Protocolo TFTP

Matías Roqueta

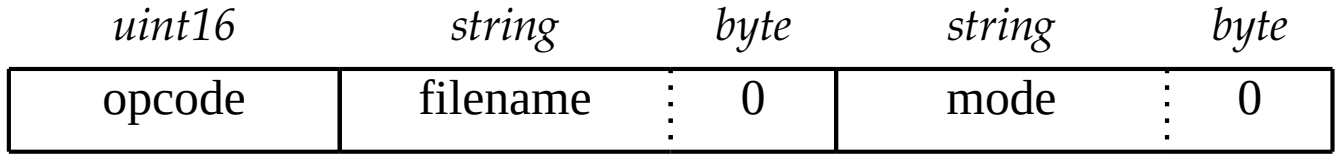
Ingeniería en Telecomunicaciones, Instituto Balseiro

Tramas TFTP

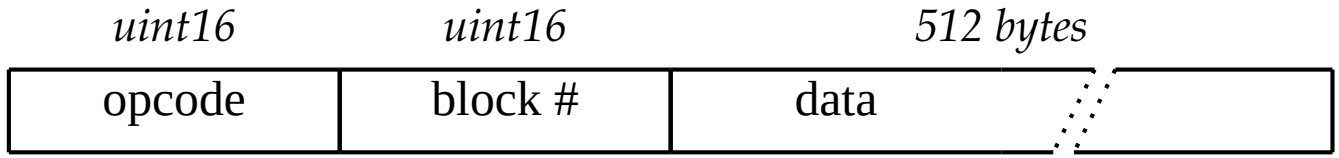
- TFTP es un protocolo simple de transmisión de archivos sobre el protocolo de internet UDP.
- Existen 5 tipos de trama en el protocolo TFTP.
- Cada trama es identificada por un *header* que contiene mínimamente su código de operación (opcode).

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

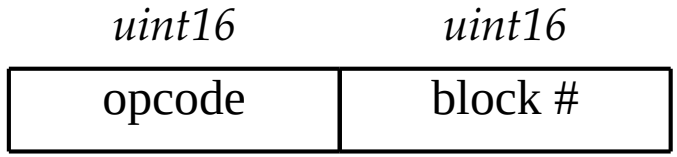
Tramas RRQ y WRQ



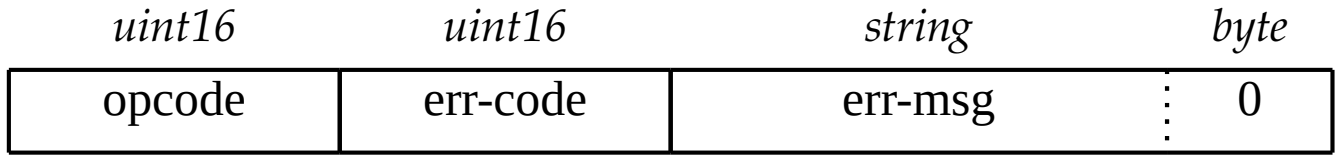
Trama DATA



Trama ACK



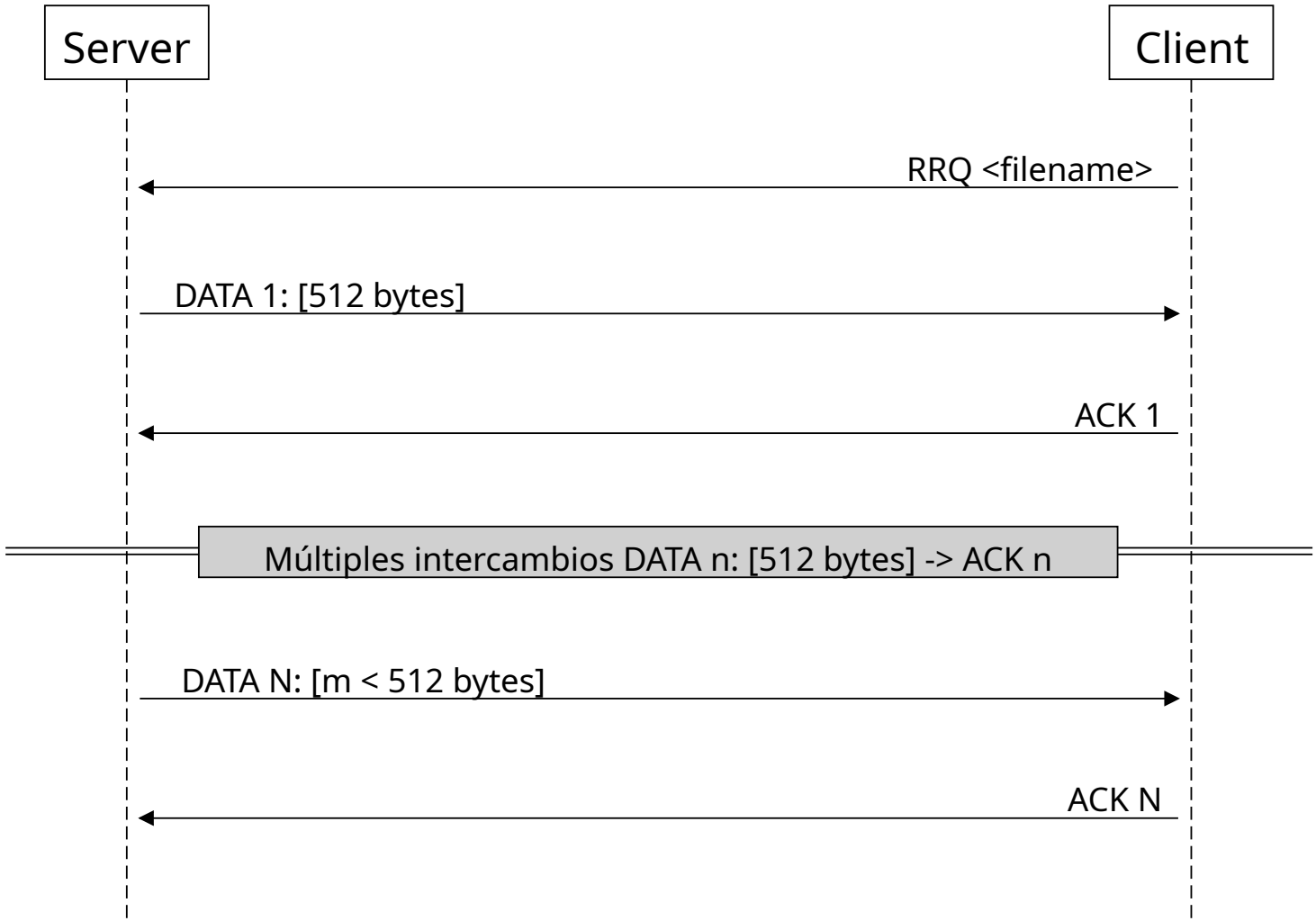
Trama ERROR



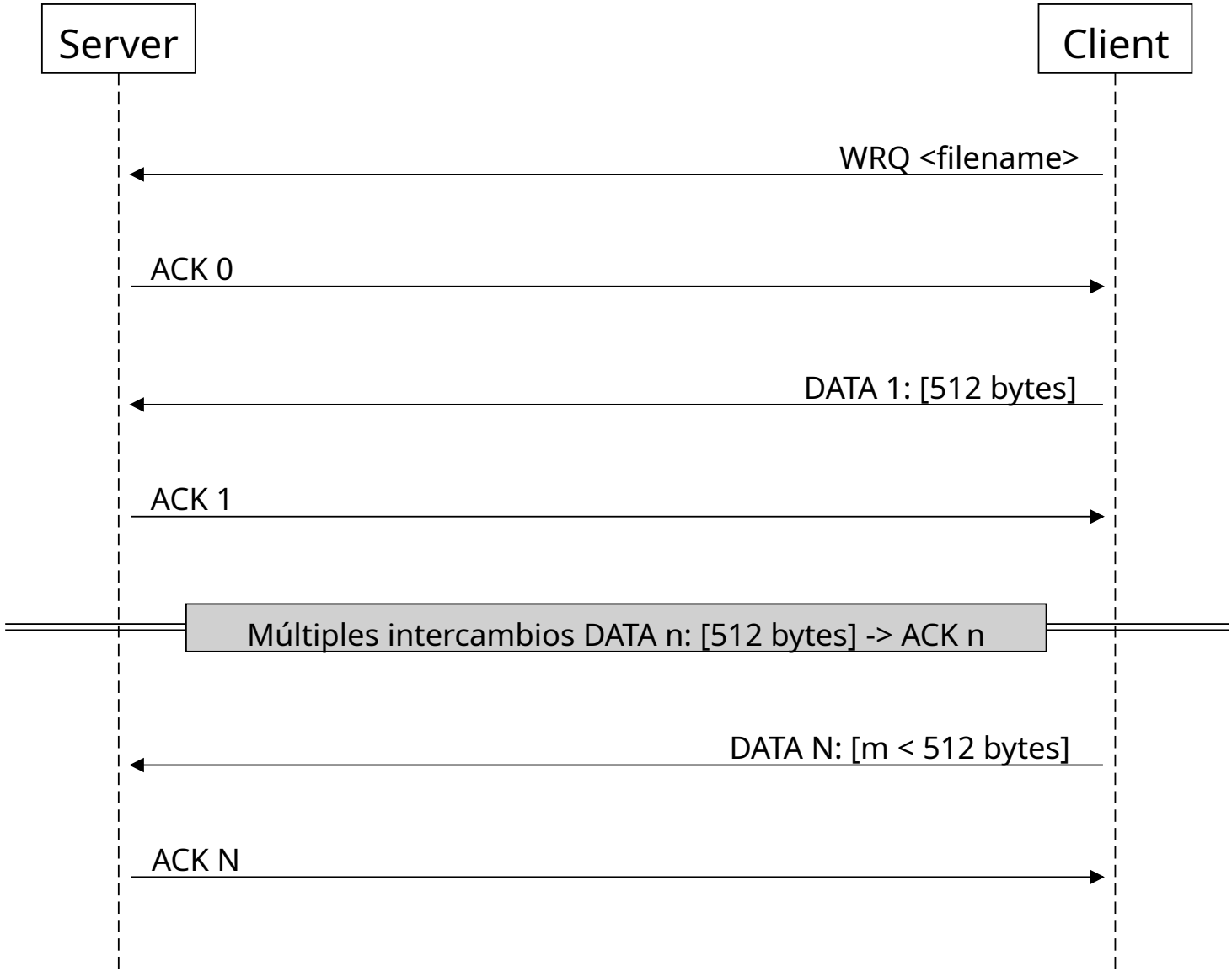
Transferencia de Archivos

- 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



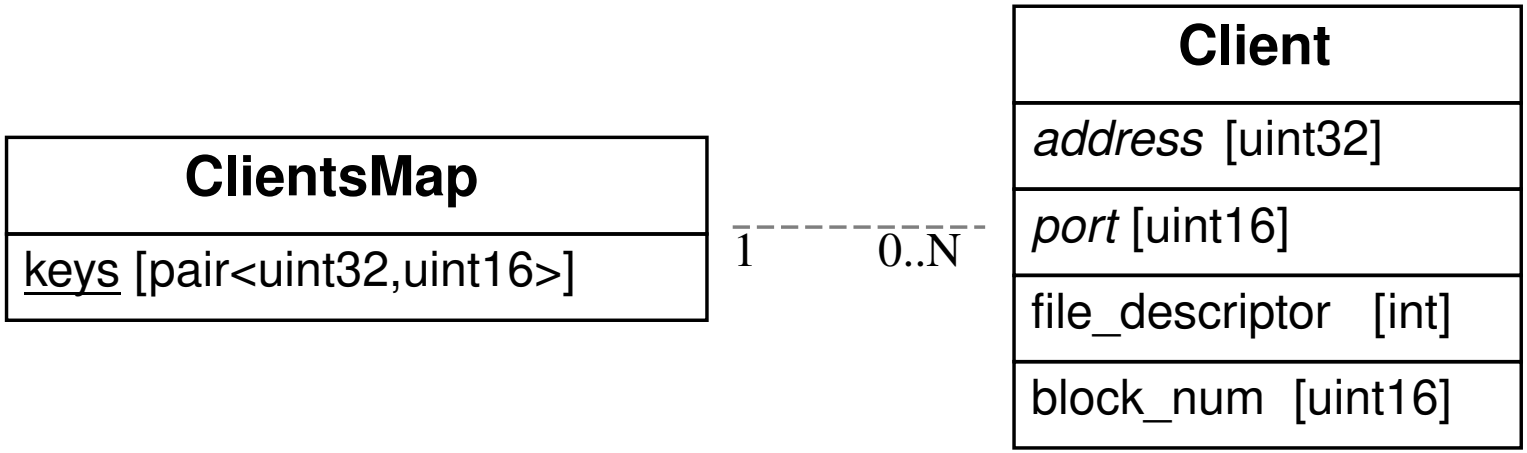
Procedimiento Escritura



Sockets UDP

```
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



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

```
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...
    }
}
```