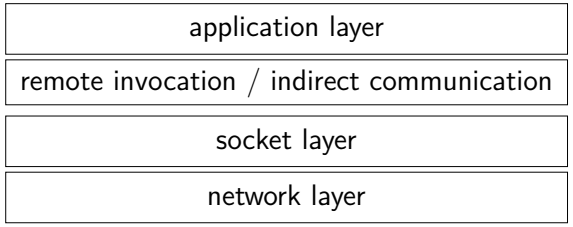
Remote invocation

Johan Montelius

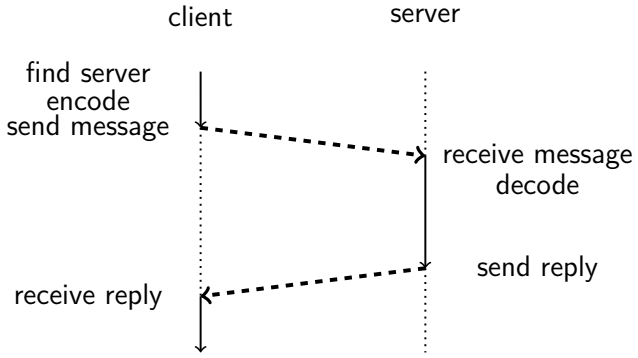
KTH

HT15

middleware

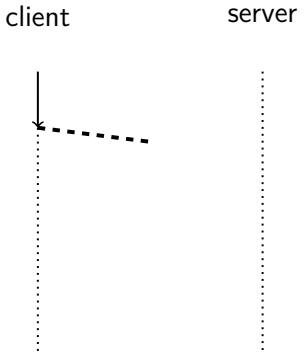


Request reply



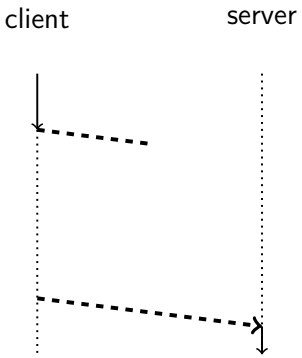
- identify and locate the server
- encode/decode the message
- send reply to the right client
- attach reply to request

lost request



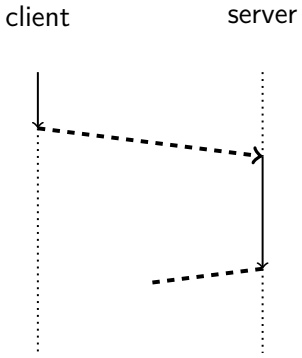
What do we do if request is lost?

resend request



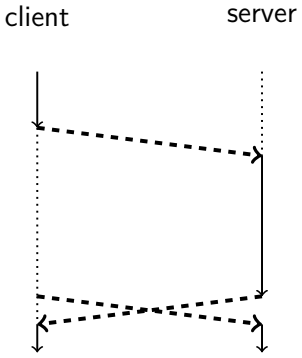
- need to detect that message is potentially lost
- wait for a timeout (how long) or error from underlying layer
- resend the request
- simple, problem solved

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!

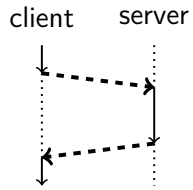
history

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

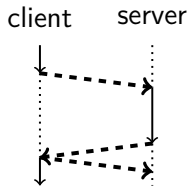
Keep a history of all request and the replies. If a request is resent the same reply can be sent without re-execution.

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 at most once.

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

at most or at least once

What about errors:

If an operation fails/is lost, then

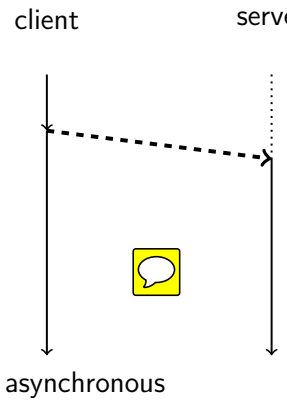
at-most-once:

at-least-once: 

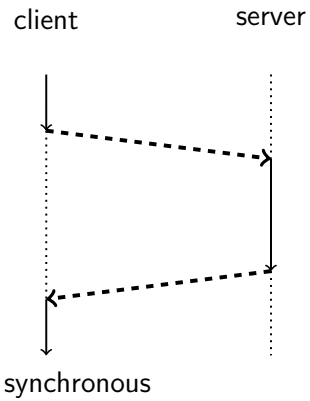
UDP or TCP

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

synchronous/asynchronous

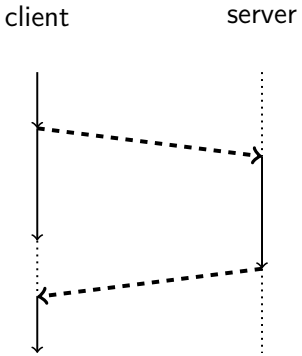


asynchronous



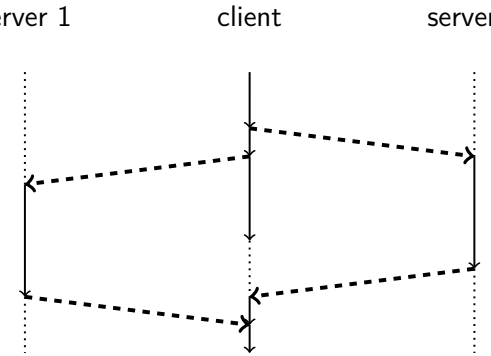
synchronous

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.

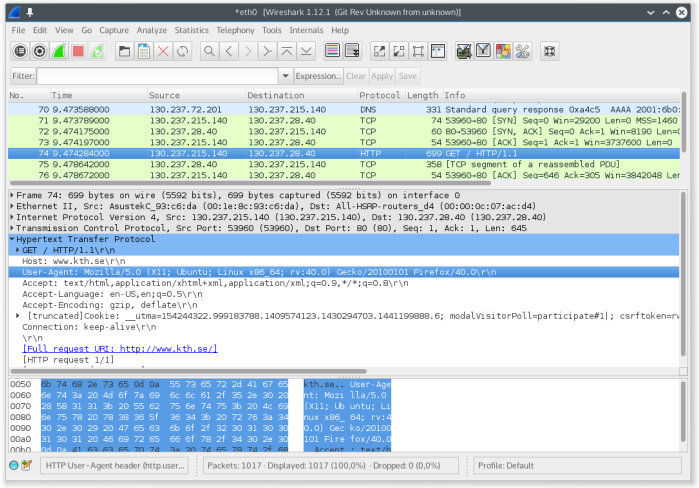
```
Request      = Request-Line *(header CRLF) CRLF [ message-body ]

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



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];
```


operational semantics

```
void proc(int x[], y[]) {
        x[0] = x[0]+1;
        y[0] = y[0] + 1;
}
```

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

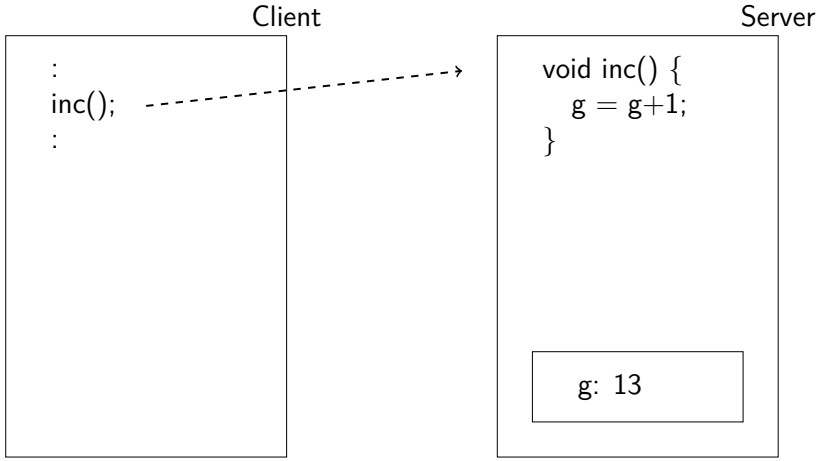
call by value/reference

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

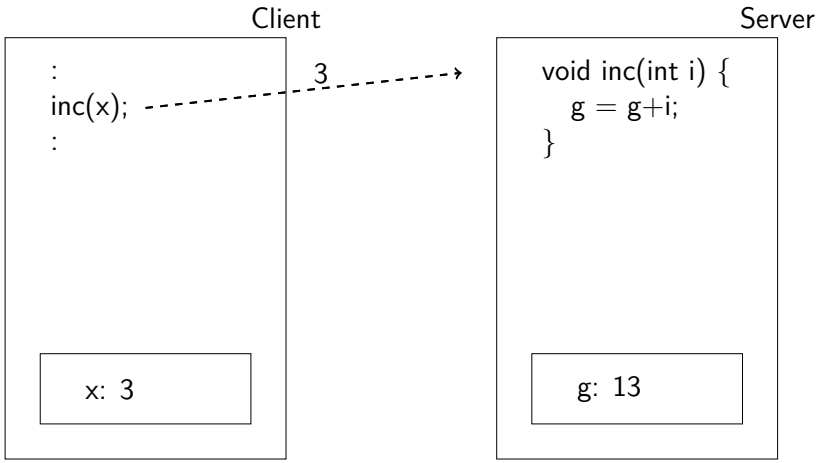
what if the datum is a reference and we pass a copy of the datum

why is this important?

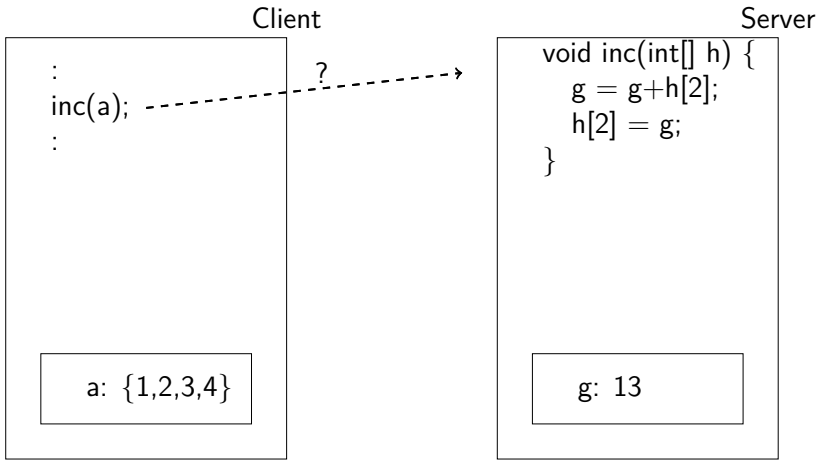
remote procedure call



remote procedure call



remote procedure call



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.