CS 755 – System and Network Architectures and Implementation

Module 4 – Remote Services

Martin Karsten

mkarsten@uwaterloo.ca
Notice

Some figures are taken from third-party slide sets. In this module, parts are taken from the Kurose/Ross and the Tanenbaum/van Steen slide set. See details on next slides...
A note on the use of these ppt slides:

We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2009
J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 4
Communication
Overview

- messaging / message queueing
- remote procedure call
- security
Transport - Review

- multiplexing, virtual channel
  - process-to-process communication
- reliability
- flow and congestion control
- connection management
- participants: online and available!
Communication - Synchronization

- Synchronize at request submission
- Synchronize at request delivery
- Synchronize after processing by server

Client -> Request

Server

Storage facility
Transmission interrupt

Reply

Time
Messaging

- **persistent** communication
  - sender can terminate after sending message
  - receiver does not need to be online
  - vs. **transient** communication
- **asynchronous** communication
  - sender can continue other work after sending
    - vs. sender waits for acknowledgement
  - receiver is notified when message is available
    - vs. receiver blocks waiting for message
Persistency and Synchronization

Sender running

 Sender running

 Sender passive

 Sender passive

Receiver running

 Receiver passive

 Receiver running

 Receiver passive

(a)  (b)  (c)  (d)
Messaging Middleware

- persistence – reliability
- management, tracing, availability
- flexible integration with heterogeneous systems
  - OS, network, programming language, etc.
- group communication: publish / subscribe
  - underlying distribution model: unicast vs. broadcast
Messaging Queueing Primitives

- Put – append message to queue (send)
- Get – retrieve message from queue (receive)
- Poll – check queue(s) for message availability
- Notify – install asynchronous retrieve handler

- need buffer decoupled from sender, receiver
- relay nodes for larger networks
  - addressing, routing, forwarding, etc., as usual
Architecture

![Diagram showing the architecture of a network system with layers and addresses.](image-url)
Architecture
Example: Email

mail servers

- incoming messages mailbox
- outgoing message queue
- communication protocol: SMTP
  - reliable server-to-server transfer
Email Access Protocols

- sender: synchronous, transient to server
- receiver: asynchronous, persistent from server
  - Post Office Protocol (POP) – old & simple
  - Internet Mail Access Protocol (IMAP) – better
  - HTTP – POP, IMAP, etc in background
  - remote file system and file-based (elm, pine, etc.)
Advanced Message Queuing Protocol (AMQP)

Applications
Produce
Messages

Exchanges
Route and Filter
Messages

Queues
Store and Forward
Messages

Applications
Consume
Messages

Clients

Server (a.k.a. Broker)

Clients
Message Passing Interface (MPI)

- portable abstraction of socket interface
- weaker semantics than message queueing

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_bsend</td>
<td>Append outgoing message to a local send buffer</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait until copied to local or remote buffer</td>
</tr>
<tr>
<td>MPI_ssend</td>
<td>Send a message and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_sendrecv</td>
<td>Send a message and wait for reply</td>
</tr>
<tr>
<td>MPI_isend</td>
<td>Pass reference to outgoing message, and continue</td>
</tr>
<tr>
<td>MPI_issend</td>
<td>Pass reference to outgoing message, and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_recv</td>
<td>Receive a message; block if there is none</td>
</tr>
<tr>
<td>MPI_irecv</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>
Publish/Subscribe

- special case of messaging
- notion of “queue” replaced by arbitrary filter
  - structured / topic
  - unstructured / content
Remote Procedure Call

- transparent execution of remote functionality

- example: Sun RPC aka ONC RPC

- classic UNIX RPC system
  - developed with/for Network File System (NFS)

- available on most UNIX systems

- see: man rpc
Conventional Procedure Call

int len = read(fd, buf, nbytes);
RPC - Challenges

- machine architecture
- address space
- parameter passing
- independent failures

- goal: transparency
Remote Invocation

Client

Call remote procedure

Wait for result

Return from call

Request

Reply

Server

Call local procedure and return results

Time
RPC Details

1. Client procedure calls *client stub* locally.
2. Client stub builds message and calls local OS.
   - *marshalling*: parameters $\rightarrow$ message
3. Client OS sends message to server OS.
4. Server OS gives message to *server stub*.
5. Server stub unpacks parameters and calls server routine.
   - *de/unmarshalling*: message $\rightarrow$ parameters
RPC Details

6. Server routine executes and returns to stub.
7. Server stub builds message and calls local OS.
8. Server OS sends message to client OS.
9. Client OS gives message to client stub.
10. Client stub unpacks result and returns to client.
RPC Details

1. Client call to procedure
2. Stub builds message
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to "add"
Data Representation

- transparency across platforms
  - Sun RPC: eXtensible Data Representation (XDR)

- hardware architecture
- operating system
- programming language
- runtime environment
Data Representation

- common example: integer representation
  - little endian vs. big endian

- others: float, string, structures...

- dynamic data structures: list, tree, etc.

- objects?
Synchronous RPC

Client

Wait for result

Call remote procedure

Request

Call local procedure and return results

Server

Reply

Return from call

Time →

(a)
Asynchronous RPC

Client  Wait for acceptance

Call remote procedure  Return from call

Request  Accept request

Server  Call local procedure  Time →

(b)
Two-Way Asynchronous RPC

Client

Call remote procedure
Request
Wait for acceptance
Call local procedure

Server

Accept request
Return from call
Return results
Call client with one-way RPC
Time

Interrupt client
Acknowledge
Runtime

1. Register end point
2. Register service
3. Look up server
4. Ask for end point
5. Do RPC

Client machine

Directory machine

Directory server

Server machine

Server

DCE daemon

End point table
Distributed Objects

Client machine

Client invokes a method

Client

Proxy

Same interface as object

Server machine

Server

Object

State

Method

Skeleton

Skeleton invokes same method at object

Server OS

Interface

Network

Marshalled invocation is passed across network

Client OS

CS 755 - Fall 2014
Object References

Machine A
- Local object O1
  - Local reference L1

Machine B
- Remote object O2
  - Remote reference R1

Machine C
- Copy of O1
  - Remote invocation with L1 and R1 as parameters

Client code with RMI to server at C (proxy)

Copy of R1 to O2

Server code (method implementation)
Other RPC-Type Systems

- DCE -> DCOM/ODBC
- CORBA
- Java RMI
- SOAP

- Data Representation: XML
What is network security?

Authentication: sender, receiver want to confirm identity of each other

Confidentiality: only sender, intended receiver should "understand" message contents
- sender encrypts message
- receiver decrypts message

Message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Access and availability: services must be accessible and available to users
Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages
Who might Bob, Alice be?

• … well, *real-life* Bobs and Alices!
• Web browser/server for electronic transactions (e.g., on-line purchases)
• on-line banking client/server
• DNS servers
• routers exchanging routing table updates
• etc…
There are bad guys (and girls) out there!

Q: What can a “bad guy” do?
A: a lot!

- **eavesdrop**: intercept messages
- actively **insert** messages into connection
- **impersonation**: can fake (spoof) source address in packet (or any field in packet)
- **hijacking**: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service**: prevent service from being used by others (e.g., by overloading resources)
The language of cryptography

- **Symmetric key crypto**: Sender, receiver keys *identical*
- **Public-key crypto**: Encryption key *public*, decryption key *secret* (private) – or vice versa

![Diagram showing encryption and decryption processes]
Symmetric key cryptography

**substitution cipher:** substituting one thing for another
- monoalphabetic cipher: substitute one letter for another

plaintext:  abcdefghijklmnopqrstuvwxyz

| ciphertext:  mnbvcxzasdfghjklpoiuytrewq |

**E.g.:**
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc

**Q: How hard to break this simple cipher?:**
- brute force (how hard?)
- ciphertext-only vs known-plaintext vs chosen-plaintext
Symmetric key cryptography

Symmetric key crypto: Bob and Alice share know same (symmetric) key: $K_{A-B}$

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
- **Q:** how do Bob and Alice agree on key value?
Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase ("Strong cryptography makes the world a safer place") decrypted (brute force) in 4 months
  - no known "backdoor" decryption approach
- making DES more secure:
  - use three keys sequentially (3-DES) on each datum
  - use cipher-block chaining
AES: Advanced Encryption Standard

- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES
Public key cryptography

**symmetric key crypto**
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

**public key cryptography**
- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver
Public key cryptography

plaintext message, m

encryption algorithm

K^+(m)

ciphertext

decryption algorithm

m = K^-_B(K^+_B(m))

plaintext message

Bob’s public key

K^+_B

Bob’s private key

K^-_B
Public key encryption algorithms

Requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that $K_B^-(K_B^+(m)) = m$

2. given $K_B^+$, cannot easily compute $K_B^-$

**RSA:** Rivest, Shamir, Adleman algorithm
RSA: another important property

The following property will be very useful later:

\[ K_B(K_B^-(m)) = m = K_B(K_B^+(m)) \]

use public key first, followed by private key
use private key first, followed by public key

Result is the same!
Message Integrity

Bob receives msg from Alice, wants to ensure:
  • message originally came from Alice
  • message not changed since sent by Alice

Cryptographic Hash:
  • takes input m, produces fixed length value, H(m)
    • e.g., as in Internet checksum
  • computationally infeasible to find two different messages, x, y such that H(x) = H(y)
    • equivalently: given m = H(x), (x unknown), can not determine x.
    • note: Internet checksum fails this requirement!
Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

<table>
<thead>
<tr>
<th>message</th>
<th>ASCII format</th>
<th>message</th>
<th>ASCII format</th>
</tr>
</thead>
<tbody>
<tr>
<td>I O U 1</td>
<td>49 4F 55 31</td>
<td>I O U 9</td>
<td>49 4F 55 39</td>
</tr>
<tr>
<td>0 0 . 9</td>
<td>30 30 2E 39</td>
<td>0 0 . 1</td>
<td>30 30 2E 31</td>
</tr>
<tr>
<td>9 B O B</td>
<td>39 42 4F 42</td>
<td>9 B O B</td>
<td>39 42 4F 42</td>
</tr>
</tbody>
</table>

B2 C1 D2 AC  different messages but identical checksums!
Digital Signatures

cryptographic technique analogous to hand-written signatures.

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- **verifiable, nonforgeable**: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
Digital Signatures

simple digital signature for message \(m\):

- Bob “signs” \(m\) by encrypting with his private key \(K_B\), creating “signed” message, \(K_B(m)\)

Bob’s message, \(m\)

Dear Alice
Oh, how I have missed you. I think of you all the time! ...(blah blah blah)
Bob

Bob’s message, \(m\), signed (encrypted) with his private key

Bob’s private key

\(K_B\)

public key encryption algorithm

\(K_B(m)\)
Digital Signatures (more)

- suppose Alice receives msg m, digital signature $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob’s public key $K_B^+$ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- if $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob’s private key.

Alice thus verifies that:
- Bob signed m.
- No one else signed m.
- Bob signed m and not m’.

non-repudiation:
- Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.
Digital signature

Bob sends digitally signed message:

large message m

H: hash function

H(m)

digital signature (encrypt)

encrypted msg digest K_B^-(H(m))

Bob’s private key K_B

Alice verifies signature and integrity of digitally signed message:

large message m

H: hash function

H(m)

digital signature (decrypt)

encrypted msg digest K_B^+(H(m))

Bob’s public key K_B

Bob’s private key K_B

equal ?

CS 755 - Fall 2014
Symmetric vs. Public Key

- symmetric (shared) key
  - less computational overhead
- public/private key
  - easier to set up
- typical compromise
  - both: key “wear-and-tear”, information leakage
  - use private key during session setup
  - negotiate shared key for session duration
**Authentication**

**Goal**: Bob wants Alice to “prove” her identity to him

**Protocol ap1.0**: Alice says “I am Alice”

Failure scenario??
Authentication

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”

in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice

“I am Alice”
Authentication: another try

**Protocol ap2.0:** Alice says “I am Alice” in an IP packet containing her source IP address

<table>
<thead>
<tr>
<th>Alice’s IP address</th>
<th>“I am Alice”</th>
</tr>
</thead>
</table>

Failure scenario??
Authentication: another try

**Protocol ap2.0:** Alice says “I am Alice” in an IP packet containing her source IP address

Trudy can create a packet “spoofing” Alice’s address

| Alice’s IP address | “I am Alice” |
Authentication: another try

**Protocol ap3.0:** Alice says “I am Alice” and sends her secret password to “prove” it.

```
<table>
<thead>
<tr>
<th>Alice’s IP addr</th>
<th>Alice’s password</th>
<th>“I’m Alice”</th>
</tr>
</thead>
</table>
```

Failure scenario??

```
<table>
<thead>
<tr>
<th>Alice’s IP addr</th>
<th>OK</th>
</tr>
</thead>
</table>
```
Authentication: another try

**Protocol ap3.0:** Alice says “I am Alice” and sends her secret password to “prove” it.

**playback attack:** Trudy records Alice’s packet and later plays it back to Bob.
Authentication: yet another try

**Protocol ap3.1:** Alice says “I am Alice” and sends her *encrypted* secret password to “prove” it.

Failure scenario??
Authentication: another try

**Protocol ap3.1:** Alice says “I am Alice” and sends her *encrypted* secret password to “prove” it.
Authentication: yet another try

**Goal:** avoid playback attack

**Nonce:** number (R) used only *once* -in-a-lifetime

**ap4.0:** to prove Alice “live”, Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key

```
K (R)
```

```
A-B
```

“*I am Alice*”

Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

Failures, drawbacks?
Authentication: ap5.0

ap4.0 requires shared symmetric key
• can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography

“I am Alice”

Bob computes
\[ K_A^+ (K_A^- (R)) = R \]
and knows only Alice could have the private key, that encrypted R such that
\[ K_A^+ (K_A^- (R)) = R \]
ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)

I am Alice
\[ R \]
\[ K^-(R) \]
Send me your public key
\[ A \]

I am Alice
\[ K^+(R) \]
Send me your public key
\[ T \]

Trudy gets
\[ T \]
\[ K^+(m) \]
m = \( K^- \) (\( m \)) sends \( m \) to Alice encrypted with Alice’s public key
\[ A \]
\[ A \]

m = \( K^- \) (\( K^+ \) (m))
ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)

Difficult to detect:
- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation)
- problem is that Trudy receives all messages as well!
Public Key Certification

public key problem:
• When Alice obtains Bob’s public key (from web site, e-mail, diskette), how does she *know* it is Bob’s public key, not Trudy’s?

solution:
• trusted certification authority (CA)
Certification Authorities

- **Certification Authority (CA):** binds public key to particular entity, E.
- E registers its public key with CA.
  - E provides “proof of identity” to CA.
  - CA creates certificate binding E to its public key.
  - Certificate containing E’s public key digitally signed by CA: CA says “This is E’s public key.”
Certification Authorities

- when Alice wants Bob’s public key:
  - gets Bob’s certificate (Bob or elsewhere).
  - apply CA’s public key to Bob’s certificate, get Bob’s public key

\[ K_{CA}(K_B^+) \quad \text{digital signature (decrypt)} \quad K_B^+ \]

CA public key

Bob’s public key

CS 755 - Fall 2014 4-71
A certificate contains:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)

- info about certificate issuer
- valid dates
- digital signature by issuer