# UDP Programming

RES, Lecture 3

Olivier Liechti

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

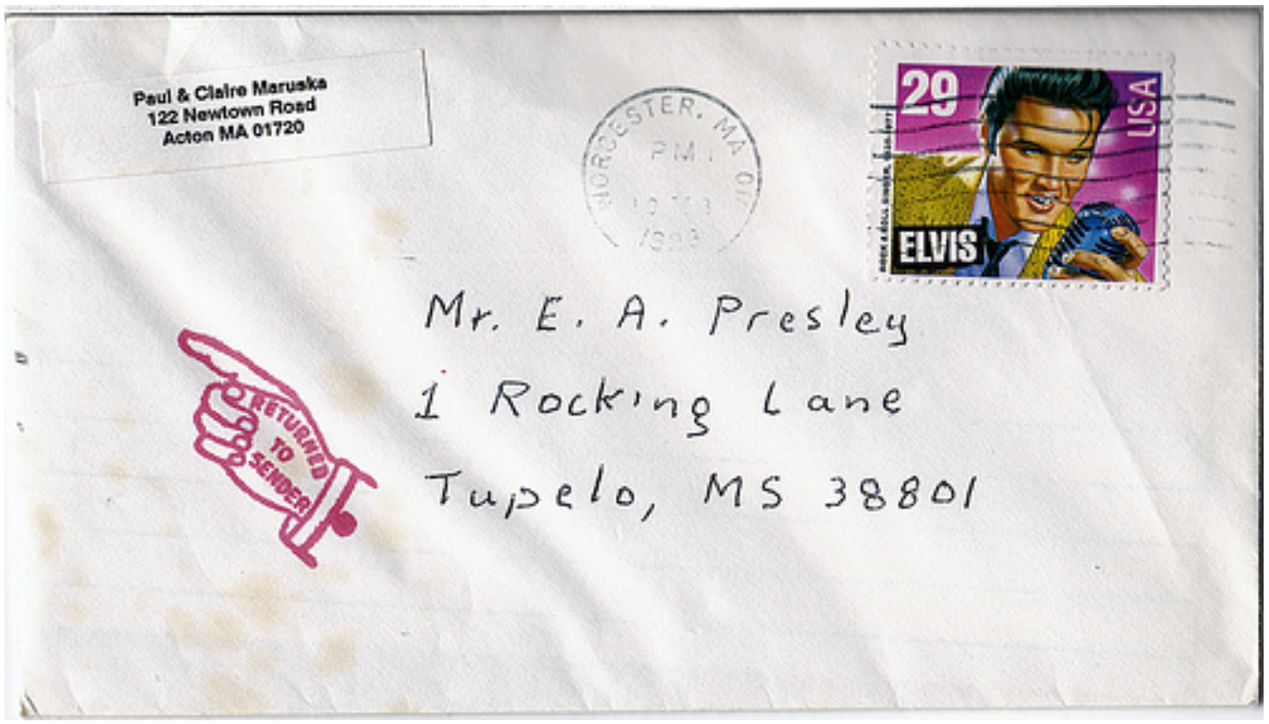
heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

TCP

UDP

# The UDP Protocol



# The UDP Protocol

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- When using UDP, **you do not work with the abstraction of an IO stream**.
- Rather, **you work with the abstraction of individual datagrams**, which you can send and receive. Every datagram is independent from the others.
- You **do not have any guarantee**: datagrams can get lost, datagrams can arrive out-of-order, datagrams can be duplicated.

- What do you put "**inside**" the datagram (i.e. what is the payload)?
  - A notification.
  - A request, a query, a command. A reply, a response, a result.
  - A portion of a data stream (managed by the application).
- What do you put "**outside**" of the datagram (i.e. what is the header)?
  - A destination address (IP address in the IP packet header + UDP port)
  - A source address (IP + port)

# Unicast, Broadcast, Multicast

## Destination Address

192.168.10.2

255.255.255.255

239.255.22.5

IANA Guidelines for IPv4 Multicast Address Assignments
http://tools.ietf.org/html/rfc5771

# Using the Socket API for a UDP **Sender**

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

1. Create a datagram socket (without giving any address / port)

2. Create a datagram and put some data (bytes) in it

3. Send the datagram via the socket, specifying the destination address and port

**If a reply is expected:**

4. Accept incoming datagrams on the socket

If we do not specify the port when creating the socket, the operating system will **automatically assign a free one** for us.

This port will be in the "source port" field of the UDP header. The receiver of our datagram will extract this value and will use it to send us the reply (it is a part of the ***return address***.

# Using the Socket API for a UDP **Receiver**

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Application-level protocol specifications often define **a standard UDP port**, where the clients can send their requests. When this is the case, we have to specify the port number when creating the socket.

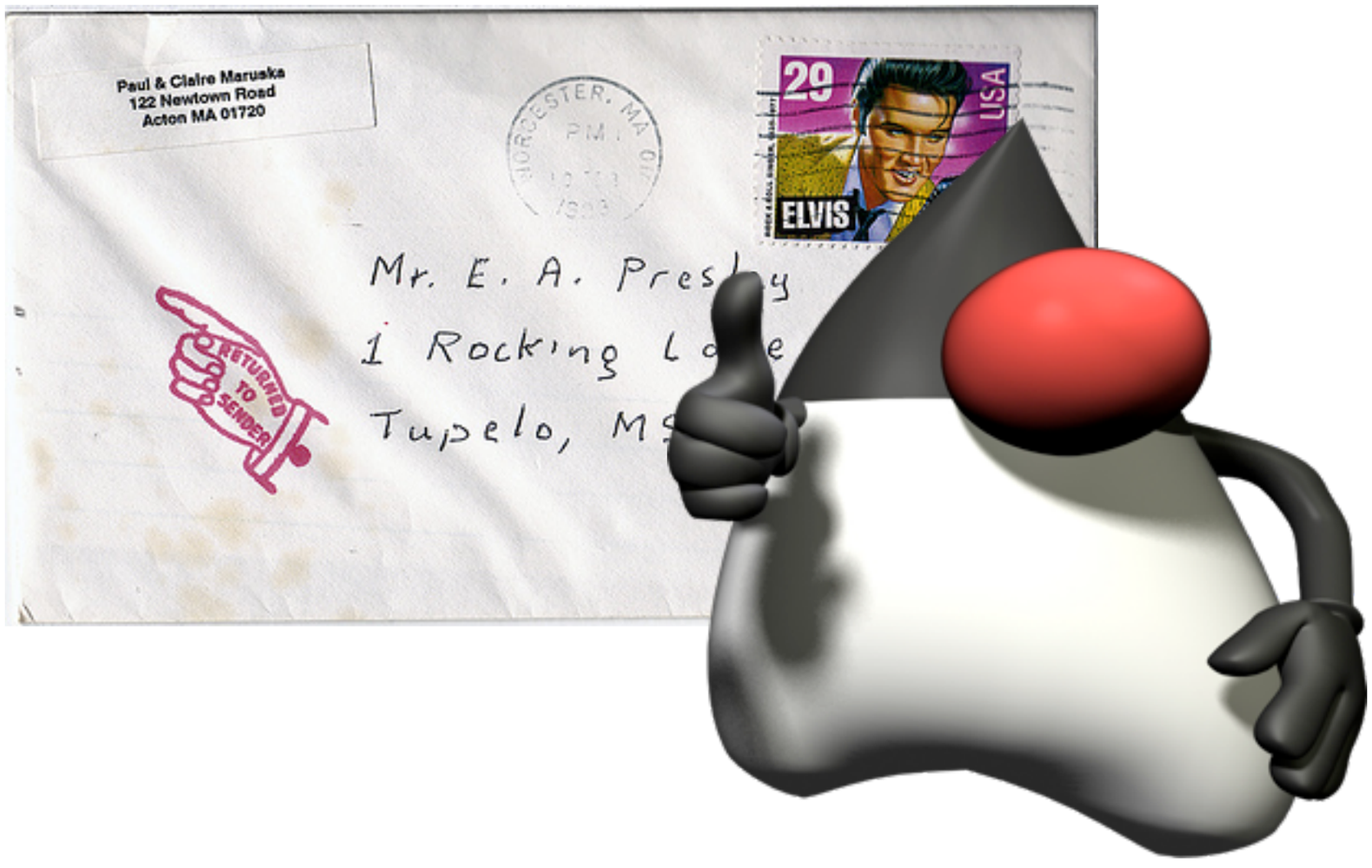
```
1. Create a datagram socket, specifying a particular port

2. Loop
   2.1. Accept an incoming datagram via the socket
   2.2. Process the datagram
   2.3. If a reply is expected by the sender
        2.3.1. Extract the return address from the request datagram
        2.3.2. Prepare a reply datagram
        2.3.3. Send the reply datagram via the socket
```

It is at this stage that we extract the source port and source IP address from the incoming datagram. We have found the ***return address***, so we know where to the send the response.

Example: **09-thermometers**

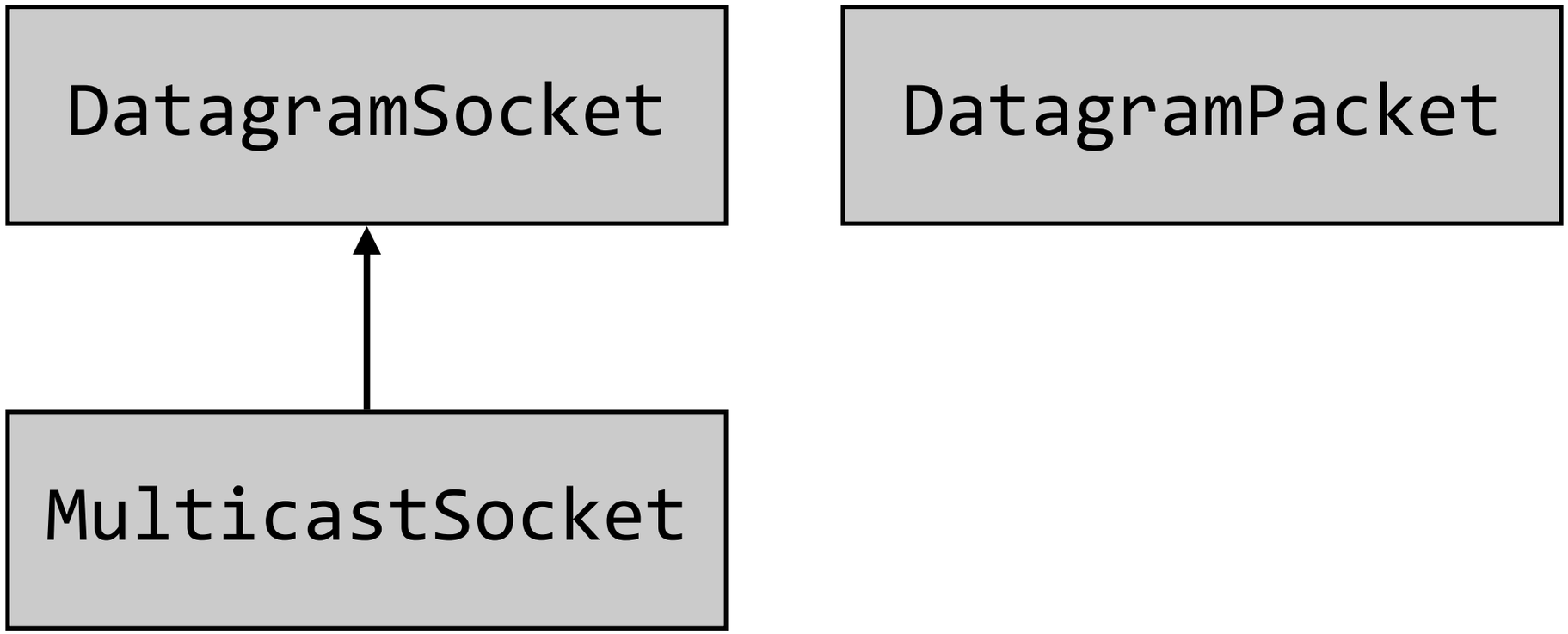
Example: **10-BroadcastWithNodeJS**

# Using UDP in Java



# Using UDP in Java

`java.net`



```
// Sending a message to a multicast group

socket = new DatagramSocket();
byte[] payload = "Java is cool, everybody should know!!".getBytes();

DatagramPacket datagram = new DatagramPacket(payload, payload.length,
InetAddress.getByName("239.255.3.5"), 4411);

socket.send(datagram);
```

# Publisher (multicast)

# Subscriber (multicast)

```
// Listening for broadcasted messages on the local network

MultiCastSocket socket = new MultiCastSocket(port);
InetAddress multicastGroup = InetAddress.getByName("239.255.3.5");
socket.joinGroup(multicastGroup);

while (true) {
    byte[] buffer = new byte[2048];
    DatagramPacket datagram = new DatagramPacket(buffer, buffer.length);
    try {
        socket.receive(datagram);
        String msg = new String(datagram.getData(), datagram.getOffset(), datagram.getLength());
        LOG.log(Level.INFO, "Received a datagram with this message: " + msg);
    } catch (IOException ex) {
        LOG.log(Level.SEVERE, ex.getMessage(), ex);
    }
}
socket.leaveGroup(multicastGroup);
```

```
// Broadcasting a message to all nodes on the local network

socket = new DatagramSocket();
socket.setBroadcast(true);

byte[] payload = "Java is cool, everybody should know!!".getBytes();

DatagramPacket datagram = new DatagramPacket(payload, payload.length,
InetAddress.getByName("255.255.255.255"), 4411);

socket.send(datagram);
```

Publisher

Subscriber

```
// Listening for broadcasted messages on the local network

DatagramSocket socket = new DatagramSocket(port);

while (true) {
    byte[] buffer = new byte[2048];
    DatagramPacket datagram = new DatagramPacket(buffer, buffer.length);
    try {
        socket.receive(datagram);
        String msg = new String(datagram.getData(), datagram.getOffset(), datagram.getLength());
        LOG.log(Level.INFO, "Received a datagram with this message: " + msg);
    } catch (IOException ex) {
        LOG.log(Level.SEVERE, ex.getMessage(), ex);
    }
}
```

Example: **11-BroadcastMulticastWithJava**

# Reliability

# Reliability

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- UDP is **not a reliable transport protocol**:
  - Datagrams can arrive in a different order from which they were sent.
  - Datagrams can get lost.
- For some application-level protocols, this is not an issue because they are **tolerant to data loss**. For example, think of a **media streaming** protocol.
- But **if no data loss can be tolerated at the application level** (which is typically the case for file transfer protocols), does it mean that it is impossible to use UDP?

# Who Is Responsible for Reliability?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **It is actually possible** to implement reliable application-level protocols on top of UDP, but...
  - It is the **responsibility of the application-level protocol specification** to include appropriate mechanisms to overcome the limitations of UDP.
  - It is the **responsibility of the application developer** to implement these mechanisms.
- In other words, it is up to the application developer to "**do the kind of stuff that TCP usually does**".
- Do you remember about **acknowledgments**, **timers**, **retransmissions**, **stop-and-wait**, **sliding windows**, etc?

# A First Example: TFTP

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Trivial File Transfer Protocol (TFTP)**
  - http://tools.ietf.org/html/rfc1350
  - "TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. **Each nonterminal packet is acknowledged separately.** This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions."
  - "Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet **before** the next packet can be sent."

# A Second Example: CoAP

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

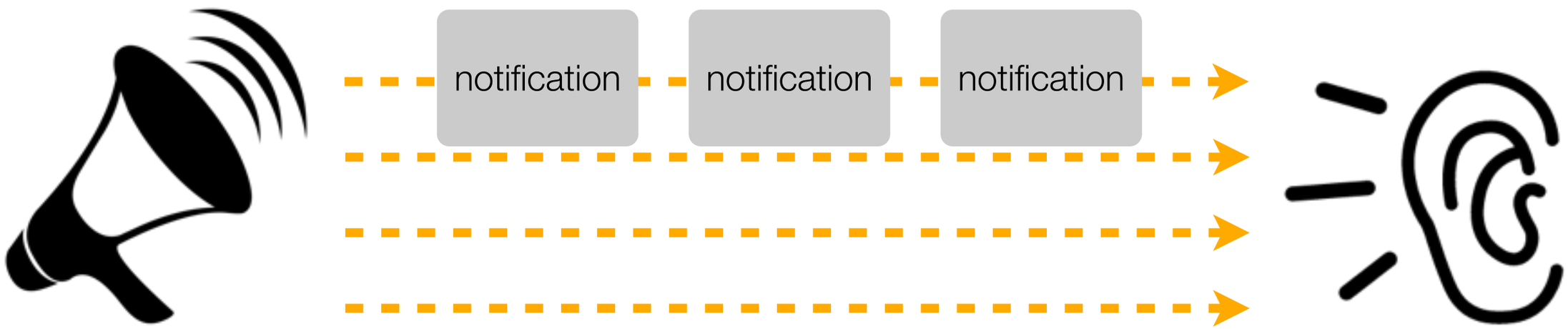
- **Constrained Application Protocol (CoAP)**
  - http://tools.ietf.org/html/draft-ietf-core-coap-18
  - "The Constrained Application Protocol (CoAP) is a **specialized web transfer protocol** for use with constrained nodes and constrained (e.g., low-power, lossy) networks. [...] The protocol is designed for **machine-to-machine (M2M) applications** such as smart energy and building automation."
  - "CoAP provides a **request/response interaction model** between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types."
- "**2.1. Messaging Model**. The CoAP messaging model is based on the exchange of messages over UDP between endpoints. CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message format is shared by requests and responses. The CoAP message format is specified in Section 3. **Each message contains a Message ID used to detect duplicates and for optional reliability.** (The Message ID is compact; its 16-bit size enables up to about 250 messages per second from one endpoint to another with default protocol parameters.)
- "**4.2. Messages Transmitted Reliably**. The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header. [...] A recipient MUST **acknowledge** a Confirmable message with an Acknowledgement message [...].

# Messaging Patterns

# Pattern: Fire-and-Forget

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

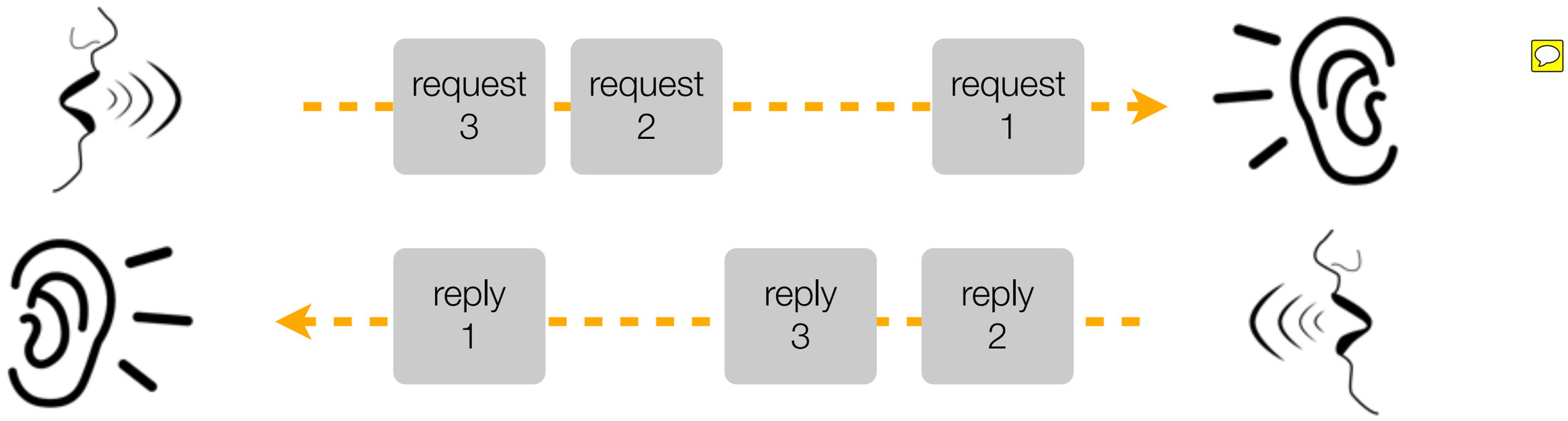
- The sender generates messages, sends them to a single receiver or to a group of receivers.
- The sender may do that on a **periodic** basis.
- The sender does not expect any answer. **He is telling, not asking**.



# Pattern: Request-Reply

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- This pattern is used to implement the **typical client-server model**. Both the client and the server **produce and consume** datagrams.
- **The client produces requests** (aka queries, commands). The client also listens for incoming replies (aka responses, results). The server listens for requests. **The server also sends replies back to the client**.
- Can the client send a new request, even if he has not received the response to the previous request yet? If yes, and because UDP datagrams can be delivered out of order, how can the client associate a reply to the corresponding request?



# Service Discovery Protocols

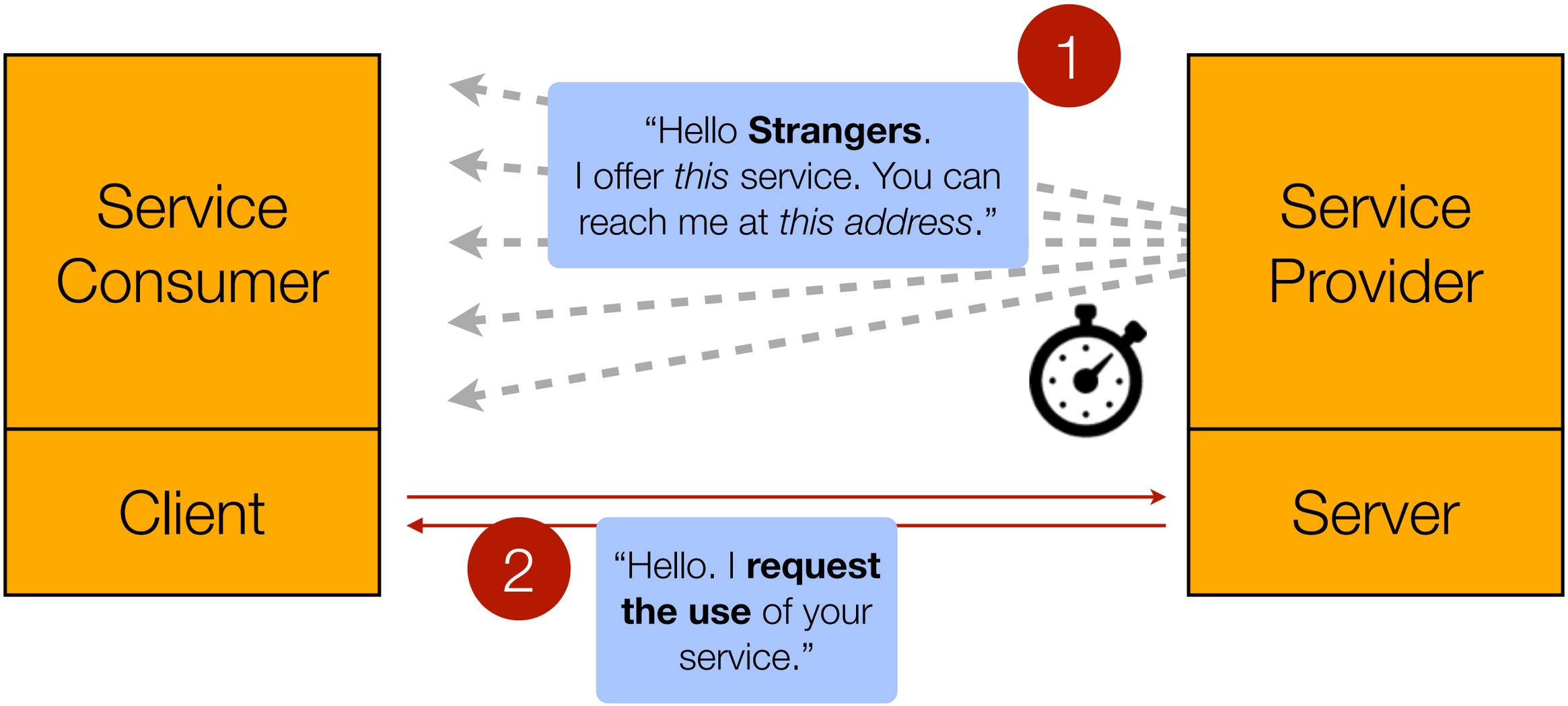
# Service Discovery Protocols

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- With **unicast** message transmission, the sender must know who the receiver is and must know how to reach it (i.e. know its address).
- With **broadcast** and **multicast**, this is not required. The sender knows that nodes that are either *nearby*, or that have *expressed their interest*, will receive a copy of the message.
- This property can be used to **discover the availability of services or resources** in a *dynamic environment*.

# Model 1: Advertise Service Periodically

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



# Model 2: Query Service Availability

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud