UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Distributed Systems

Protocol design

Contents

- What is a protocol?
- Syntax
- Semantics
- Synchronization
- TFTP example
- Example: Sockets
- Example: Protocol Buffers

Protocol

Definition:

- Set of rules that allows communication between two or more entities

It involves three aspects:

- **Syntax**
- **Semantics**
- **Synchronization / timing**

Usually written in a non-formal language (ie. human language)

(we are going to see with a real example)

An example: TFTP

Trivial File Transfer Protocol

- Standard (RFC1350)
- Very simple
- Reliable (ACK based)
- Encapsulated over UDP
- Still commonly used
    - Firmware upload

Syntax

Specifies the structure of messages:

- Fields, data types, lengths (# bits/bytes)

TFTP message formats:

```
Type   Op #     Format without header

       2 bytes    string   1 byte     string   1 byte
       -----------------------------------------------
RRQ/  | 01/02 |  Filename  |   0  |    Mode    |   0  |
WRQ    -----------------------------------------------

       2 bytes    2 bytes       n bytes
       ---------------------------------
DATA  | 03    |   Block #  |    Data    |
       ---------------------------------

       2 bytes    2 bytes
       -------------------
ACK   | 04    |   Block #  |
       --------------------

       2 bytes  2 bytes        string    1 byte
       ----------------------------------------
ERROR | 05    |  ErrorCode |   ErrMsg   |   0  |
       ----------------------------------------
```

Semantics

Specifies meaning of fields, allowed values, etc.

TFTP message types:

opcode	operation
1	Read request (RRQ)
2	Write request (WRQ)
3	Data (DATA)
4	Acknowledgment (ACK)
5	Error (ERROR)

TFTP Error Codes:

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

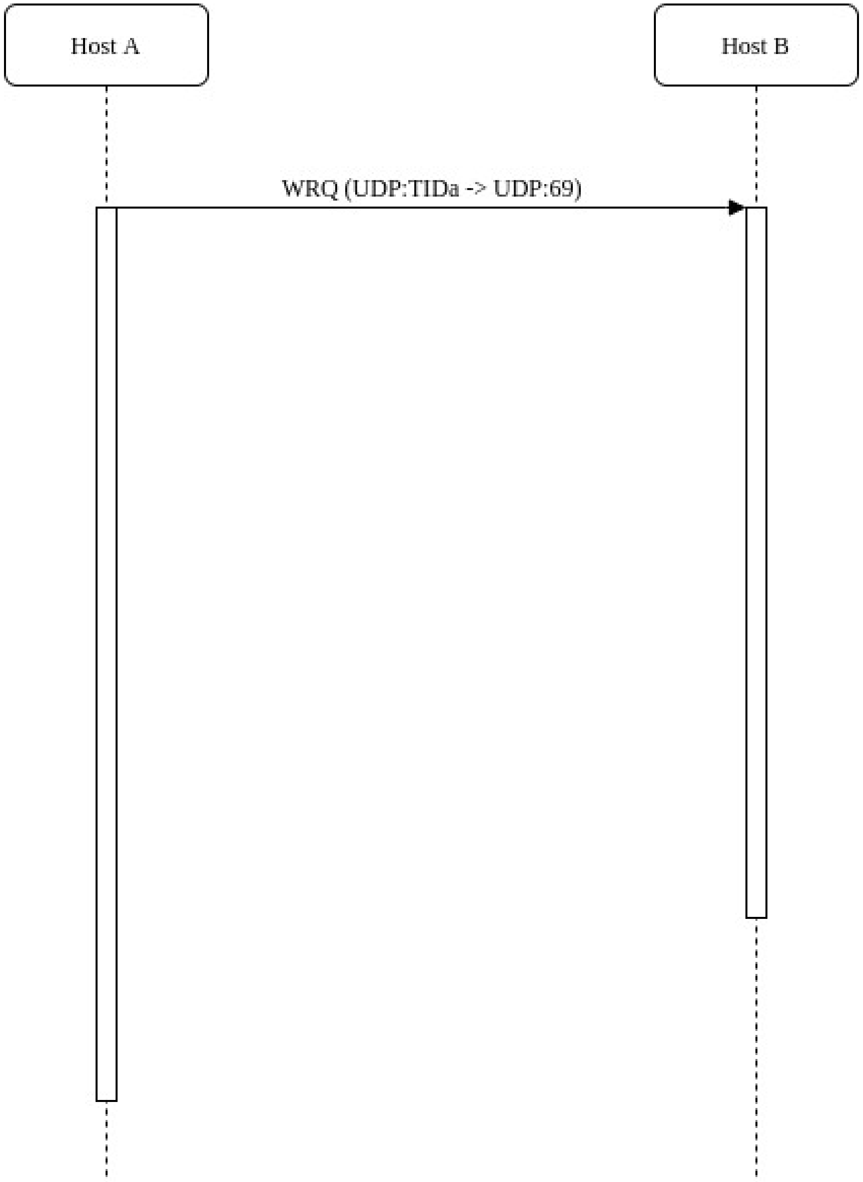
Synchronization

Specifies valid message interchange patterns, communication phases, timers, states, etc.

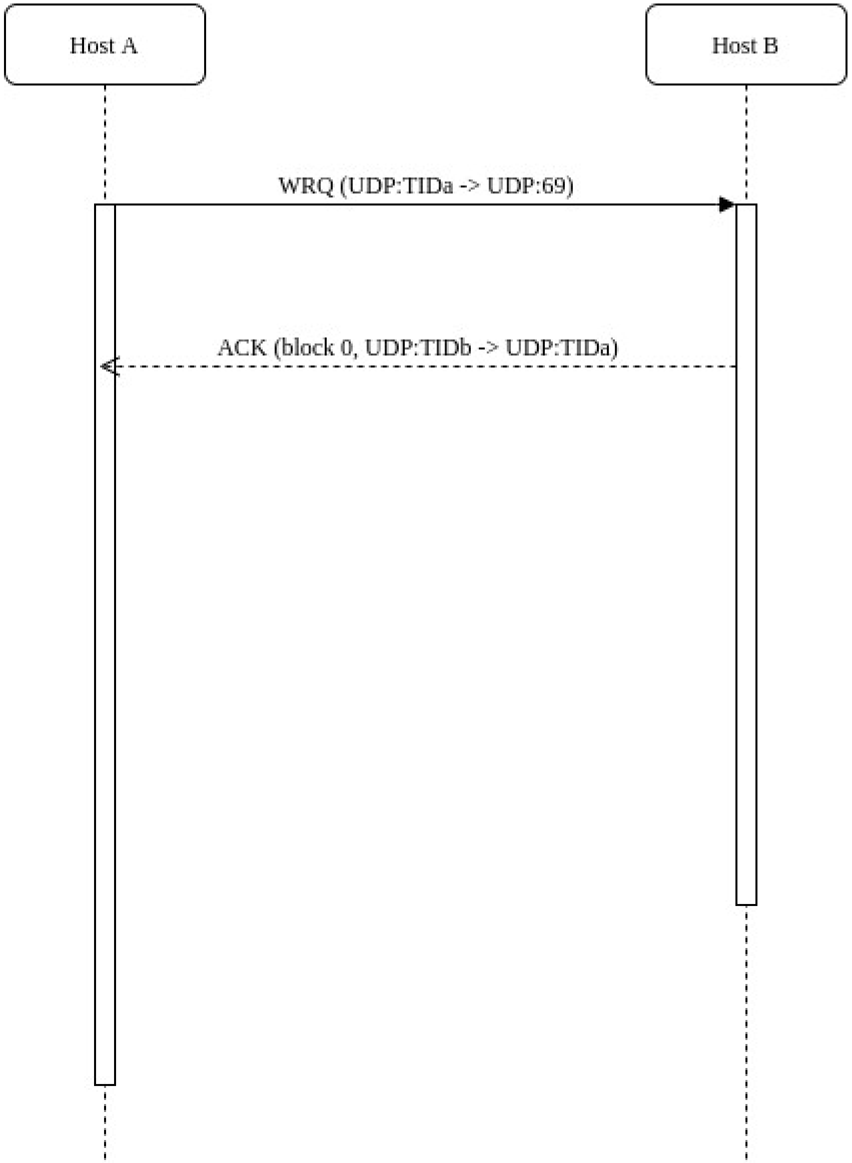
TFTP upload file transfer:

```
1. Host A sends a "WRQ" to host B with source=A's TID,
   destination = 69.

2. Host B sends a "ACK" (with block number=0) to host A with
   source = B's TID, destination= A's TID.

3. Host A sends a “WRQ” (block number=1) with 512B in the DATA field
    (if it is not the last message).
```

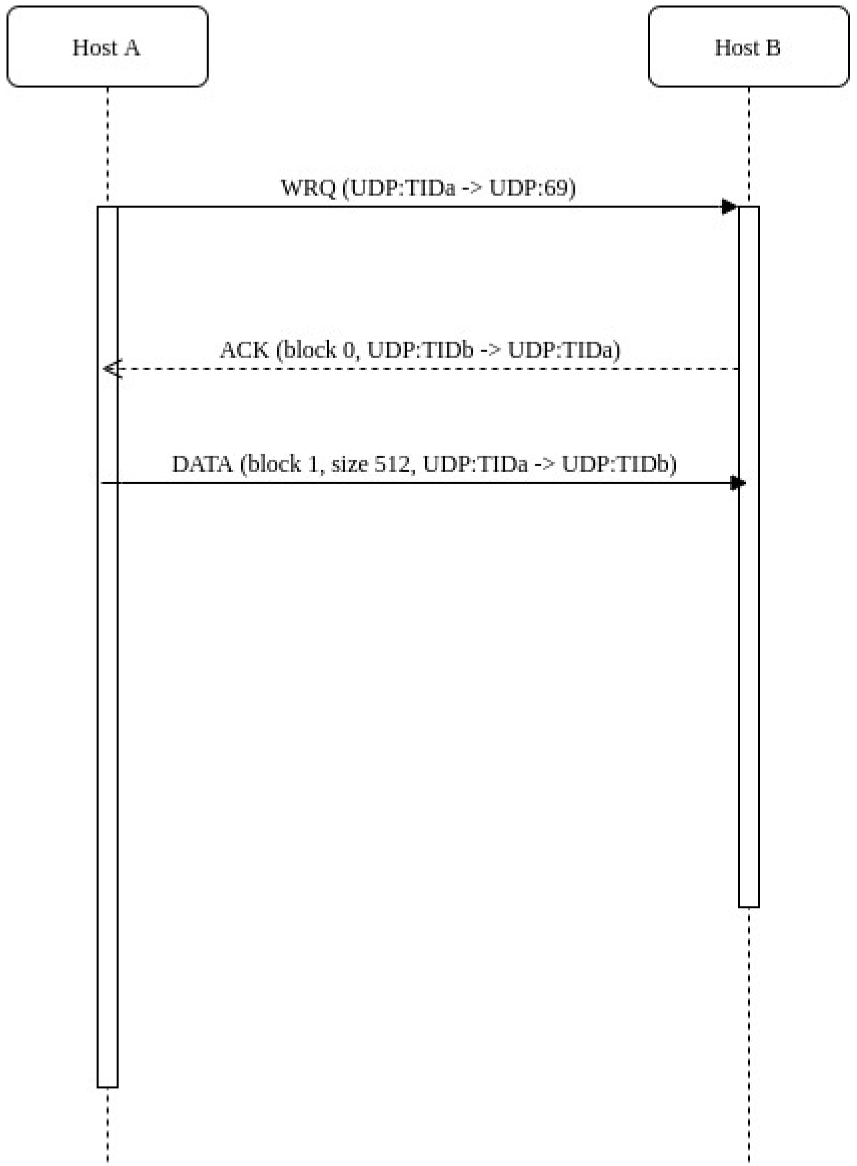
Synchronization



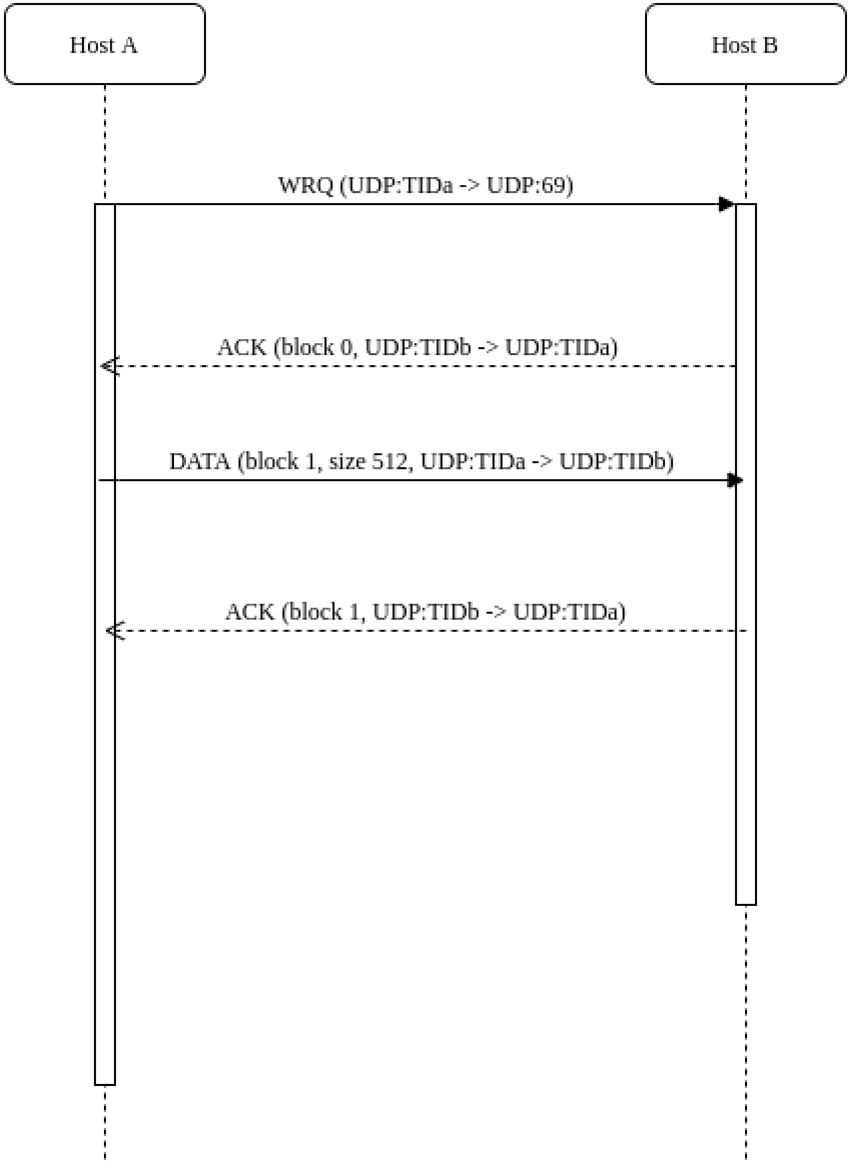
Synchronization



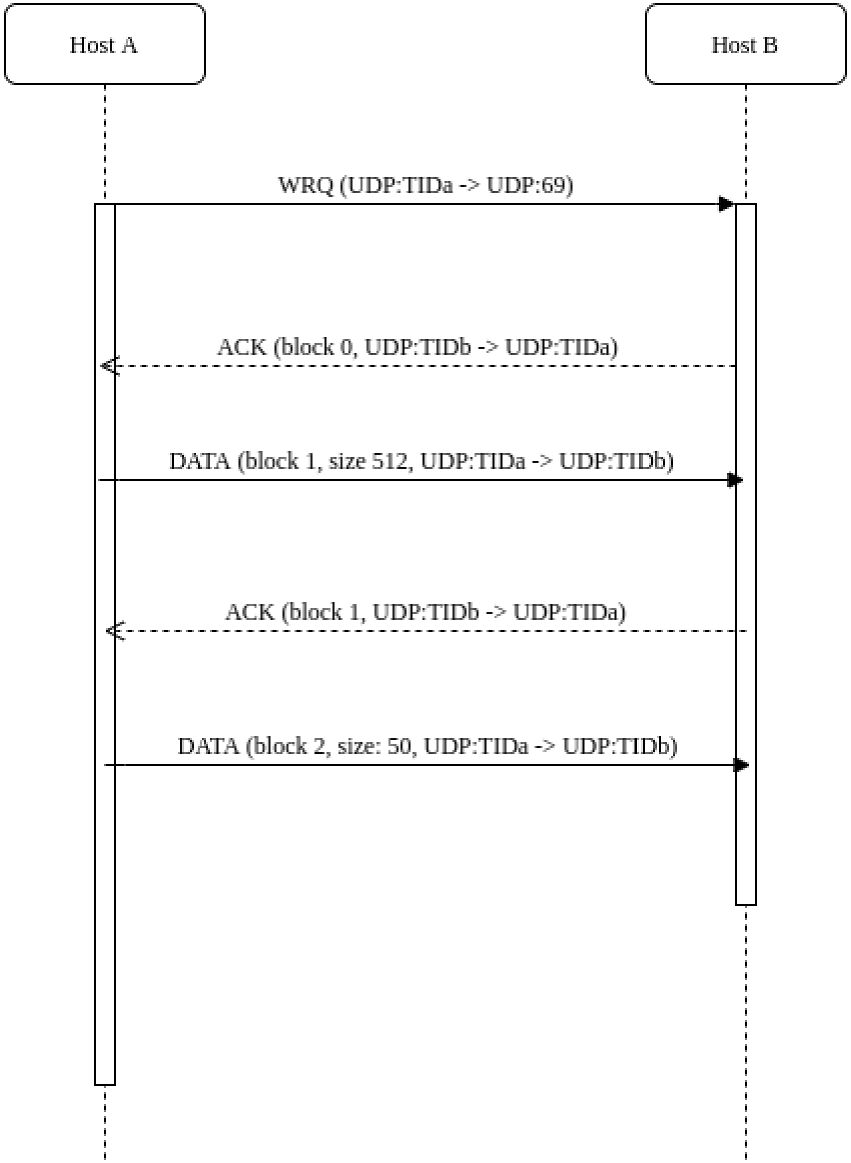
Synchronization


Host A
Host B
WRQ (UDP:TIDa -> UDP:69)
ACK (block 0, UDP:TIDb -> UDP:TIDa)
DATA (block 1, size 512, UDP:TIDa -> UDP:TIDb)

Synchronization


Host A
Host B
WRQ (UDP:TIDa -> UDP:69)
ACK (block 0, UDP:TIDb -> UDP:TIDa)
DATA (block 1, size 512, UDP:TIDa -> UDP:TIDb)
ACK (block 1, UDP:TIDb -> UDP:TIDa)

Synchronization


Host A
Host B
WRQ (UDP:TIDa -> UDP:69)
ACK (block 0, UDP:TIDb -> UDP:TIDa)
DATA (block 1, size 512, UDP:TIDa -> UDP:TIDb)
ACK (block 1, UDP:TIDb -> UDP:TIDa)
DATA (block 2, size: 50, UDP:TIDa -> UDP:TIDb)

TFTP session example DIY

Install server and client:

```
$ sudo apt install tftpd-hpa tftp
```

Put a file on server directory:

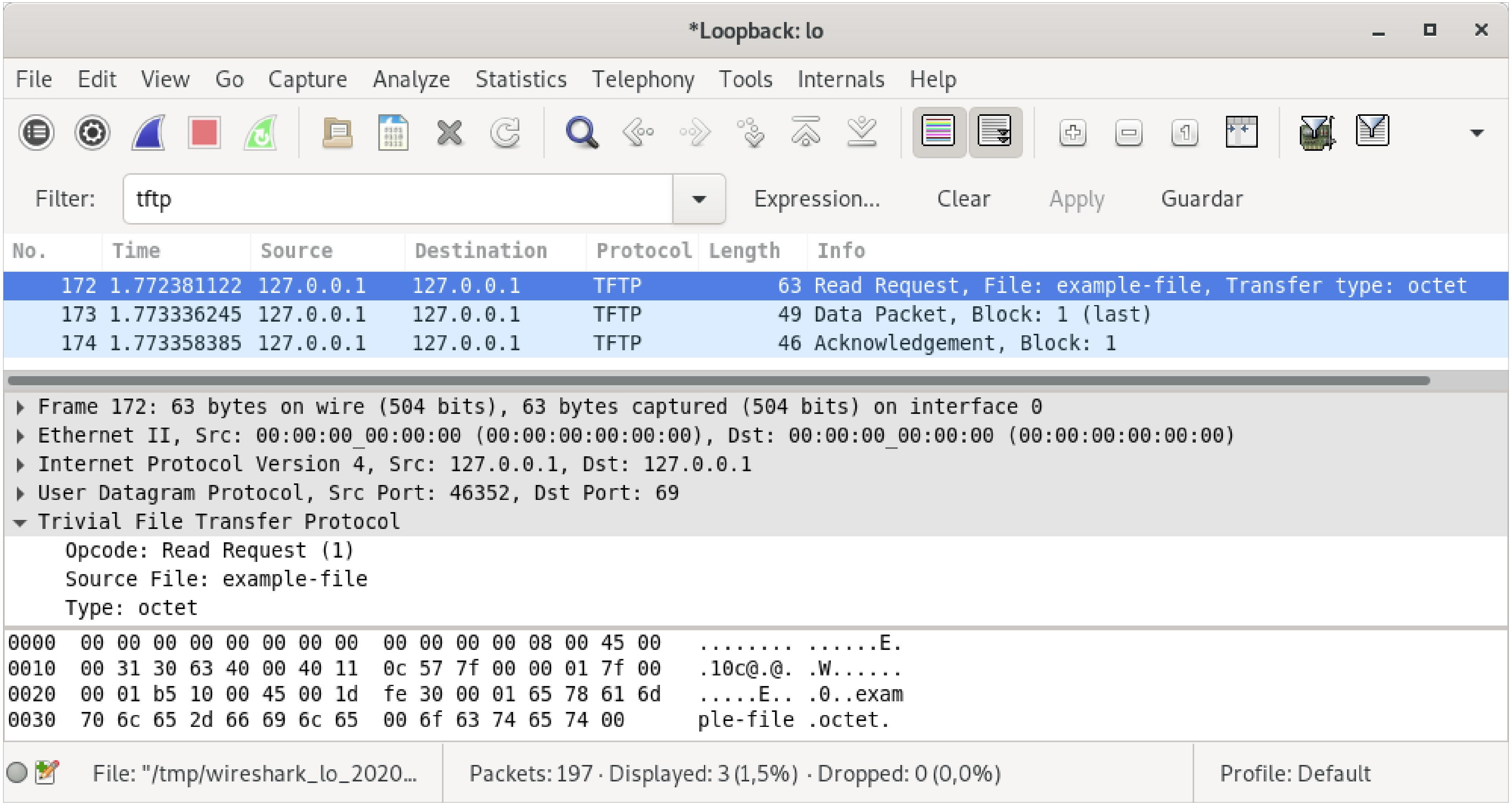
```
$ echo hi | sudo tee /srv/tftp/example-file
```

Download file with the client:

```
$ tftp 0.0.0.0
tftp> trace
Packet tracing on.
tftp> mode binary
tftp> get example-file
sent RRQ <file=example-file, mode=octet>
received DATA <block=1, 3 bytes>
Received 3 bytes in 0.0 seconds
tftp>
```

TFTP session example

DIY



Filter: tftp

```
No.   Time         Source     Destination  Protocol Length Info
  172 1.772381122  127.0.0.1  127.0.0.1    TFTP         63 Read Request, File: example-file, Transfer type: octet
  173 1.773336245  127.0.0.1  127.0.0.1    TFTP         49 Data Packet, Block: 1 (last)
  174 1.773358385  127.0.0.1  127.0.0.1    TFTP         46 Acknowledgement, Block: 1
```

```
▸ Frame 172: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 46352, Dst Port: 69
▾ Trivial File Transfer Protocol
    Opcode: Read Request (1)
    Source File: example-file
    Type: octet
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 31 30 63 40 00 40 11  0c 57 7f 00 00 01 7f 00   .10c@.@. .W......
0020  00 01 b5 10 00 45 00 1d  fe 30 00 01 65 78 61 6d   .....E.. .0..exam
0030  70 6c 65 2d 66 69 6c 65  00 6f 63 74 65 74 00      ple-file .octet.
```

File: "/tmp/wireshark_lo_2020... Packets: 197 · Displayed: 3 (1,5%) · Dropped: 0 (0,0%) Profile: Default

Steps to design a protocol

A protocol design is **not much different to an API or class interface design**.

- Functionality overview. What is the protocol for?
- **Semantics**
  - Involved entities and their relations.
  - Services provided by each entity.
- **Synchronization**
  - Request/reply/ack patterns (if required) per each service.
- **Syntax**
  - Data types and formats for any of the fields and messages.

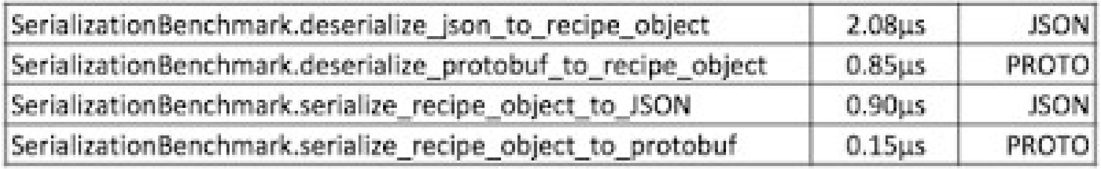
Design considerations

Some non-functional aspects affecting the design:

- Security
  - Confidentiality: encryption, entity validation
  - Integrity: error detection/correction
- Extendability
- Efficiency
  - Marshaling formats: binary, XML, JSON, text.

A marshaling example:
Google Protocol Buffers

- Used in many data oriented services
- Binary marshalling: small messages and fast processing
- Backward compatibility: new protocol version should work with legacy programs.
- Multi-language support: Java, Python, Objective-C, C++, etc.
- [https://developers.google.com/protocol-buffers/]



SerializationBenchmark.deserialize_json_to_recipe_object	2.08µs	JSON
SerializationBenchmark.deserialize_protobuf_to_recipe_object	0.85µs	PROTO
SerializationBenchmark.serialize_recipe_object_to_JSON	0.90µs	JSON
SerializationBenchmark.serialize_recipe_object_to_protobuf	0.15µs	PROTO

JSON request was 789 bytes versus the Protobuf at 518 bytes.

Source: dzone

Protocol Buffers

- Manage marshaling from the programming language structures to binary sequences and vice-versa.
- Builtin types:
  - bool, string, int32, int64, float, double, etc.
  - Enumerations, nested, any, oneof, maps, etc.

Protocol Buffers

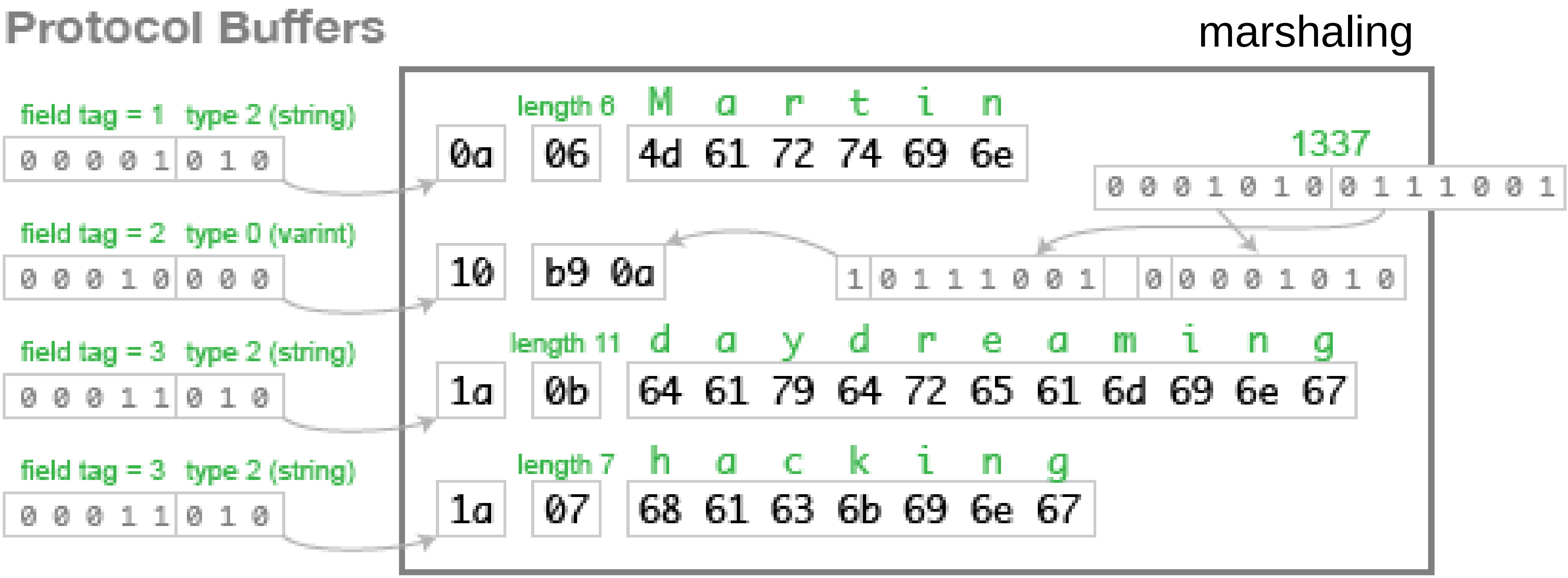
specification

```
syntax = "proto3";

message Person {
  string user_name          = 1;
  int64  favourite_number   = 2;
  repeated string interests = 3;
}
```

input data

```
{
  "userName": "Martin",
  "favouriteNumber": 1337,
  "interests": [
    "daydreaming", "hacking"
  ]
}
```

marshaling



Source: massivetechinterview.blogspot.com.es

Python `struct` Example

DIY

Description: UDP client issuing sensor readings to server

- `[examples:sockets.struct]`

See and play with:
- **`udp-server.py`**
- **`udp-client.py`**

Run server:

```
socket.struct$ ./udp-server.py
New message ('127.0.0.1', 36137)
Sensor 8 (2) value:16.30 bar
```

Run client:

```
socket.struct$ ./udp-client.py localhost
b'\x00\x08\x02A\x82ff\x03bar'
```

Protocol Buffers
Example

DIY

Description: UDP client issuing sensor readings to server

- **[examples:sockets.protobuf]**

See and play with:
- **sensor.proto**
- **udp-server.py**
- **udp-client.py**

Compile:

```
socket.protobuf$ make
protoc -I . --python_out=. sensor.proto
```

Run server:

```
socket.protobuf$ ./udp-server.py
sensor: ('127.0.0.1', 53957),
    raw-data: b'\x08\x01\x10\x01\x1d\xcd\xccL>"\x05kg/m3'
Sensor 1 (HUMIDITY) value:0.20 kg/m3
```

Run client:

```
socket.protobuf$ ./udp-client.py localhost
```

What you have learned?

- Open and public protocols **decouple** implementations
  - Provide transparency
  - Ensure interoperability
- Protocols are contracts among services and clients
- Protocol specifications require:
  - Syntax, Semantics and Synchronization
- Marshalling format election impacts on efficiency

References

G. Coulouris, *Distributed Systems: Concepts and Design*, Addison Wesley 2011

- Section 4.3 – External data representation and marshalling