Protocolo TFTP

Matías Roqueta

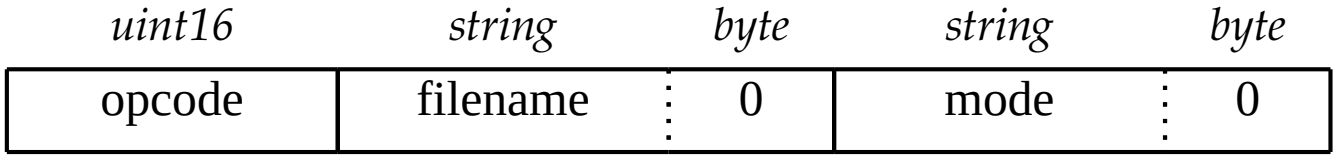
Ingeniería en Telecomunicaciones, Instituto Balseiro

Tramas TFTP

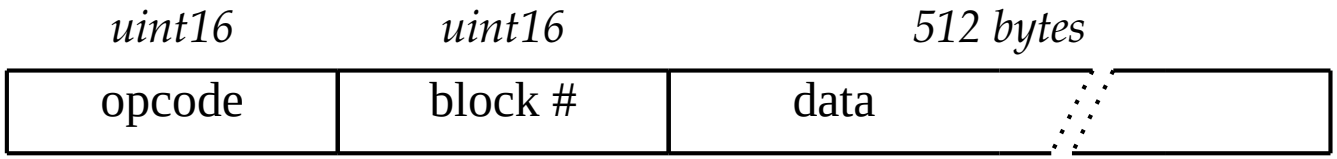
- TFTP es un protocolo simple de transmisión de archivos sobre el protocolo de internet UDP.
- Existen 5 tipos de trama en el protocolo TFTP.
- Cada trama es identificada por un *header* que contiene mínimamente su código de operación (opcode).

opcode	Operación	Descripción
1	RRQ	Read request
2	WRQ	Write request
3	DATA	Data
4	ACK	Acknowledgement
5	ERROR	Error

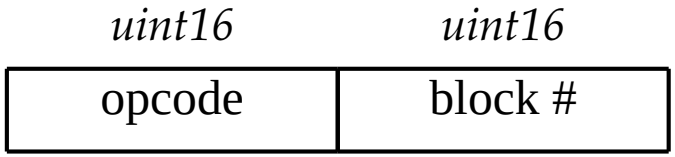
Tramas RRQ y WRQ



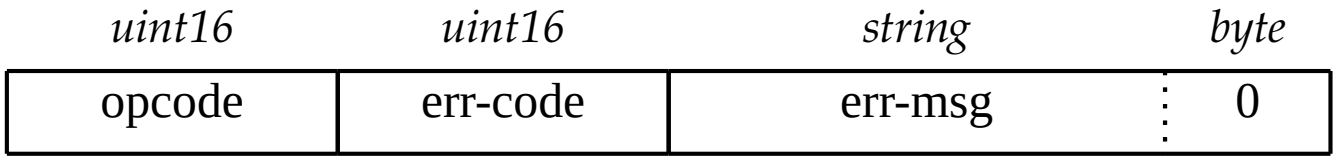
Trama DATA



Trama ACK



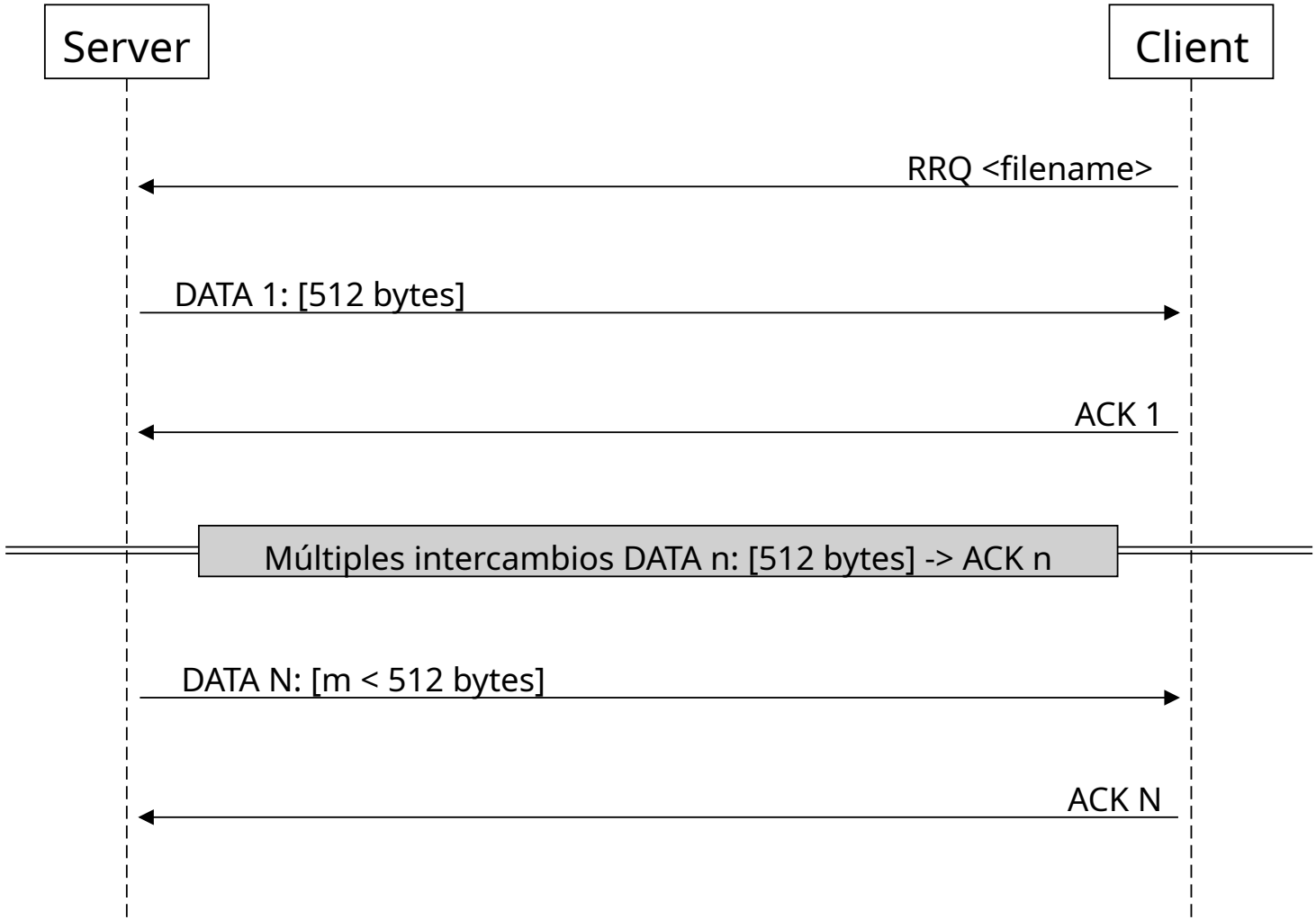
Trama ERROR



Transferencia de Archivos

- La transferencia de un archivo se inicia con el envío de una trama RRQ o WRQ del cliente al servidor.
- El archivo es transmitido en tramas DATA consecutivas con un *payload* de 512 bytes.
- Cada bloque se responde con una trama ACK antes de que se envíe el siguiente bloque.
- La transmisión de una trama DATA con *payload* menor a 512 bytes indica el fin de la transmisión del archivo.
- Un error es informado por una trama ERROR, en presencia de un error se interrumpe la transmisión.

Procedimiento Lectura



Procedimiento Escritura

Sockets UDP

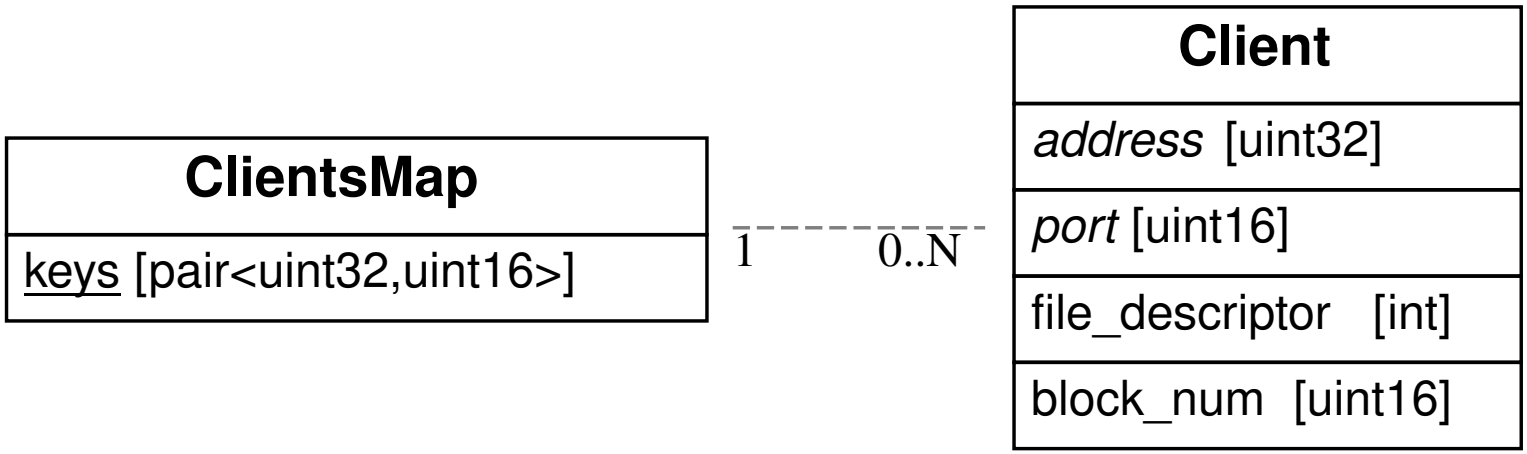
Al ser UDP, el servidor no establece conexión con los clientes, sin embargo necesita identificarlos para responder al cliente correcto.

Se implementa una función que identificará a un cliente por su dirección IP y puerto de origen luego de recibir un datagrama:

```
void clientInfo(const sockaddr_in& clientAddr,
                uint32_t &ip, uint16_t &port) {
    ip   = ntohl(clientAddr.sin_addr.s_addr);
    port = ntohs(clientAddr.sin_port);
    return;
}
```

Múltiples Clientes

El servidor necesita atender a múltiples clientes con un único socket UDP. Para esto se mantiene un registro de los clientes con transferencias activas.



El directorio de clientes se implementa con una estructura `std::unsorted_map`, en donde las claves son tipo `std::pair` con la dirección de origen y el puerto de origen.

Implementación Tramas

Las tramas se implementan como `struct` con los campos correspondientes, por ejemplo:

```
struct DATAPacket {
    struct Hdr {
        uint16_t opcode;
        uint16_t block_num;
    } hdr;
    char data[512];
    DATAPacket(uint16_t block_num) {
        hdr.opcode = htons(OP_DATA);
        hdr.block_num = htons(block_num);
        memset(this->data, 0, sizeof(this->data));
    }
};
```

Implementación Tramas

Las tramas se transmitirán y recibirán en formato (`void*`). Se implementa una función para identificar el opcode de una trama recibida.

Esto se consigue casteando el buffer a formato `uint16_t` y retornando el primer elemento.

```
uint16_t getOpcode(const void* buffer) {
    return ntohs(((uint16_t*) buffer)[0]);
}
```

En función del opcode el receptor podrá castear la trama al formato correcto usando un `switch`.

Lectura y Escritura

Leer de un archivo consiste en copiar los datos de un `file descriptor` al campo `data` de una `struct DATAPacket` en un `offset` indicado por el campo `block_num`.

Escribir a un archivo consiste en copiar los datos del campo `data` de una `struct DATAPacket` a un `file descriptor` en un `offset` indicado por el campo `block_num`.

Ejemplo lectura:

```
int readFromFile(int file_fd, DATAPacket &pk) {
    int offs = blockSize*(ntohs(pk.hdr.block_num)-1);
    ssize_t bytesRead = pread(file_fd, pk.data,
                              sizeof(pk.data), offs);
    return bytesRead;
}
```

Implementación Servidor

Consiste en un `while(true)` en el cual se reciben mensajes por un socket y luego se llama al ciclo principal del servidor:

```
void handleMessage(args...) {
    uint16_t opcode = getOpcode(buffer);
    uint32_t clientIP;
    uint16_t clientPort;
    clientInfo(clientAddr, clientIP, clientPort);
    switch (opcode) {
        case OP_WRQ:   // handle write request...
        case OP_RRQ:   // handle read request...
        case OP_DATA:  // handle received data...
        case OP_ACK:   // handle acknowledgement...
        case OP_ERROR: // handle received error...
        default:       // handle invalid opcode...
    }
}
```

Handle write request

- Se extrae el `filename` de la solicitud y se abre un nuevo archivo en modo `WRITE ONLY`. Si hay error al abrir el archivo se envía un `ERRORPacket` al cliente.
- Si no hay error, se registra una nueva entrada en el directorio de clientes.
- Se envía un `ACKPacket` al cliente con `block_num=0`.

Handle read request

- Se extrae el `filename` de la solicitud y se abre un nuevo archivo en modo `READ ONLY`. Si hay error al abrir el archivo se envía un `ERRORPacket` al cliente.
- Si no hay error, se registra una nueva entrada en el directorio de clientes.
- Se inicializa un nuevo `DATAPacket` con `block_num=1`, se lee del `file descriptor` al `DATAPacket` y este se transmite.
- Si el número de bytes leidos es menor a 512, se cierra el `file descriptor` y se elimina el cliente del directorio.

Handle received data

- Se busca el cliente en el directorio para obtener el `file descriptor` de su archivo.
- Se escribe del `DATAPacket` recibido al `file_descriptor` obtenido del registro de clientes.
- Se responde con el `ACKPacket` de este `block_num`, y luego se incrementa el `block_num` del cliente registrado.
- Si el número de bytes escritos es menor a 512, se cierra el `file descriptor` y se elimina el cliente del directorio.

Handle acknowledgeemnt

- Se busca el cliente en el directorio para obtener el `file descriptor` de su archivo. Si el cliente no está en el directorio se ignora la trama.
- Se incrementa el `block_num` y inicializa un nuevo `DATAPacket` con el nuevo `block_num`, se lee del `file descriptor` al `DATAPacket` y este se transmite.
- Si el número de bytes leidos es menor a 512, se cierra el `file descriptor` y se elimina el cliente del directorio.

Test realizado

Se realiza una prueba para validar que el servidor implementado pueda atender a múltiples clientes ejecutando el cliente implementado en modos lectura y escritura simultáneamente:

```
int main() {
    const char* SERVER_IP = "127.0.0.1";
    pid_t pid = fork();
    if (pid == 0) {
        execl("./client", "client", SERVER_IP,
              "r_example.txt", "write", nullptr);
    } else if (pid > 0) {
        execl("./client", "client", SERVER_IP,
              "w_example.txt", "read", nullptr);
    }
    return 0;
}
```