



Comunicazione in Distribuito

SCD

Anno accademico 2014/15
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

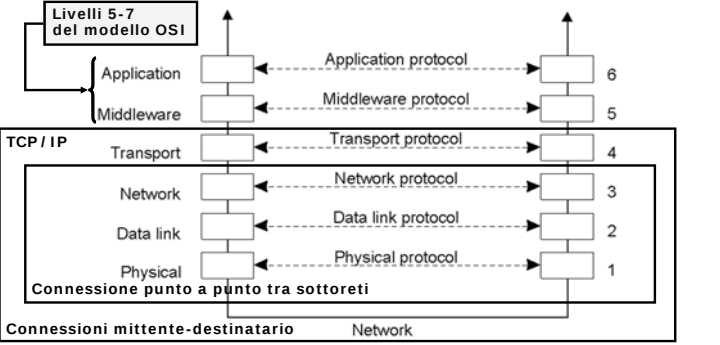
Laurea Magistrale in Informatica, Università di Padova

1 / 43



Sistemi distribuiti: comunicazione

Visione a livelli – 1



Laurea Magistrale in Informatica, Università di Padova

3 / 43



Sistemi distribuiti: comunicazione

Evoluzione di modelli

- ❑ **Remote Procedure Call (RPC)**
 - Trasparente rispetto allo scambio messaggi necessario per supportare l'interazione cliente-servente a livello applicazione
- ❑ **Remote (Object) Method Invocation (RMI)**
 - L'interazione a livello applicazione avviene attraverso oggetti distribuiti
- ❑ **Scambio messaggi a livello middleware**
 - Con paradigmi espliciti a livello applicazione
- ❑ **Stream o comunicazioni a flusso continuo**
 - Flusso di dati che richiedono continuità temporale

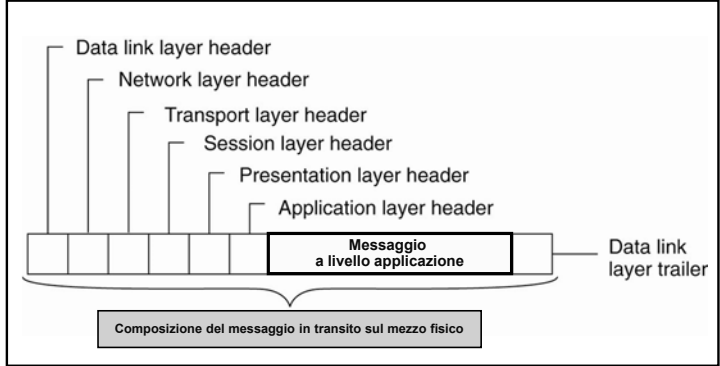
Laurea Magistrale in Informatica, Università di Padova

2 / 43



Sistemi distribuiti: comunicazione

Visione a livelli – 2



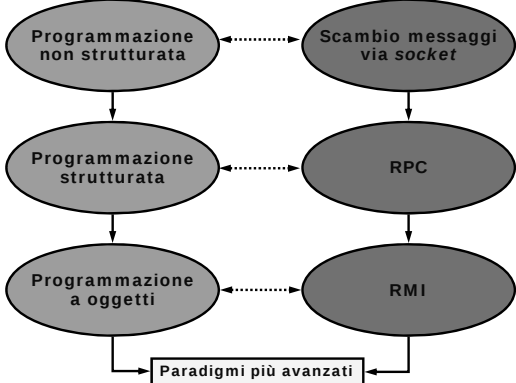
Laurea Magistrale in Informatica, Università di Padova

4 / 43



Sistemi distribuiti: comunicazione

Visione per analogie



Laurea Magistrale in Informatica, Università di Padova

5 / 43



Sistemi distribuiti: comunicazione

RPC – 1

- ❑ Consentire a un processo C residente su un nodo E1 di invocare ed eseguire una procedura P residente su un nodo E2
- ❑ Durante l'invocazione il chiamante viene sospeso
 - I parametri di ingresso viaggiano da chiamante a chiamato
 - I parametri di ritorno viaggiano da chiamato a chiamante
- ❑ Chiamante e chiamato non sono coinvolti nello scambio di messaggi sottostante
 - Trasparenza!

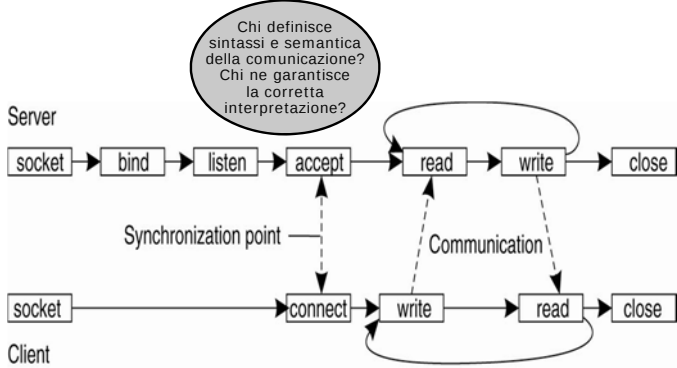
Laurea Magistrale in Informatica, Università di Padova

7 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi



Tratto da: Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc.

Laurea Magistrale in Informatica, Università di Padova

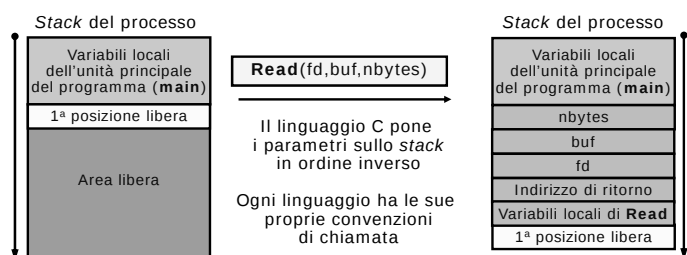
6 / 43



Sistemi distribuiti: comunicazione

RPC – 2

- ❑ Chiamata di procedura locale



Laurea Magistrale in Informatica, Università di Padova

8 / 43



Sistemi distribuiti: comunicazione

RPC – 3

❑ I parametri di procedura locale possono essere inviati per valore (*call-by-value*) o per riferimento (*call-by-reference*)

- Un parametro inviato per valore viene semplicemente copiato sullo *stack* del chiamato
 - Le modifiche apportate dal chiamato non hanno effetto sul chiamante
- Un parametro passato per riferimento fornisce un accesso (puntatore) a una variabile nello spazio del chiamante
 - Le modifiche apportate dal chiamante hanno effetto sul chiamato
- La variante *call-by-value-return* produce effetto sul chiamante solo al ritorno

Laurea Magistrale in Informatica, Università di Padova

9 / 43



Sistemi distribuiti: comunicazione

RPC – 5

❑ Le procedure remote nello spazio del chiamante sono descritte da una procedura fittizia detta *client stub* invocabile con le convenzioni locali

- Questa procedura svolge le azioni necessarie per effettuare la chiamata remota e riceverne il ritorno
 - Inoltra chiamata e attesa ritorno
- Tali azioni avvengono tramite scambio messaggi trasparente all'applicazione

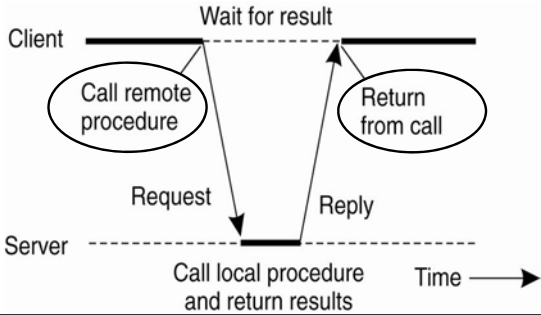
Laurea Magistrale in Informatica, Università di Padova

11 / 43



Sistemi distribuiti: comunicazione

RPC – 4



Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Laurea Magistrale in Informatica, Università di Padova

10 / 43



Sistemi distribuiti: comunicazione

RPC – 6

❑ L'arrivo del messaggio nello spazio del chiamato attiva una procedura fittizia detta *server stub*

- Questa procedura trasforma il messaggio in chiamata locale alla procedura invocata, ne raccoglie l'esito e lo inoltra al chiamante come messaggio

❑ In questo modo il chiamante e il chiamato hanno trasparenza di locazione

Laurea Magistrale in Informatica, Università di Padova

12 / 43



Sistemi distribuiti: comunicazione

RPC – 7

Il *client stub* trasforma la chiamata in una sequenza di messaggi da inviare sulla rete

- Parameter marshaling
 - Relativamente agevole con parametri passati per valore
 - Occorre solo assicurare trasparenza di accesso
 - Rappresentazione dei valori secondo le convenzioni di chiamante e chiamato
 - Molto più difficile con parametri passati per riferimento

Il *server stub* esegue una trasformazione analoga e opposta

- Parameter un-marshaling

Laurea Magistrale in Informatica, Università di Padova

13 / 43



Sistemi distribuiti: comunicazione

RPC – 9

Tre aspetti chiave caratterizzano lo specifico protocollo RPC

- Il formato dei messaggi scambiati tra gli *stub*
- La rappresentazione dei dati attesa da chiamante e chiamato
 - Encoding
- La modalità di comunicazione su rete
 - P.es.: TCP, UDP, ...

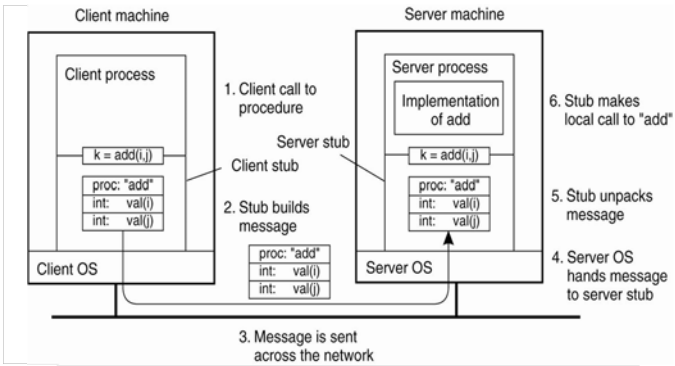
Laurea Magistrale in Informatica, Università di Padova

15 / 43



Sistemi distribuiti: comunicazione

RPC – 8



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"

Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Laurea Magistrale in Informatica, Università di Padova

14 / 43



Sistemi distribuiti: comunicazione

RPC – 10

Un servente si rende noto ai suoi clienti tramite registrazione del suo nodo di residenza presso un'anagrafe pubblica

Il cliente prima localizza il nodo di residenza del servente e poi il processo servente (la sua porta)

- Binding
- In ascolto sulla porta del servente può trovarsi un *daemon*

Laurea Magistrale in Informatica, Università di Padova

16 / 43



Sistemi distribuiti: comunicazione

RPC – 11

- ❑ RPC di base è sincrona
 - Ma ammette anche variante asincrona
- ❑ Può avere semantica diversa in presenza di errori di rete
 - *At-least-once*
 - *At-most-once*
 - *Exactly-once*
- ❑ Determinata dal protocollo di richiesta e risposta in uso tra gli *stub*

Laurea Magistrale in Informatica, Università di Padova

17 / 43



Sistemi distribuiti: comunicazione

Semantica della comunicazione – 2

- ❑ Semantica *maybe o best effort*
 - Nessun meccanismo in uso
 - Il cliente non può sapere quante volte la sua richiesta sia stata eseguita
- ❑ Semantica *at least once*
 - Il lato cliente usa RR1, il lato servente niente
 - Se la risposta arriva il cliente non sa quante volte sia stata calcolata dal servente

Laurea Magistrale in Informatica, Università di Padova

19 / 43



Sistemi distribuiti: comunicazione

Semantica della comunicazione – 1

- ❑ Il protocollo di *request-reply* alla base di RPC combina 3 meccanismi base
 - Lato cliente: *Request Retry – RR1*
 - Il cliente prova fino a ottenere risposta o certezza del guasto del destinatario
 - Lato servente: *Duplicate Filter – DF*
 - Il servente scarta gli eventuali duplicati provenienti dallo stesso cliente
 - Lato servente: *Result Retransmit – RR2*
 - Il servente conserva le risposte per poterle ritrasmettere senza ricalcolarle
 - Fondamentale per calcolo non idempotente (!)

Laurea Magistrale in Informatica, Università di Padova

18 / 43



Sistemi distribuiti: comunicazione

Semantica della comunicazione – 3

- ❑ Semantica *at most once*
 - Tutti i meccanismi in uso
 - Se la risposta arriva il cliente sa che è stata calcolata una sola volta
 - La risposta non arriva solo a causa di guasti permanenti del servente
- ❑ Semantica *exactly once*
 - Ha bisogno di meccanismi supplementari per tollerare guasti di lato servente
 - P.es. replicazione trasparente

Laurea Magistrale in Informatica, Università di Padova

20 / 43



Sistemi distribuiti: comunicazione

RMI – 1

Il paradigma RPC può essere facilmente esteso al modello a oggetti distribuiti

Soluzioni storiche

- CORBA (Common Object Request Broker Architecture)
 - OMG
- DCOM (Distributed Component Object Model) poi .NET
 - Microsoft
- J2EE (Java 2 Platform Enterprise Edition) poi Enterprise Java Beans
 - Sun Microsystems ora Oracle

Laurea Magistrale in Informatica, Università di Padova

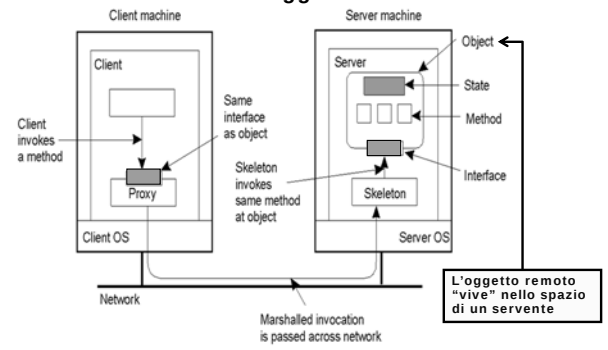
21 / 43



Sistemi distribuiti: comunicazione

RMI – 3

Realizzazione di oggetti distribuiti



L'oggetto remoto "vive" nello spazio di un servente

Laurea Magistrale in Informatica, Università di Padova

23 / 43



Sistemi distribuiti: comunicazione

RMI – 2

La separazione logica tra interfaccia e oggetto facilita la distribuzione

- L'interfaccia di un oggetto può essere distribuita senza che lo sia il suo stato interno
- Al *binding* di un cliente con un oggetto distribuito, una copia dell'interfaccia del servente (*proxy*) viene caricata nello spazio del cliente
 - Il ruolo del *proxy* è analogo a quello del *client stub* in ambiente RPC
- La richiesta in arrivo all'oggetto remoto viene trattata da un "agente" (*skeleton*) del cliente localmente al servente
 - Il ruolo dello *skeleton* è analogo a quello del *server stub* in ambiente RPC

Laurea Magistrale in Informatica, Università di Padova

22 / 43



Sistemi distribuiti: comunicazione

RMI – 4

Vi sono oggetti di tipo *compile-time*

- La cui realizzazione è completamente determinata dal linguaggio di programmazione
 - Ambiente e protocollo d'uso noti e uniformi ma non *inter-operable*

E oggetti di tipo *run-time*

- Quando ciò che si vuole far apparire come oggetto non lo è necessariamente nella sua concreta realizzazione
 - L'entità concreta (più spesso solo la sua interfaccia) viene incapsulata in un *object wrapper* che appare all'esterno come un normale oggetto distribuito
 - In questo modo si ottiene *inter-operability*

Laurea Magistrale in Informatica, Università di Padova

24 / 43



Sistemi distribuiti: comunicazione

RMI – 5

❑ Vi sono oggetti persistenti

○ Che continuano a esistere anche al di fuori dello spazio di indirizzamento del processo servente

- Lo stato persistente dell'oggetto distribuito viene salvato in memoria secondaria e da lì ripristinato dai processi servente delegati a farlo

❑ E oggetti transitori

○ Che cessano di esistere insieme al processo servente che li contiene

❑ Modelli RMI diversi fanno scelte diverse

Laurea Magistrale in Informatica, Università di Padova

25 / 43



Sistemi distribuiti: comunicazione

RMI – 7

❑ L'invocazione remota può essere statica

○ Quando è nota al compilatore che predispone l'invocazione del *proxy* dal lato cliente

- L'interfaccia del servizio deve essere noto al programmatore del cliente
- Se cambia l'interfaccia deve cambiare anche il cliente (nuova compilazione)

❑ Oppure dinamica

○ Quando viene costruita a tempo d'esecuzione

- Sia l'oggetto distribuito che il metodo desiderato sono parametri assegnati dal programma (ignoti al compilatore)
- Cambiamenti nell'interfaccia non hanno impatto sul codice del cliente

Laurea Magistrale in Informatica, Università di Padova

27 / 43



Sistemi distribuiti: comunicazione

RMI – 6

❑ RMI offre maggiore trasparenza di RPC

○ I riferimenti a oggetti distribuiti hanno *scope* globale possono essere liberamente scambiati a livello sistema

○ Un riferimento poco scalabile usa un analogo del *daemon* RPC per interconnettere cliente e servente dell'oggetto

- <indirizzo di rete del *daemon*; identificatore del servente>

❑ Modalità di riferimento

○ *Explicit binding*

- Il cliente deve passare attraverso un registro che restituisce un puntatore al *proxy* dell'oggetto servente (**Java RMI**)

○ *Implicit binding*

- Il linguaggio risolve direttamente il riferimento (**C++ Distr_object**)

Laurea Magistrale in Informatica, Università di Padova

26 / 43



Sistemi distribuiti: comunicazione

RMI – 8

I parametri locali vengono passati per valore. Quelli remoti per riferimento: la copia dell'oggetto può essere troppo onerosa!

Laurea Magistrale in Informatica, Università di Padova

28 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 1

- ❑ **Comunicazione persistente**
 - Il messaggio del mittente viene trattenuto dal MW fino alla consegna al destinatario
- ❑ **Comunicazione transitoria**
 - Non garantisce consegna del messaggio al destinatario perché è fragile rispetto ai possibili guasti (temporanei o permanenti)
 - Analogo al modello di servizio del protocollo UDP
- ❑ **Comunicazione asincrona**
 - Il mittente attende solo fino alla presa in carico del messaggio da parte del MW
- ❑ **Comunicazione sincrona**
 - Il mittente attende fino alla ricezione del destinatario o del suo MW

Laurea Magistrale in Informatica, Università di Padova

29 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 3

- ❑ **Comunicazione persistente e asincrona**
 - Come per la posta elettronica
- ❑ **Comunicazione persistente e sincrona**
 - Mittente bloccato fino alla ricezione garantita del destinatario
- ❑ **Comunicazione transitoria e asincrona**
 - Mittente non attende ma messaggio può andare perso → UDP
- ❑ **Comunicazione transitoria e sincrona**
 1. Mittente bloccato fino all'arrivo del messaggio nel MW del destinatario
 2. Mittente bloccato fino alla copia (**non garantita**) del messaggio nello spazio del destinatario → RPC asincrona
 3. Mittente bloccato fino alla ricezione di un messaggio di risposta dal destinatario → RPC standard e RMI

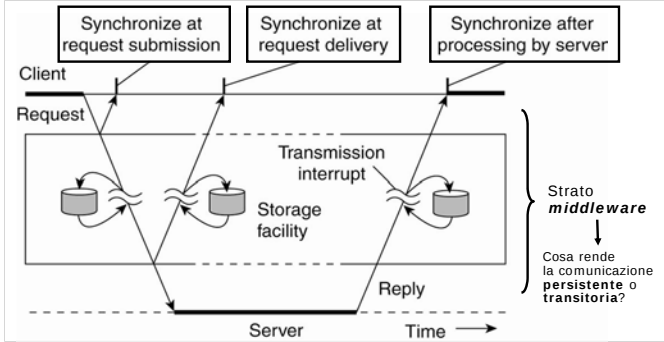
Laurea Magistrale in Informatica, Università di Padova

31 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 2



Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

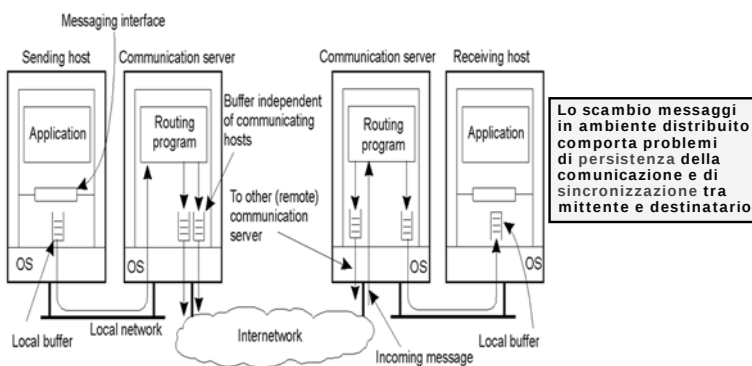
Laurea Magistrale in Informatica, Università di Padova

30 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 4



Laurea Magistrale in Informatica, Università di Padova

32 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 5

❑ **Middleware orientato a messaggi**

- Applicazioni distribuite comunicano tramite inserzione di messaggi in specifiche code → modello a code di messaggi
 - Eccellente supporto a comunicazioni persistenti e asincrone
 - Nessuna garanzia che il destinatario prelevi il messaggio dalla sua coda
- Di immediata realizzazione tramite
 - **Put**: non bloccante (asincrona → come trattare il caso di coda piena?)
 - **Get**: bloccante (sincrona rispetto alla presenza di messaggi in coda)
 - Un meccanismo di **callback** separa la coda dall'attivazione del destinatario
 - Una risorsa protetta realizza la coda con metodo **Put** di tipo **P** e metodo **Get** di tipo **E**
 - Realizzando coda *proxy* presso mittente e coda *skeleton* presso destinatario

Laurea Magistrale in Informatica, Università di Padova

33 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 7

❑ Il **middleware** realizza una rete logica sovrapposta a quella fisica (**overlay network**) con topologia propria e distinta

- Ciò richiede un proprio servizio di instradamento (**routing**)
- Una sottorete connessa di instradatori conosce la topologia della rete logica e si occupa di far pervenire il messaggio del mittente alla coda del destinatario
- Topologie complesse e variabili (scalabili) richiedono gestione dinamica delle corrispondenze coda-indirizzo di rete, in totale analogia con quanto avviene nel modello IP

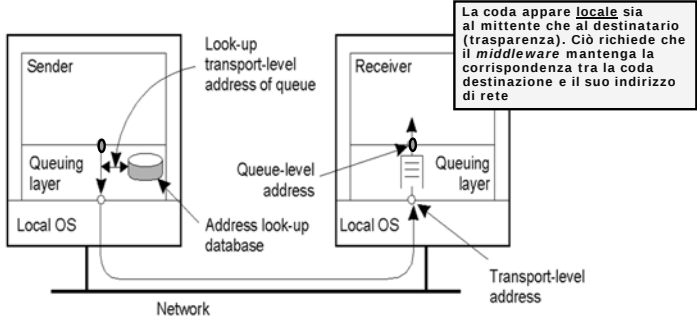
Laurea Magistrale in Informatica, Università di Padova

35 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 6



La coda appare locale sia al mittente che al destinatario (trasparenza). Ciò richiede che il **middleware** mantenga la corrispondenza tra la coda destinazione e il suo indirizzo di rete

Laurea Magistrale in Informatica, Università di Padova

34 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 8

❑ Un **broker** fornisce trasparenza di accesso a messaggi il cui formato aderisce a standard di trasporto diversi nel suo percorso

- Servizio analogo a quello offerto dai **gateway** delle reti

❑ La natura del **middleware** è di essere adattivo e non intrusivo rispetto all'ambiente ospite

Laurea Magistrale in Informatica, Università di Padova

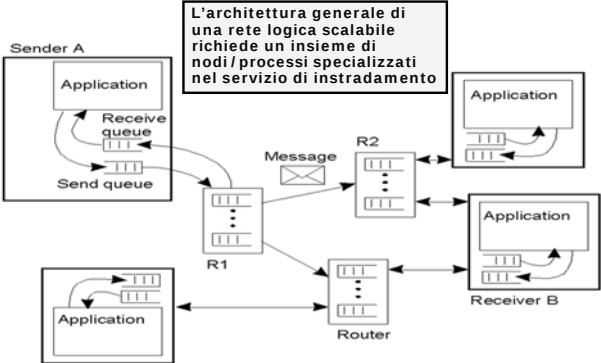
36 / 43



Sistemi distribuiti: comunicazione

Scambio messaggi – 9

L'architettura generale di una rete logica scalabile richiede un insieme di nodi/processi specializzati nel servizio di instradamento



Laurea Magistrale in Informatica, Università di Padova

37 / 43



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 2

- ❑ L'invio di *data stream* è facilitato dal mezzo trasmissivo a rappresentazione continua ma non ne è dipendente
 - Le *pipe* di UNIX e le connessioni di TCP/IP forniscono un mezzo trasmissivo a rappresentazione discreta (orientato a [gruppi di] *byte*)
 - Tuttavia possono essere usate per lo *streaming*
- ❑ I requisiti di trasmissione variano con il tipo di contenuto informativo scambiato

Laurea Magistrale in Informatica, Università di Padova

39 / 43



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 1

- ❑ Il contenuto delle comunicazioni scambiate tramite RPC, RMI e messaggi non dipende dal tempo di ricezione
 - Mezzo trasmissivo detto “a rappresentazione discreta”
 - Adatto a testo, fotografie digitali, *file* oggetto, eseguibili
- ❑ Vi sono comunicazioni il cui contenuto presenta dipendenze temporali forti
 - Mezzo trasmissivo detto “a rappresentazione continua”
 - Adatto a video, audio • *data stream* : una sequenza di unità dati collegate
 - Requisiti temporali espressi come “Qualità del Servizio” (QoS)

Laurea Magistrale in Informatica, Università di Padova

38 / 43



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 3

- ❑ Tramissione asincrona
 - Preserva l'ordinamento ma non la distanza temporale tra unità dati
- ❑ Tramissione sincrona
 - Preserva l'ordinamento e garantisce un tempo massimo di trasmissione di ogni singola unità dati
- ❑ Tramissione isocrona
 - Aggiunge la garanzia di un tempo minimo di trasmissione → *bounded (delay) jitter*

Laurea Magistrale in Informatica, Università di Padova

40 / 43



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 4

❑ I singoli *stream* possono essere composti di parti temporalmente correlate

- Questo porta un problema di sincronizzazione nel flusso di trasmissione

❑ *Stream* come connessione virtuale tra sorgente e destinazione

- *Multicast* : più destinatari di uno stesso *stream*
- La connessione deve essere configurata in termini di risorse fisiche e logiche dedicate (*Quality of Service*)

Laurea Magistrale in Informatica, Università di Padova

41 / 43



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 6

❑ Un esempio di sincronizzazione di *stream*

- MPEG (*Motion Picture Expert Group*)
 - Un insieme di algoritmi standard per la compressione di video e audio
 - Per combinare in un singolo *stream* un insieme illimitato di *stream* distinti sia discreti che continui
 - Ciascuno *stream* originario viene trasformato in un flusso di unità dati (*frame*) la cui sequenza è determinata da un'etichetta temporale generata da un orologio unico con caratteristiche fissate (p.es. 90 MHz)
 - I pacchetti di ciascuno *stream* vengono combinati mediante *multiplexing* in una sequenza composta di pacchetti a lunghezza variabile ma con propria etichetta temporale
 - A destinazione si ricompongono gli *stream* originari usando l'etichetta temporale per riprodurre la sincronizzazione tra ciascuna unità dati al suo interno

Laurea Magistrale in Informatica, Università di Padova

43 / 43



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 5

❑ QoS espressa come specifica di flusso

- Capacità di trasporto (*bandwidth*), frequenza di trasmissione, ritardi di trasmissione, ecc.
- Il *Token Bucket Algorithm* fissa il contributo dello *stream* al traffico di rete
 - Un controllore produce gettoni che “comprano” un diritto d'uscita
 - Ogni unità dati in ingresso viene inviata sulla rete se può consumare un gettone altrimenti resta in coda (o viene scartata a coda piena)
 - Le unità dati vengono emesse sulla rete con la frequenza di generazione del gettone indipendentemente da possibili irregolarità di lato mittente

Laurea Magistrale in Informatica, Università di Padova

42 / 43