Communication in Distributed Systems

Distributed Systems L-A

Sistemi Distribuiti L-A

Andrea Omicini

andrea.omicini@unibo.it

Ingegneria Due

Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008

Outline

These Slides Contain Material from [Tanenbaum and van Steen, 2007]

Slides were made kindly available by the authors of the book

- Such slides shortly introduced the topics developed in the book [Tanenbaum and van Steen, 2007] adopted here as the main book of the course
- Most of the material from those slides has been re-used in the following, and integrated with new material according to the personal view of the teacher of this course
- Every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of the teacher of this course

Outline

What You Are Supposed to Know...

... from the Courses of Computer Networks, Telecommunication Networks and Foundations of Informatics

Basics about protocols

- ISO/OSI
- Protocols and reference model
- Main network and Internet protocols

Basics about communication

- Procedure call
- Representation formats and problems – e.g., little endian vs. big endian
- Sockets

What You Are Supposed to Know...

... from the Courses of Computer Networks, Telecommunication Networks and Foundations of Informatics

Basics about protocols

- ISO/OSI
- Protocols and reference model
- Main network and Internet protocols

Basics about communication

- Procedure call
- Representation formats and problems – e.g., little endian vs. big endian
- Sockets

Outline

The Role of Interaction in Distributed System

Interaction vs. Computation

- Talking of processes, threads, LWP, and the like, is just half of the story
- Maybe, not even the most important half...

→ They represent the *computational* components of a (distributed) system

- Components of a system actually make a system only by interacting with each other

→ *Interaction* represents an orthogonal dimension with respect to *computation*

Engineering Interaction

- Methodologies and technologies for engineering communication are not the same as those for engineering computation
- New models and tools are required
- which could be seamlessly integrated with those for engineering computational components

The Role of Interaction in Distributed System

Interaction vs. Computation

- Talking of processes, threads, LWP, and the like, is just half of the story
- Maybe, not even the most important half...

→ They represent the *computational* components of a (distributed) system

- Components of a system actually make a system only by interacting with each other

→ *Interaction* represents an orthogonal dimension with respect to *computation*

Engineering Interaction

- Methodologies and technologies for engineering communication are not the same as those for engineering computation
- New models and tools are required
- which could be seamlessly integrated with those for engineering computational components

Interaction vs. Communication

Interaction is more general than communication

- Communication is a form of interaction
- Communication is interaction where information is exchanged
- Not every interaction is communication
- E.g., sharing the same space is a way of interacting without communicating

Whereas such a distinction is not always evident from the literature...

- On the one hand, we should keep this in mind
- On the other hand, in the classical field of inter-process communication, this distinction is often not essential

Interaction vs. Communication

Interaction is more general than communication

- Communication is a form of interaction
- Communication is interaction where information is exchanged
- Not every interaction is communication
- E.g., sharing the same space is a way of interacting without communicating

Whereas such a distinction is not always evident from the literature...

- On the one hand, we should keep this in mind
- On the other hand, in the classical field of inter-process communication, this distinction is often not essential

Communication in Non-distributed Settings

Communication does not belong to distributed systems only

- Communication mechanisms like procedure call and message-passing just require a plurality of interacting entities, not distributed ones
- However, communication in distributed systems presents more difficult challenges, like unreliability of communication and large scale
- Of course, communication in distributed systems first of all deals with distribution / location transparency

Outline

Outline

Layered Communication

Communication involves a number of problems at many different levels

- From the physical network level up to the application level
- Communication can be organised on layers
- A *reference model* is useful to understand how protocols, behaviours and interactions

OSI model

- Standardised by the International Standards Organization (ISO)
- Designed to allow open systems to communicate
- Rules for communication govern the format, content and meaning of messages sent and received
- Such rules are formalised in *protocols*
- The collection of protocols for a particular system is its *protocol stack*, or *protocol suite*

Layered Communication

Communication involves a number of problems at many different levels

- From the physical network level up to the application level
- Communication can be organised on layers
- A *reference model* is useful to understand how protocols, behaviours and interactions

OSI model

- Standardised by the International Standards Organization (ISO)
- Designed to allow open systems to communicate
- Rules for communication govern the format, content and meaning of messages sent and received
- Such rules are formalised in *protocols*
- The collection of protocols for a particular system is its *protocol stack*, or *protocol suite*

Types of Protocols

Connection-oriented protocols

- First of all, a connection is established between the sender and the receiver
- Possibly, an agreement over the protocol to be used is reached
- Then, communication occurs through the connection
- Finally, the connection is terminated

Connectionless protocols

- No setup is required
- The sender just send a message when it is ready

Types of Protocols

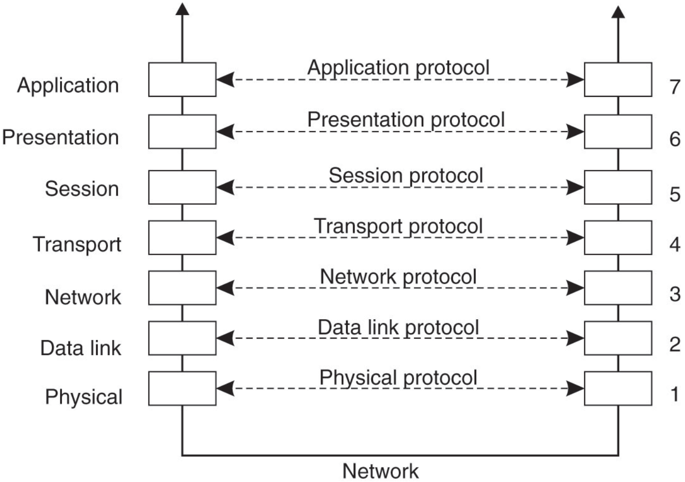
Connection-oriented protocols

- First of all, a connection is established between the sender and the receiver
- Possibly, an agreement over the protocol to be used is reached
- Then, communication occurs through the connection
- Finally, the connection is terminated

Connectionless protocols

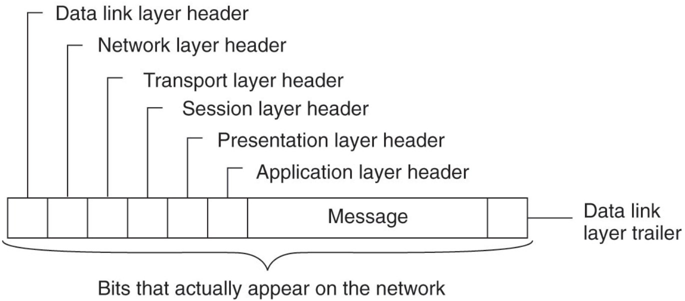
- No setup is required
- The sender just send a message when it is ready

The OSI Reference Model



Layers, interfaces, and protocols in the OSI Model
[Tanenbaum and van Steen, 2007]

A Message in the OSI Reference Model



A typical message as it appears on the network [Tanenbaum and van Steen, 2007]

OSI Model ≠ OSI Protocols

OSI protocols

- Never successful
- TCP/IP is not an OSI protocol, and still dominates its layers

OSI model

- Perfect to understand and describe communication systems through layers
- However, some problems exist when middleware comes to play

OSI Model $\neq$ OSI Protocols

OSI protocols

- Never successful
- TCP/IP is not an OSI protocol, and still dominates its layers

OSI model

- Perfect to understand and describe communication systems through layers
- However, some problems exist when middleware comes to play

Middleware Protocols

The problem

- Middleware mostly lives at the application level
- Protocols for middleware services are different from high-level application protocols

← Middleware protocols are application-independent, application protocols are obviously application-dependent

- How can we distinguish between the two sorts of protocols at the same layer?

Extending the reference model for middleware

- Session and presentation layers are replaced by a *middleware layer*, which includes all application-independent protocols
- Potentially, also the transport layer could be offered in the middleware one

Middleware Protocols

The problem

- Middleware mostly lives at the application level
- Protocols for middleware services are different from high-level application protocols

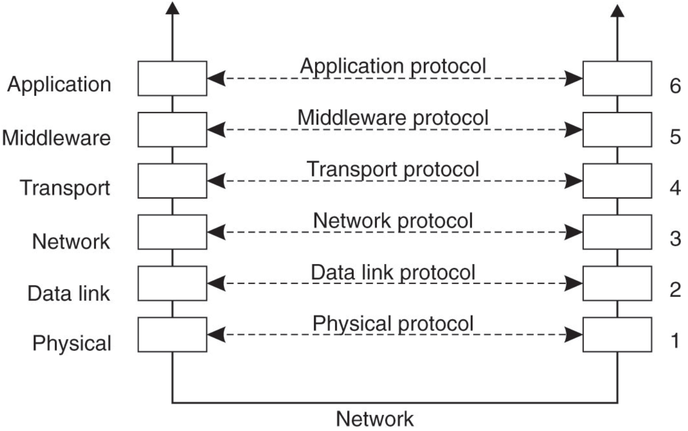
← Middleware protocols are application-independent, application protocols are obviously application-dependent

- How can we distinguish between the two sorts of protocols at the same layer?

Extending the reference model for middleware

- Session and presentation layers are replaced by a *middleware layer*, which includes all application-independent protocols
- Potentially, also the transport layer could be offered in the middleware one

Middleware as an Additional Service in Client-Server Computing



Adapted reference model for network communication [Tanenbaum and van Steen, 2007]

Outline

Types of Communication

Persistent vs. transient communication

- *Persistent communication* — A message sent is stored by the communication middleware until it is delivered to the receiver

→ No need for time coupling between the sender and the receiver

- *Transient communication* — A message sent is stored by the communication middleware only as long as both the receiver and the sender are executing

→ Time coupling between the sender and the receiver

Asynchronous vs. synchronous communication

- *Asynchronous communication* — The sender keeps on executing after sending a message

→ The message should be stored by the middleware

- *Synchronous communication* — The sender blocks execution after sending a message and waits for response – until the middleware acknowledges trasmission, or, until the receiver acknowledges the reception, or, until the receiver has completed processing the request

→ Some form of coupling in control between the sender and the receiver

Types of Communication

Persistent vs. transient communication

- *Persistent communication* — A message sent is stored by the communication middleware until it is delivered to the receiver

→ No need for time coupling between the sender and the receiver

- *Transient communication* — A message sent is stored by the communication middleware only as long as both the receiver and the sender are executing

→ Time coupling between the sender and the receiver

Asynchronous vs. synchronous communication

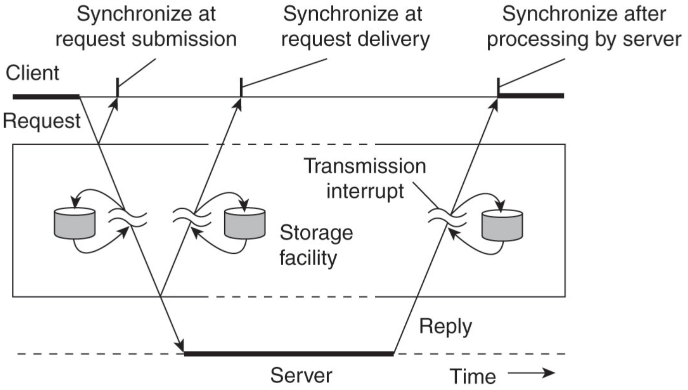
- *Asynchronous communication* — The sender keeps on executing after sending a message

→ The message should be stored by the middleware

- *Synchronous communication* — The sender blocks execution after sending a message and waits for response – until the middleware acknowledges trasmission, or, until the receiver acknowledges the reception, or, until the receiver has completed processing the request

→ Some form of coupling in control between the sender and the receiver

Communications with a Middleware Layer



Viewing middleware as an intermediate (distributed) service in application-level communication [Tanenbaum and van Steen, 2007]

Actual Communication in Distributed Systems

Persistency & synchronisation in communication

- In the practice of distributed systems, many combinations of persistency and synchronisation are typically adopted
- Persistency and synchronisation should then be taken as two dimensions along which communication and protocols could be analysed and classified

Discrete vs. streaming communication

- Communication is not always *discrete*, that is, it does not always happen through complete units of information – e.g., messages
- *Discrete communication* is then quite common, but not the only way available – and does not respond to all the needs
- Sometimes, communication needs to be continuous – through sequences of messages constituting a possibly unlimited amount of information
- *Streaming communication* — The sender delivers a (either limited or unlimited) sequence of messages representing the *stream* of information to be sent to the receiver

→ Communication may be *continuous*

Actual Communication in Distributed Systems

Persistency & synchronisation in communication

- In the practice of distributed systems, many combinations of persistency and synchronisation are typically adopted
- Persistency and synchronisation should then be taken as two dimensions along which communication and protocols could be analysed and classified

Discrete vs. streaming communication

- Communication is not always *discrete*, that is, it does not always happen through complete units of information – e.g., messages
- *Discrete communication* is then quite common, but not the only way available – and does not respond to all the needs
- Sometimes, communication needs to be continuous – through sequences of messages constituting a possibly unlimited amount of information
- *Streaming communication* — The sender delivers a (either limited or unlimited) sequence of messages representing the *stream* of information to be sent to the receiver

→ Communication may be *continuous*

Outline

Remote Procedure Call (RPC)

Basic idea

- Programs can call procedures on other machines
- When a process *A* calls a procedure on a machine *B*, *A* is suspended, and execution of procedure takes place on *B*
- Once the procedure execution has been completed, its completion is sent back to *A*, which resumes execution

Information in RPC

- Information is not sent directly from sender to receiver
- Parameters are just packed and transmitted along with the request
- Procedure results are sent back with the completion
- *No message passing*

Remote Procedure Call (RPC)

Basic idea

- Programs can call procedures on other machines
- When a process *A* calls a procedure on a machine *B*, *A* is suspended, and execution of procedure takes place on *B*
- Once the procedure execution has been completed, its completion is sent back to *A*, which resumes execution

Information in RPC

- Information is not sent directly from sender to receiver
- Parameters are just packed and transmitted along with the request
- Procedure results are sent back with the completion
- *No message passing*

Issues of RPC

Main problems

- The address space of the caller and the callee are separate and different

$\longrightarrow$ Need for a common reference space

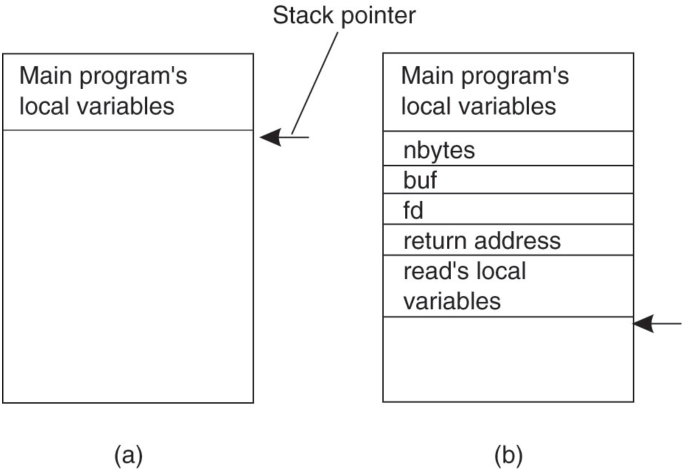
- Parameters and results have to be passed and handled correctly

$\longrightarrow$ Need for a common data format

- Either / both machines could unexpectedly crash

$\longrightarrow$ Need for suitable fault-tolerance policies

Conventional Procedure Call



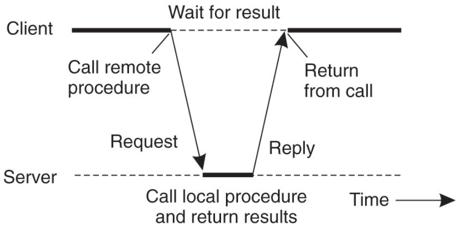
Parameter passing in a local procedure call
[Tanenbaum and van Steen, 2007]

Client & Server Stubs

Main goal: transparency

- RPC should be like local procedure call from the viewpoint of both the caller and the callee

→ Procedure calls are sent to the *client stub* and transmitted to the *server stub* through the network to the called procedure



Principle of RPC between a client and server program [Tanenbaum and van Steen, 2007]

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Steps for a RPC

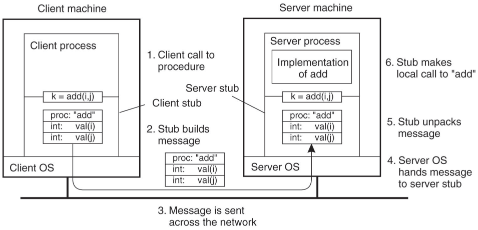
- The client procedure calls the client stub in the normal way
- The client stub builds a message and calls the local operating system
- The client's OS sends the message to the remote OS
- The remote OS gives the message to the server stub
- The server stub unpacks the parameters and calls the server
- The server does the work and returns the result to the stub
- The server stub packs it in a message and calls its local OS
- The server's OS sends the message to the client's OS
- The client's OS gives the message to the client stub
- The stub unpacks the result and returns to the client

Parameter Passing

Passing value parameters

- Parameters are *marshalled* to pass across the network

→ Procedure calls are sent to the *client stub* and transmitted to the *server stub* through the network to the called procedure



Steps of a remote computation through a RPC [Tanenbaum and van Steen, 2007]

Issues in Parameter Passing

Passing value parameters

- Problems of representation and meaning
- E.g., little endian vs. big endian
- In order to ensure transparency, stubs should be in charge of the mapping & translation
- Possible approach: interfaces described through and IDL (Interface Definition Languange), and consequent handling compiled into the stubs

Passing reference parameters

- Main problem: reference space is local
- First solution: forbidding reference parameters
- Second solution: copying parameters (suitably updating the reference), then copying them back (according to the original reference)

→ Call-by-reference becomes copy&restore

- Third solution: creating a global/accessible reference to the caller space from the callee

Issues in Parameter Passing

Passing value parameters

- Problems of representation and meaning
- E.g., little endian vs. big endian
- In order to ensure transparency, stubs should be in charge of the mapping & translation
- Possible approach: interfaces described through and IDL (Interface Definition Languange), and consequent handling compiled into the stubs

Passing reference parameters

- Main problem: reference space is local
- First solution: forbidding reference parameters
- Second solution: copying parameters (suitably updating the reference), then copying them back (according to the original reference)

→ Call-by-reference becomes copy&restore

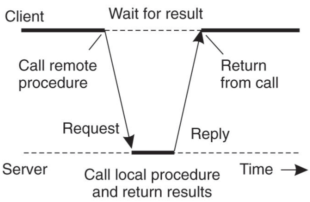
- Third solution: creating a global/accessible reference to the caller space from the callee

Asynchronous RPC

Synchronicity might be a problem in distributed systems

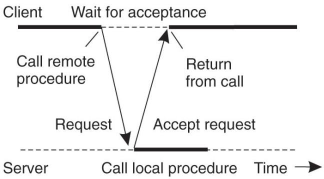
- Synchronicity is often unnecessary, and may create problems

→ *Asynchronous RPC* is an available alternative in many situations



(a)

Traditional RPC



(b)

Asynchronous RPC

[Tanenbaum and van Steen, 2007]

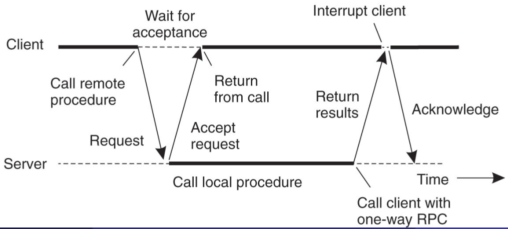
Deferred Synchronous RPC

Combining asynchronous RPCs

- Sometimes some synchronicity is required, but too much is too much

→ *Deferred Synchronous RPC* combines two asynchronous RPC to provide an *ad hoc* form of synchronicity

- The first asynchronous call selects the procedure to be executed and provides for the parameters
- The second asynchronous call goes for the results
- In between, the caller may keep on computing



Limits of RPC

Coupling in time

- Co-existence in time is a requirement for any RPC mechanism
- Sometimes, a too-hard requirement for effective communication in distributed systems
- An alternative is required that does not require the received to be executing when the message is sent

The alternative: messaging

- Please notice: message-oriented communication is not synonym of uncoupling
- However, we can take this road toward uncoupled communication

Limits of RPC

Coupling in time

- Co-existence in time is a requirement for any RPC mechanism
- Sometimes, a too-hard requirement for effective communication in distributed systems
- An alternative is required that does not require the received to be executing when the message is sent

The alternative: messaging

- Please notice: message-oriented communication is not synonym of uncoupling
- However, we can take this road toward uncoupled communication

Outline

Message-oriented Transient Communication

Basic idea

- Messages are sent through a channel abstraction
- The channel connects two running processes
- Time coupling between sender and receiver
- Transmission time is measured in terms of milliseconds, typically

Examples

- Berkeley Sockets — typical in TCP/IP-based networks
- MPI (Message-Passing Interface) — typical in high-speed interconnection networks among parallel processes

Message-oriented Transient Communication

Basic idea

- Messages are sent through a channel abstraction
- The channel connects two running processes
- Time coupling between sender and receiver
- Transmission time is measured in terms of milliseconds, typically

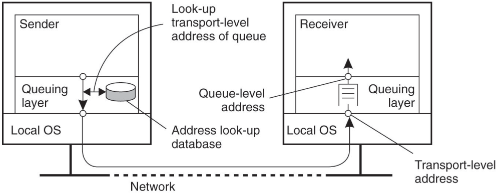
Examples

- Berkeley Sockets — typical in TCP/IP-based networks
- MPI (Message-Passing Interface) — typical in high-speed interconnection networks among parallel processes

Message-Oriented Persistent Communication

Message-queuing systems – a.k.a. Message-Oriented Middleware (MOM)

- Basic idea: MOM provides message storage service
- A message is put in a queue by the sender, and delivered to a destination queue
- The target(s) can retrieve their messages from the queue
- Time uncoupling between sender and receiver
- Example: IBM's WebSphere



General architecture of a message-queuing system
[Tanenbaum and van Steen, 2007]

Outline

Streams

Sequences of data

- A stream is transmitted by sending sequences of related messages
- Single vs. complex streams: a single sequence vs. several related simple streams
- Data streams: typically, streams are used to represent and transmit huge amounts of data
- Examples: JPEG images, MPEG movies

Streams & Time

Continuous vs. discrete media

- In the case of *continuous (representation) media*, time is relevant to understand the data – e.g., audio streams
- In the case of *discrete (representation) media*, time is not relevant to understand the data – e.g., still images

Transmission of time-dependent information

Asynchronous transmission mode
: data items of a stream are transmitted in sequence without further constraints—e.g., a file representing a still image

Synchronous transmission mode
: data items of a stream are transmitted in sequence with a maximum end-to-end delay—e.g., data generation by a pro-active sensor

Isochronous transmission mode
: data items of a stream are transmitted in sequence with both a maximum and a minimum end-to-end delay—e.g., audio & video

Streams & Time

Continuous vs. discrete media

- In the case of *continuous (representation) media*, time is relevant to understand the data – e.g., audio streams
- In the case of *discrete (representation) media*, time is not relevant to understand the data – e.g., still images

Transmission of time-dependent information

Asynchronous transmission mode data items of a stream are transmitted in sequence without further constraints—e.g., a file representing a still image

Synchronous transmission mode data items of a stream are transmitted in sequence with a maximum end-to-end delay—e.g., data generation by a pro-active sensor

Isochronous transmission mode data items of a stream are transmitted in sequence with both a maximum and a minimum end-to-end delay—e.g., audio & video

Streams & Quality of Service

Quality of service

- Timing and other non-functional properties are typically expressed as *Quality of Service* (QoS) requirements
- In the case of streams, QoS typically concerns *timing*, *volume*, and *reliability*
- In the case of middleware, the issue is how can a given middleware ensure QoS to distributed applications

A practical problem

- Whatever the theory, many distributed systems providing streaming services rely on top of the IP stack
- IP specification allow for a protocol implementation dropping packets when needed
- QoS should be enforced at the higher levels

Streams & Quality of Service

Quality of service

- Timing and other non-functional properties are typically expressed as *Quality of Service* (QoS) requirements
- In the case of streams, QoS typically concerns *timing*, *volume*, and *reliability*
- In the case of middleware, the issue is how can a given middleware ensure QoS to distributed applications

A practical problem

- Whatever the theory, many distributed systems providing streaming services rely on top of the IP stack
- IP specification allow for a protocol implementation dropping packets when needed
- QoS should be enforced at the higher levels

Summing Up

Interaction & communication

- Interaction as an orthogonal dimension w.r.t. computation
- Communication as a form of interaction

High-level abstractions for process-level communication

- Remote Procedure Call
- Message-oriented models
- Streaming
- Other forms like multicasting and epidemic protocols are important, but are not a subject for this course

Summing Up

Interaction & communication

- Interaction as an orthogonal dimension w.r.t. computation
- Communication as a form of interaction

High-level abstractions for process-level communication

- Remote Procedure Call
- Message-oriented models
- Streaming
- Other forms like multicasting and epidemic protocols are important, but are not a subject for this course

Bibliography

Tanenbaum, A. S. and van Steen, M. (2007).
Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

Communication in Distributed Systems

Distributed Systems L-A
Sistemi Distribuiti L-A

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008