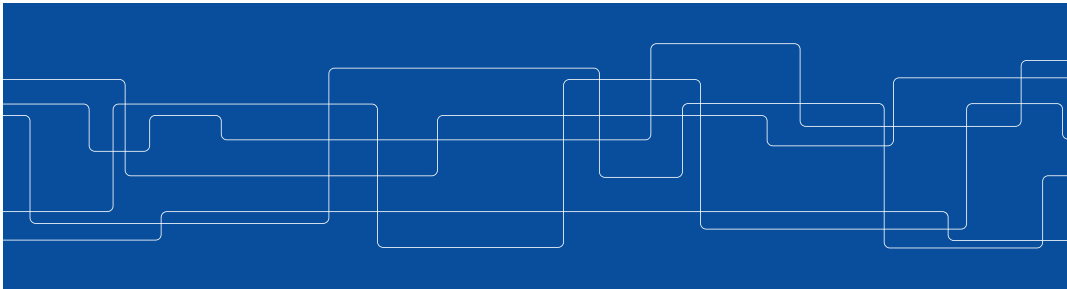


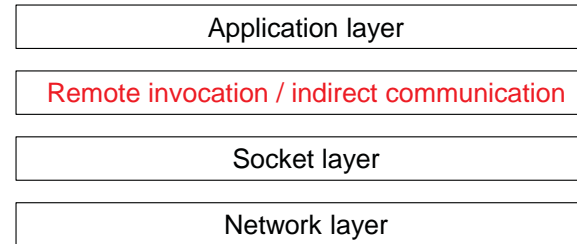


Remote Invocation

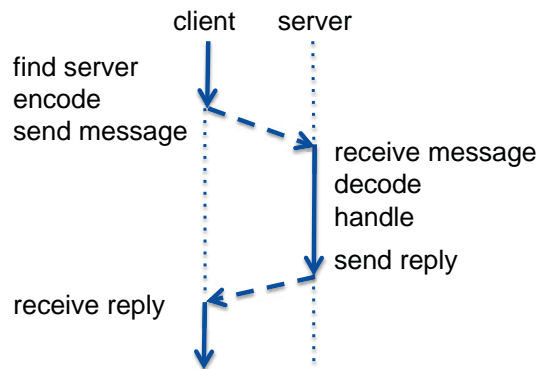
Vladimir Vlassov and Johan Montelius



Middleware



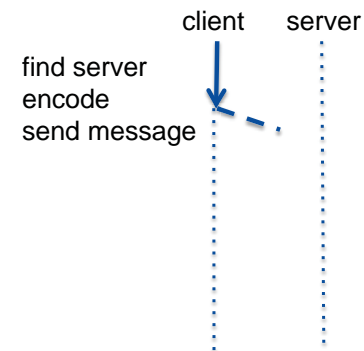
Request / Reply



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



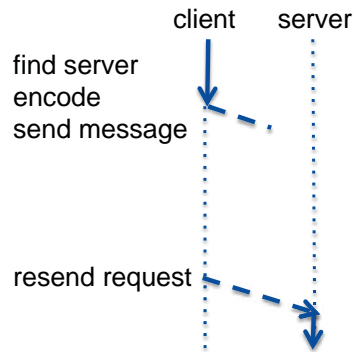
Lost request



What do we do if **request is lost**?



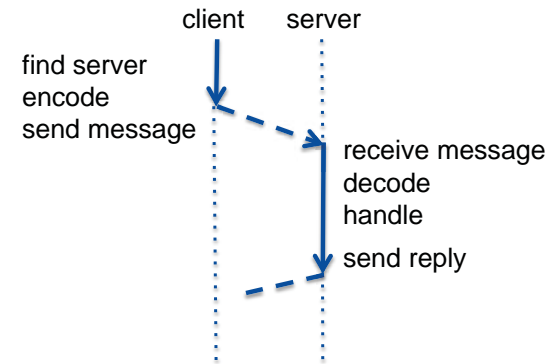
Resend request



- 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



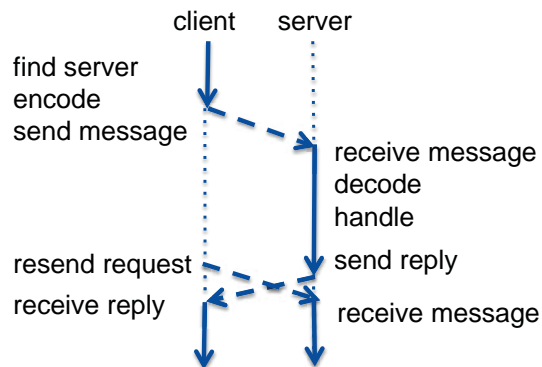
Lost reply



- 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

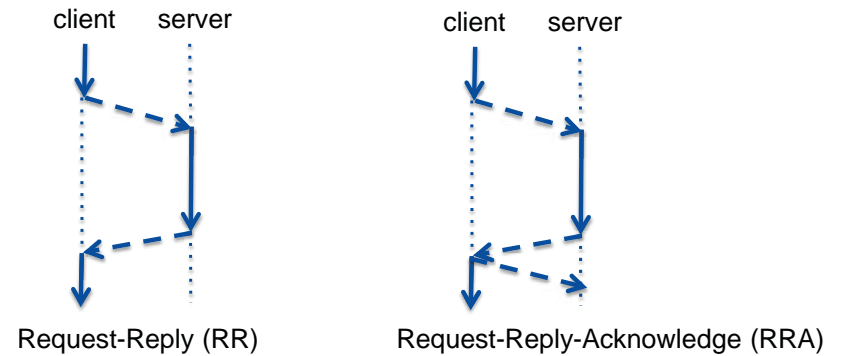
If operations are **not idempotent**, the server must make sure that the same request is **not executed twice**.

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



At-most-once 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 re-sending 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

How about **errors**:

Even if we do resend messages we will have to give 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 re-sending 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?

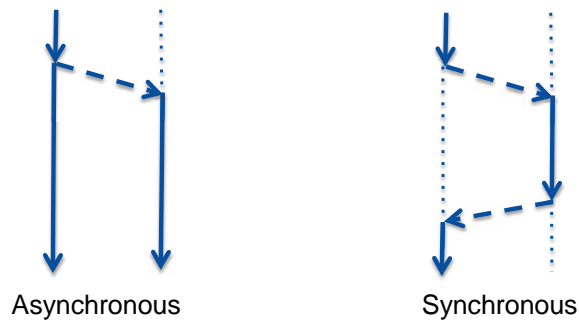


UDP or TCP

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



Synchronous or Asynchronous

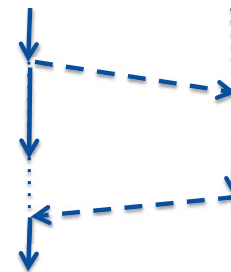


Asynchronous

Synchronous

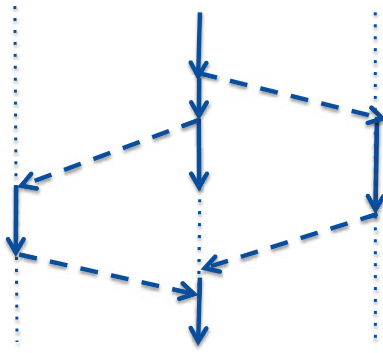


RR over Asynchronous



- 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]

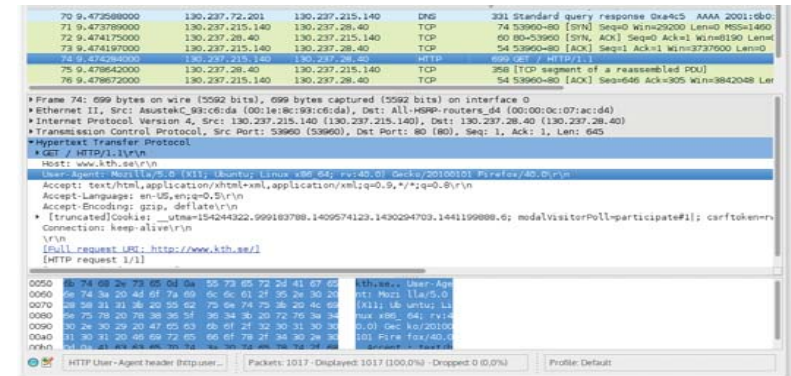
Request-Line = Method SP Request-URI SP HTTP-Version CRLF

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/20100101
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
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 is 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**?



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

- A procedure is given a copy of the datum

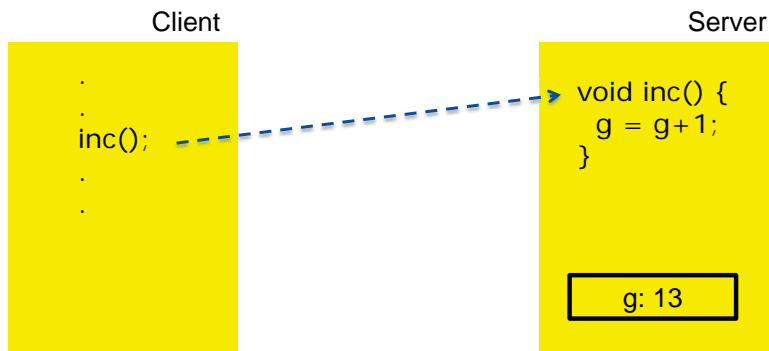
Call by reference

- A procedure is given a reference to the datum

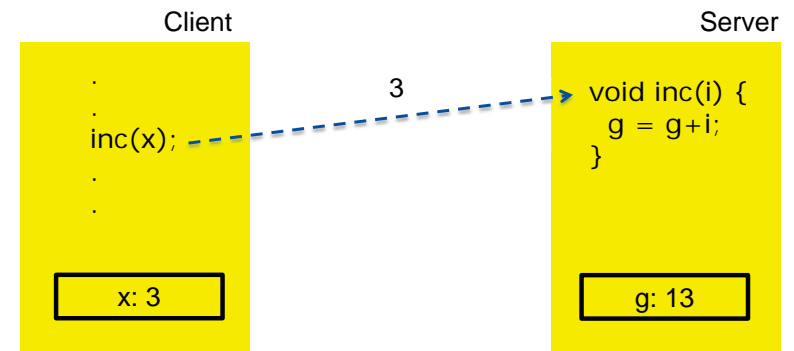
What if the datum is a reference and we pass a copy of the datum?
Why is this important?



RPC: Remote Procedure Call

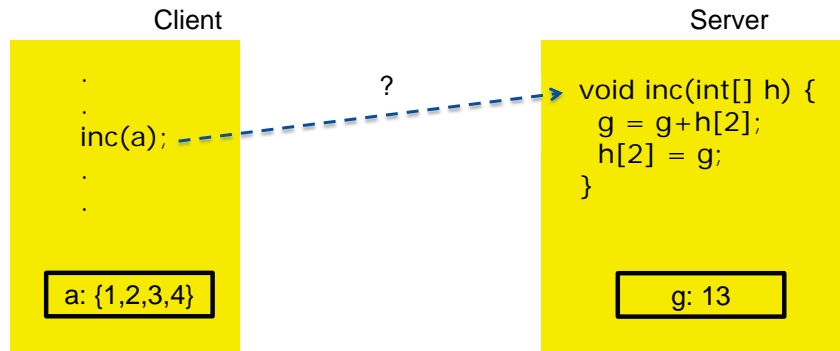


RPC: Remote Procedure Call





RPC: Remote Procedure Call



Open Network Computing (ONC) RPC (SunRPC)

- targeting intranet, file servers etc
- **at-least-once** call semantics
- procedures described in **Interface Definition Language (IDL)**
- XDR (eXternal Data Representation) specifies message structure
- used UDP as transport protocol (TCP also available)



Java RMI (Remote Method Invocation)

- 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:

A **remote object** is passed as a reference (**by reference**) i.e. it remains as at the original place where it was created.

A **serializable object** is passed as a copy (**by value**) i.e. the object is duplicated.



Finding the procedure/object

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-least-once, IDL, XDR, binder
- **JavaRMI**: call-by-value/reference, at-most-once, interface, JRMP (Java Remote Method Protocol), rmiregistry
- **Erlang**: message passing, maybe, no, ETF (External Term Format), local registry only
- **CORBA** (Common Object Request Broker Architecture): call-by-reference, IDL, ORB (Object Request Broker), tnameserv
- **Web Services**: WSDL (Web Services Description Language), UDDI (Universal Description, Discovery, and Integration)



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.