, 0 o 3 respuesta) Se usa para ver si un host remoto está funcionando. Será el objeto de nuestra práctica.
- Objetivo: desarrollo de un cliente de tipo *ping*, que muestre si la IP de destino se alcanzó correctamente, y si no es así, que indique el motivo, procesando los códigos de error.