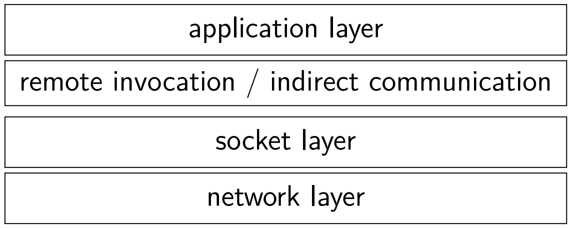
Remote invocation

Johan Montelius

KTH

HT15

middleware



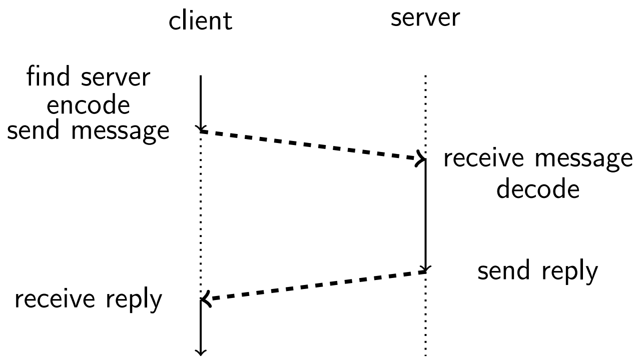
application layer

remote invocation / indirect communication

socket layer

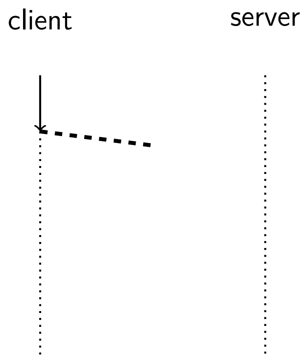
network layer

Request reply



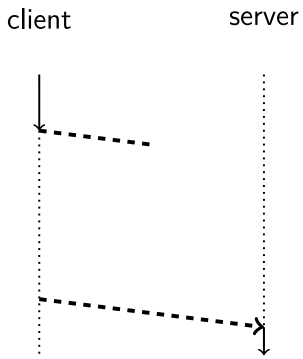
- identify and locate the server
- encode/decode the message
- send reply to the right client
- attach reply to request

lost request



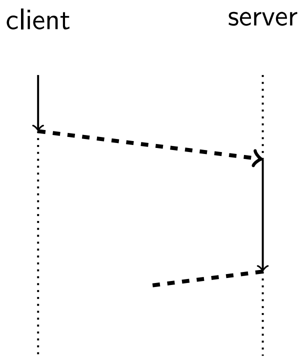
What do we do if request is lost?

resend request



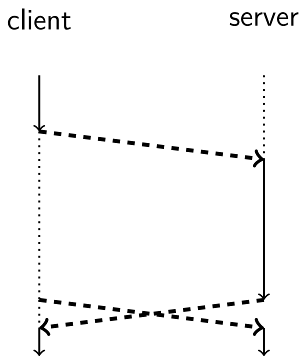
- need to detect that message is potentially lost
- wait for a timeout (how long) or error from underlying layer
- resend the request
- simple, problem solved

lost reply



- client will wait for timeout and re-send request
- not a problem

problem



- a problem
- server might need a history of all previous request
- *might need*

idempotent operations

- add 100 euros to my account
- what is the status of my account
- Sweden scored yet another goal!
- The standing is now 2-1!

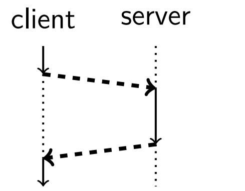
history

If operations are not idempotent, the server must make sure that the same request is not executed twice.

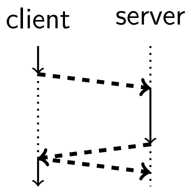
Keep a history of all request and the replies. If a request is resent the same reply can be sent without re-execution.

For how long do you keep the history?

request-reply-acknowledge



Request-Reply (RR)



Request-Reply-Acknowledge (RRA)

at most or at least once

How about this:

If an operation succeeds, then..

at-most-once: the request has been executed once.

> *Implemented using a history or simply not resending requests.*

at-least-once: the request has been executed at least once.

> *No need for a history, simply resend requests until a reply is received.*

at most or at least

What about errors:

> *Even if we do resend messages we will have to giv up at some time.*

If an operation fails/is lost, then..

at-most-once:

at-least-once:

at most or at least

Pros and cons:

- *at-most-once without resending requests:* simple to implement, not fault-tolerant
- *at-most-once with history:* expensive to implement, fault-tolerant
- *at-least-once:* simple to implement, fault-tolerant

Can you live with at-least-once semantics?

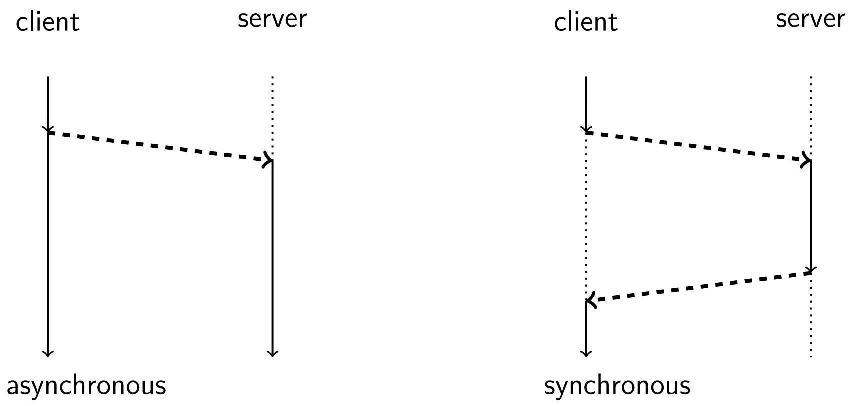
Erlang

What does Erlang message passing give you?

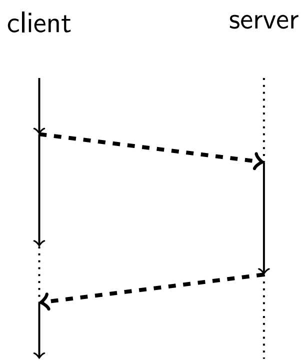
UDP or TCP

Should we implement a request-reply protocol over UDP or TCP?

synchronous/asynchronous

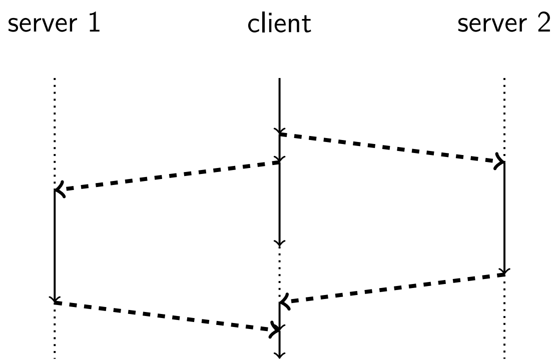


RR over asynchronous



- send request
- continue to execute
- suspend if not arrived
- read reply

hide the latency



HTTP

A request reply protocol, described in RFC 2616.

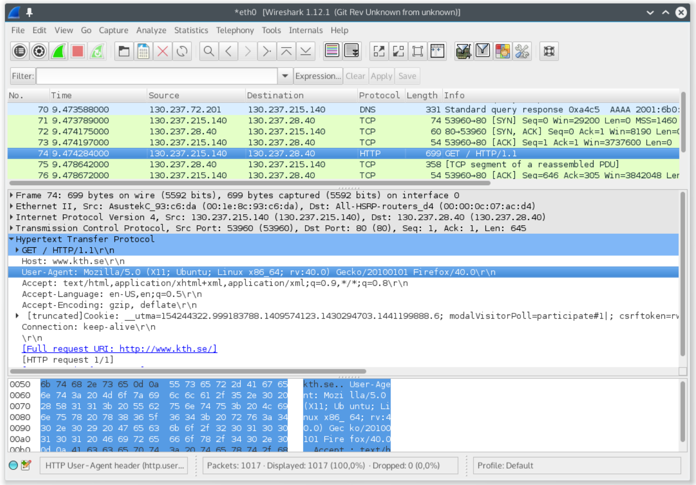
```
Request        = Request-Line *(header CRLF) CRLF [ message-body ]

Request-Line   = Method SP Request-URI SP HTTP-Version CRLF
```

```
GET /index.html HTTP/1.1\r\n foo 42 \r\n\r\nHello
```

HTTP methods

- GET: request a resource, *should be idempotent*
- HEAD: request only header information
- POST: upload information to a resource, included in body, status of server could change
- PUT: add or replace a resource, idempotent
- DELETE: add or replace content, idempotent

Wireshark


*eth0 [Wireshark 1.12.1 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

```
No.   Time          Source            Destination       Protocol Length Info
   70 9.473588000   130.237.72.201    130.237.215.140   DNS       331 Standard query response 0xa4c5  AAAA 2001:6b0:
   71 9.473789000   130.237.215.140   130.237.28.40     TCP        74 53960→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
   72 9.474175000   130.237.28.40     130.237.215.140   TCP        60 80→53960 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=(
   73 9.474197000   130.237.215.140   130.237.28.40     TCP        54 53960→80 [ACK] Seq=1 Ack=1 Win=3737600 Len=0
   74 9.474284000   130.237.215.140   130.237.28.40     HTTP      699 GET / HTTP/1.1
   75 9.478642000   130.237.28.40     130.237.215.140   TCP       358 [TCP segment of a reassembled PDU]
   76 9.478672000   130.237.215.140   130.237.28.40     TCP        54 53960→80 [ACK] Seq=646 Ack=305 Win=3842048 Ler
```

```
▸ Frame 74: 699 bytes on wire (5592 bits), 699 bytes captured (5592 bits) on interface 0
▸ Ethernet II, Src: AsustekC_93:c6:da (00:1e:8c:93:c6:da), Dst: All-HSRP-routers_d4 (00:00:0c:07:ac:d4)
▸ Internet Protocol Version 4, Src: 130.237.215.140 (130.237.215.140), Dst: 130.237.28.40 (130.237.28.40)
▸ Transmission Control Protocol, Src Port: 53960 (53960), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 645
▾ Hypertext Transfer Protocol
  ▸ GET / HTTP/1.1\r\n
    Host: www.kth.se\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
  ▸ [truncated]Cookie: __utma=154244322.999183788.1409574123.1430294703.1441199888.6; modalVisitorPoll=participate#1|; csrftoken=rv
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://www.kth.se/]
    [HTTP request 1/1]
```

```
0050  6b 74 68 2e 73 65 0d 0a  55 73 65 72 2d 41 67 65   kth.se.. User-Age
0060  6e 74 3a 20 4d 6f 7a 69  6c 6c 61 2f 35 2e 30 20   nt: Mozi lla/5.0 
0070  28 58 31 31 3b 20 55 62  75 6e 74 75 3b 20 4c 69   (X11; Ub untu; Li
0080  6e 75 78 20 78 38 36 5f  36 34 3b 20 72 76 3a 34   nux x86_ 64; rv:4
0090  30 2e 30 29 20 47 65 63  6b 6f 2f 32 30 31 30 30   0.0) Gec ko/20100
00a0  31 30 31 20 46 69 72 65  66 6f 78 2f 34 30 2e 30   101 Fire fox/40.0
00b0  0d 0a 41 63 63 65 70 74  3a 20 74 65 78 74 2f 68   ..Accept : text/h
```

HTTP User-Agent header (http.user... | Packets: 1017 · Displayed: 1017 (100,0%) · Dropped: 0 (0,0%) | Profile: Default

HTTP GET

```
GET / HTTP/1.1
Host: www.kth.se
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/20
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
Accept-Language: en-US,en;q=0.5
Accept.Encoding: gzip, deflate
Cookie: ......
Connection: keep-alive
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Tue, 08 Sep 2015 10:37:49 GMT
Server: Apache/2.2.15 (Red Hat)
X-UA-Compatible: IE=edge
Set-Cookie: JSESSIONID=CDC76A3;Path=/; Secure; HttpOnly
Content-Language: sv-SE
Content-Length: 59507
Connection: close
Content-Type: text/html;charset=UTF-8

<!DOCTYPE html>

<html lang="sv">
<title>KTH | Valkommen till KTH</title>
```

the web

On the *web* the resource i often a HTML document that is presented in a browser.

HTTP could be used as a general purpose request-reply protocol.

REST and SOAP

Request-reply protocols for Web-services:

- REST (Representational State Transfer)
  - content described in XML, JSON, . . .
  - light weight,
- SOAP (Simple Object Access Protocol)
  - over HTTP, SMTP . . .
  - content described in SOAP/XML
  - standardized, heavy weight

HTTP over TCP

HTTP over TCP - a good idea?

masking a request-reply

Could we use a regular program construct to hide the fact that we do a request-reply?

- RPC: remote procedure call
- RMI: remote method invocation

procedure calls

What is a procedure call:

- find the procedure
- give the procedure access to arguments
- pass control to the procedure
- collect the reply if any
- continue execution

How do we turn this into a tool for distributed programming?

operational semantics

```
int x, n;
n = 5;
proc(n);
x = n;
```

```
int x, arr[3];
arr[0] = 5;
proc(arr);
x = arr[0];
```

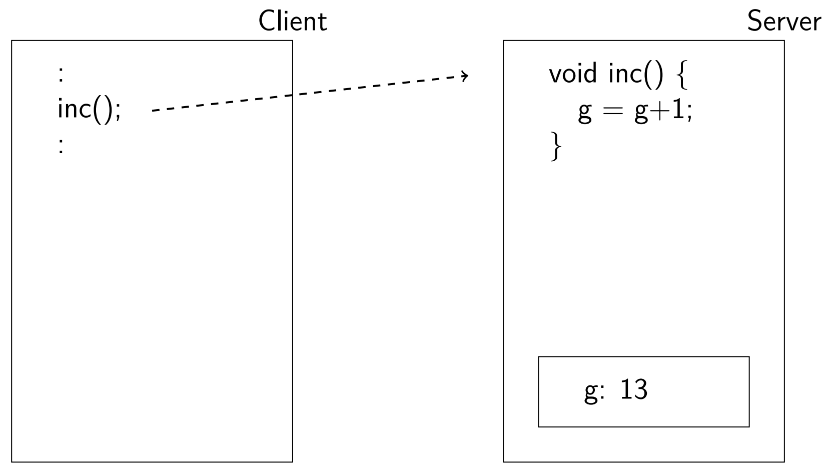
call by value/reference

- call by value
  - procedures are given a copy of the datum
- call by reference
  - procedures are given a reference to the datum

what if the datum is a reference and we pass a copy of the datum

why is this important?

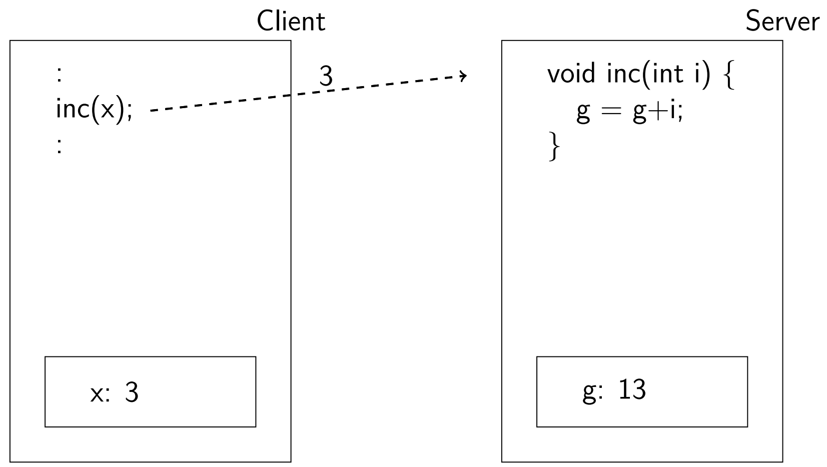
remote procedure call



```
:
inc();
:
```

```
void inc() {
  g = g+1;
}
```

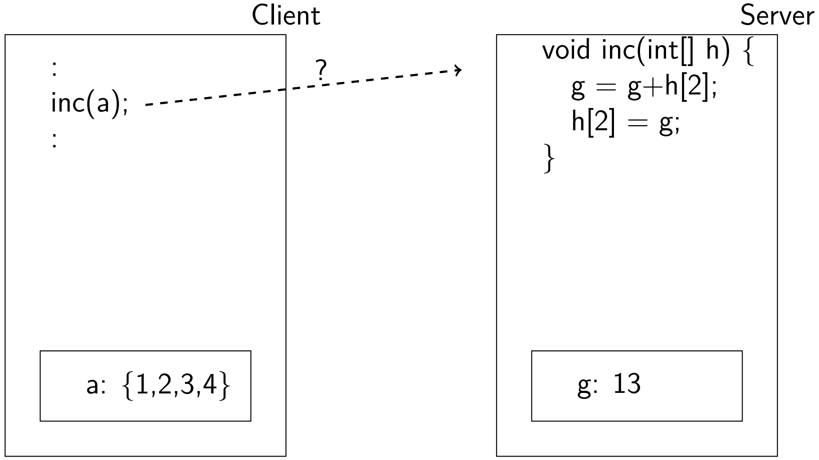
remote procedure call



```
:
inc(x);
:
```

```
void inc(int i) {
  g = g+i;
}
```

remote procedure call



```
void inc(int[] h) {
  g = g+h[2];
  h[2] = g;
}
```

ONC RPC (SunRPC)

- targeting intranet, file servers etc
- used UDP as transport protocol (TCP also available)
- at-least-once call semantics
- XDR (eXternal Data Representation) specifies message structure

ONC RPC (SunRPC)

- targeting intranet, file servers etc
- used UDP as transport protocol (TCP also available)
- at-least-once call semantics
- XDR (eXternal Data Representation) specifies message structure

Java RMI

- similar to RPC but:
  - we now invoke methods of *remote objects*
  - at-most-once semantics
- Objects can be passed as arguments, how should this be done?
  - by value
  - by reference

Java RMI

We can do either:

Remote objects are passed as references i.e. they remain as one object.

Serializable objects are passed as copies i.e. the object is duplicated.

finding the procedure

How do we locate a remote procedure/object/process?

Network address that specifies the location or..

a known “binder” process that keeps track of registered resources.

remote invocation design decisions

- failure handling: maybe / at-most-once / at-least-once
- call-by-value / call-by-reference
- message specification and encoding
- specification of resource
- procedure binder

examples

- SunRPC: call-by-value, at-lest-once, XDR, binder
- JavaRMI: call-by-value/reference, at-most-once, interface, JRMP, registry
- Erlang: message passing, maybe, no, ETF, local registry only

more

- CORBA: (interface description language) IDL, (object request broker) ORB
- Web Services: WSDL, UDDI

summary

Implementations of remote invocations: procedures, methods, messages to processes, have fundamental problems that needs to be solved.

Try to see similarities between different implementations.

When they differ, is it fundamentally different or just implementation details.