

## CS 5523 Lecture 7

### Request-reply over TCP and UDP

---

- *Questions from Laboratory 1*
- *Discuss questions 3.1, 3.7 and 3.14 of CDK*
- *Introduction to request-reply*
- *HTTP as a request-reply protocol*
- *Request-reply using TCP*
- *Request-reply using UDP*
- *Request-reply-acknowledge using UDP*
- *Idempotent operations*

## Discussion questions from CDK Chapter 3

---

*[CDK 3.1]*

*A client sends a 200-byte request message to a service, which produces a response containing 5000 bytes. Estimate the total time to complete the request in each of the following cases with the performance assumptions listed below:*

- *Using connectionless communication (e.g., UDP);*
  - *Using connection-oriented communication (e.g., TCP);*
  - *The server on the same machine as the client*
- Latency per packet: 5 ms; Connection setup time: 5 ms;*  
*Data transfer rate: 10 Mbps; MTU: 1000 bytes;*  
*Server request processing time: 2 ms*

## Discussion questions from CDK Chapter 3

---

*[CDK 3.7]*

*Compare connectionless (UDP) and connection-oriented (TCP) communication for the implementation of each of the following application-level or presentation-level protocols:*

- i) virtual terminal access (for example, Telnet);*
- ii) file transfer (for example, FTP);*
- iii) user location (for example, rwho, finger);*
- iv) information browsing (for example, HTTP);*
- v) remote procedure call.*

*Aside: what are presentation-level protocols?*

## Discussion questions from CDK Chapter 3

---

*[CDK 3.14]*

*Consider the use of TCP in a telnet remote terminal client. How should the keyboard input be buffered at the client? Investigate Nagle's and Clark's algorithms (Nagle 1984, Clark 1982) for flow control and compare them with the simple algorithm described on page 103 when TCP is used by:*

- *a web server;*
- *a telnet application;*
- *a remote graphical application with continuous mouse input.*

## Request-reply protocols

---

*Most client-server interactions are built on a request-reply protocol:*

*Client:*

- | *makes a request*
- | *waits for reply*

*Server:*

- | *reads request*
- | *processes the request*
- | *sends the reply*

*How does HTTP fit into this protocol?*

Figure 4.15  
HTTP request message

---

<i>method</i>	<i>URL or pathname</i>	<i>HTTP version</i>	<i>headers</i>	<i>message body</i>
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

Figure 4.16  
HTTP reply message

---

<i>HTTP version</i>	<i>status code</i>	<i>reason</i>	<i>headers</i>	<i>message body</i>
HTTP/1.1	200	OK		resource data

## HTTP as a request-reply protocol:

---

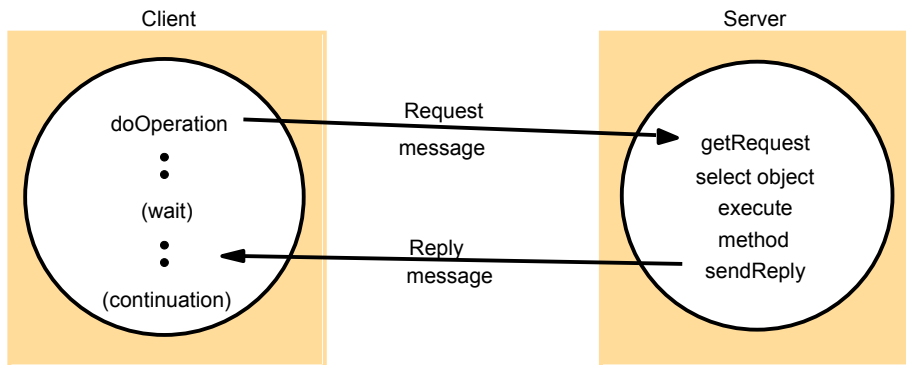
- *requests and replies are “marshalled” into ASCII strings*
- *resources are represented as byte sequences*
- *header lines after the request and response line can convey additional information*
- *data can be prefixed with its MIME type to tell the browser what it is*

## MIME

---

- *MIME stands for Multipurpose Internet Mail Extensions*
- *standard for sending multipart data*
- *data is prefixed by its MIME type so the receiver can handle it*
- *a MIME type consists of type/subtype, e.g., image/jpeg*

Figure 4.11  
Request-reply communication



Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3  
© Addison-Wesley Publishers 2000

## Request-reply as a "remote procedure call"

*Encapsulate the client's interaction with the server into a function:*

*Client (doOperation):*

- | *makes a request*
- | *waits for reply*
- | *return success or error code*

*Server (reads requests and processes them):*

- | *reads request (getRequest)*
- | *processes the request*
- | *sends the reply (sendReply)*

*Draw separate state diagrams for the client and server – states are the steps for each – useful representation for your program*

## Request-reply over TCP:

---

- *TCP has integrity (information is not corrupted)*
- *Information never arrives out-of-order*
- *TCP cannot guarantee delivery, but is usually considered to be a “reliable” protocol*
- *TCP cannot distinguish between a network failure and a process failure*
- *Communicating processes can not tell whether their recent messages have been received*
- *The receiver does not receive the same message twice, even if there were multiple retries at the TCP level*

*Look request-reply over TCP using the state diagram*

## “Reliable” communication:

---

- *Validity - any message in the outgoing message buffer is eventually delivered to the incoming message buffer*
- *Integrity - the received message is identical to the sent message and is delivered exactly once.*

*How does TCP provide integrity?*

*How does TCP make a best effort at validity?*

*How does TCP recognize that a connection has failed?*

*What kind of failures can request-reply have over TCP?*

Figure 2.11 (CDK)  
Omission and arbitrary failures

---

<i>Class of failure</i>	<i>Affects</i>	<i>Description</i>
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a <i>send</i> , but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

---

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3  
© Addison-Wesley Publishers 2000

## Request-reply over UDP:

---

- *UDP has integrity (information is not corrupted)*
- *Information may arrive out-of-order*
- *UDP is usually considered to be an "unreliable" protocol.*
- *UDP does not detect failures – the application must use timeouts with resend to*
- *Resent requests may be received multiple times by receiver*

*Look at the behavior of request-reply over UDP using the state diagram. How is the behavior different from TCP?*

## Request-reply over UDP

---

*Encapsulate the client's interaction with the server into a function:*

*Client (doOperation):*

- *makes a request*
- *waits for reply*
- *return success or error code*

*Server (reads requests and processes them):*

- *reads request (getRequest)*
- *processes the request*
- *sends the reply (sendReply)*

*Draw separate state diagrams for the client and server – states are the steps for each – useful representation for your program*

## Strategies for providing delivery guarantees:

---

- *Retry the request message*
- *Filter duplicate messages at the server*
- *Keep a history of results so that if duplicate requests are received the results can be retransmitted*

## Idempotent operations:

---

- *An idempotent information can be performed repeatedly with same effect.*
- *The HTTP 1.0 GET is idempotent*
- *Idempotent operations don't maintain state on the server*

*How would you make a file server based on idempotent operations?*

*Is writing a block at a specific offset an idempotent operation in UNIX?*

## Request-reply-acknowledge over UDP

---

*Client must:*

- *Use timeouts and retries at each step*
- *Keep sequence numbers*

*Server must:*

- *Maintain status of execution of all non-idempotent operations until it receives an acknowledge*

*Draw a state diagram of client and of server. What is the consequence of a connectionless protocol?*

## Multicast communication:

---

- *Multicast – sends a single message from one process to members of a group of processes (hosts)*
- *Broadcast – sends a single message from one process to all processes (hosts)*

*Why is the word hosts in parentheses?*

*What issues have to be addressed when using multicast?*

## Uses of multicast:

---

- *Fault tolerance based on replicated services*
- *Discovery in spontaneous networking*
- *Managing replicated data*
- *Propagation of event notifications in a distributed environment*

Figure 4.17  
Multicast peer joins a group and sends and receives datagrams

---

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
    public static void main(String args[]){
        // args give message contents & destination multicast group (e.g. "228.5.6.7")
        try {
            InetAddress group = InetAddress.getByName(args[1]);
            MulticastSocket s = new MulticastSocket(6789);
            s.joinGroup(group);
            byte [] m = args[0].getBytes();
            DatagramPacket messageOut =
                new DatagramPacket(m, m.length, group, 6789);
            s.send(messageOut);

            // this figure continued on the next slide
        }
    }
}
```

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3  
© Addison-Wesley Publishers 2000

Figure 4.17  
continued

---

```
        // get messages from others in group
        byte[] buffer = new byte[1000];
        for(int i=0; i< 3; i++) {
            DatagramPacket messageIn =
                new DatagramPacket(buffer, buffer.length);
            s.receive(messageIn);
            System.out.println("Received:" + new String(messageIn.getData()));
        }
        s.leaveGroup(group);
    } catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
    } catch (IOException e){System.out.println("IO: " + e.getMessage());}
}
}
```

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3  
© Addison-Wesley Publishers 2000

## For next time:

---

*Read CDK Section 4.3 – 4.5*

*Read CDK Chapter 5.1*

*Be prepared to discuss CDK exercises 4.3, 4.10, 4.11, 4.12*