



Sistemi distribuiti: comunicazione

Comunicazione



Anno accademico 2009/10
Sistemi Concorrenti e Distribuiti

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

Corso di Laurea Magistrale in Informatica, Università di Padova

1 / 37



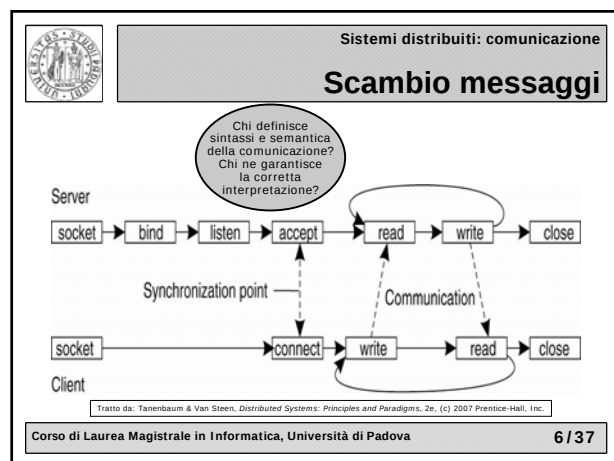
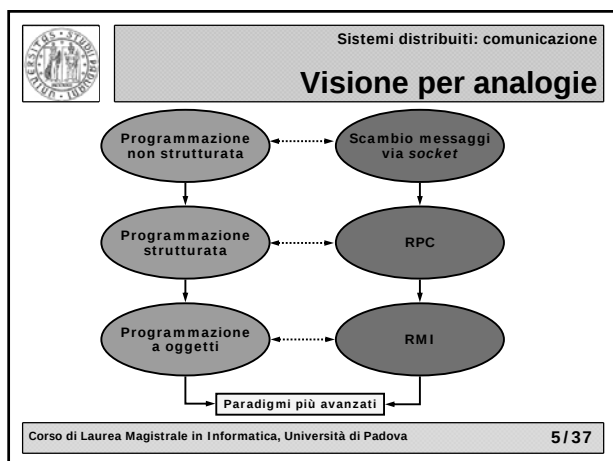
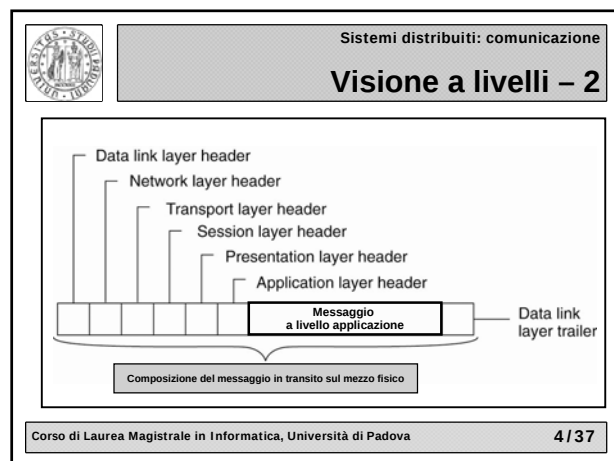
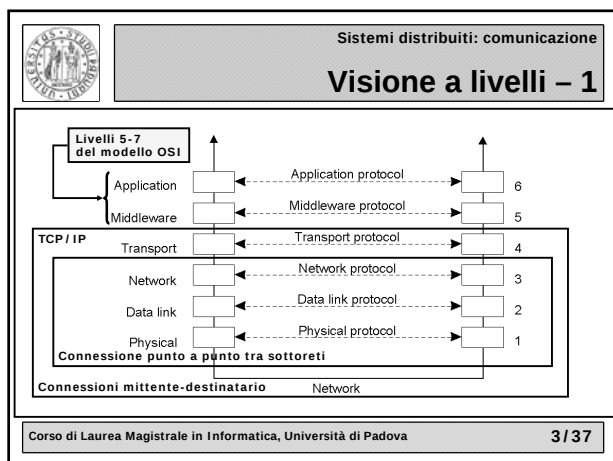
Sistemi distribuiti: comunicazione

Evoluzione di modelli

- ❑ **Remote Procedure Call (RPC)**
 - Modello cliente-servente trasparente rispetto allo scambio messaggi
- ❑ **Remote (Object) Method Invocation (RMI)**
 - Modello cliente-servente basato su oggetti distribuiti
- ❑ **Scambio messaggi a livello *middleware***
 - Modelli più avanzati e protocolli più potenti e specializzati di quelli di rete
- ❑ **Stream o comunicazioni a flusso continuo**
 - Flusso di dati che richiedono continuità temporale

Corso di Laurea Magistrale in Informatica, Università di Padova

2 / 37





Sistemi distribuiti: comunicazione

RPC – 1

- ❑ Consentire a un processo C residente su un elaboratore (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!

Corso di Laurea Magistrale in Informatica, Università di Padova

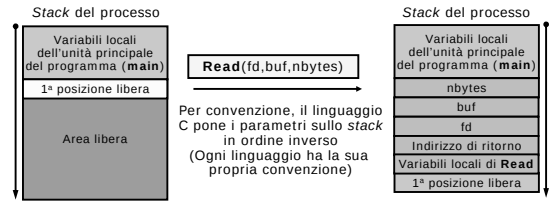
7 / 37



Sistemi distribuiti: comunicazione

RPC – 2

❑ Chiamata di procedura “normale”



Corso di Laurea Magistrale in Informatica, Università di Padova

8 / 37



Sistemi distribuiti: comunicazione

RPC – 3

❑ I parametri di procedura “normale” possono essere inviati per valore (*call-by-copy*) 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 chiamato hanno effetto sul chiamante
 - Ma nella variante *call-by-value-return* l'effetto si ha solo al ritorno

Corso di Laurea Magistrale in Informatica, Università di Padova

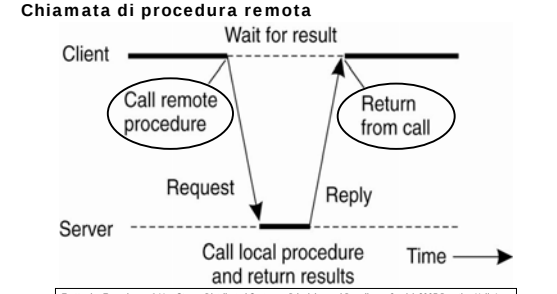
9 / 37



Sistemi distribuiti: comunicazione

RPC – 4

Chiamata di procedura remota



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

Corso di Laurea Magistrale in Informatica, Università di Padova

10 / 37



Sistemi distribuiti: comunicazione

RPC – 5

❑ Trasparenza di locazione di chiamante e chiamato

- Nello spazio del chiamante le procedure invocabili in remoto sono descritte da una procedura fittizia detta *client stub* invocabile con le normali convenzioni
- Questa racchiude tutte le azioni necessarie per effettuare la chiamata remota e a riceverne l'esito
 - Invio della richiesta e ricezione dei valori di ritorno avvengono tramite scambio messaggi
- Nello spazio del chiamato l'arrivo del messaggio attiva una procedura fittizia detta *server stub*
- Questa trasforma il messaggio in una chiamata locale alla procedura invocata, ne raccoglie l'esito e lo invia al chiamante come messaggio sulla rete

Corso di Laurea Magistrale in Informatica, Università di Padova

11 / 37



Sistemi distribuiti: comunicazione

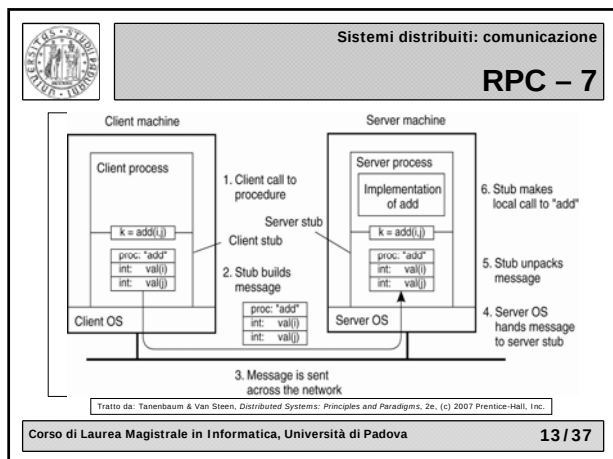
RPC – 6

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

- *Parameter marshalling*
 - 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 arduo con parametri passati per riferimento
- ❑ Il *server stub* esegue una trasformazione analogica e opposta
 - *Parameter unmarshalling*

Corso di Laurea Magistrale in Informatica, Università di Padova

12 / 37



Sistemi distribuiti: comunicazione

RPC – 8

- ❑ Il formato dei messaggi scambiati sulla rete è un aspetto del protocollo di RPC
- ❑ La rappresentazione dei dati (*encoding*) attesa da chiamante e chiamato è un altro aspetto
 - Sia per i tipi elementari che per le strutture
- ❑ Ulteriore aspetto è la modalità di comunicazione su rete (p.es. TCP, UDP)

Corso di Laurea Magistrale in Informatica, Università di Padova 14/37

Sistemi distribuiti: comunicazione

RPC – 9

- ❑ 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 può trovarsi un *daemon*
- ❑ RPC è sincrona ma può essere resa asincrona e con varie semantiche di trattamento di errori
 - *At-least-once*, *At-most-once*, *Exactly-once*
 - In relazione al protocollo di gestione di richieste e risposte

Corso di Laurea Magistrale in Informatica, Università di Padova 15/37

Sistemi distribuiti: comunicazione

Semantica della comunicazione – 1

- ❑ Il protocollo di *request-reply* determina la semantica di comunicazione in presenza di errori
- ❑ Usando una combinazione di 3 meccanismi di base
 - Lato cliente: *Riprova (Request Retry – RR1)*
 - Il cliente continua a provare fino a ottenere risposta o la certezza del guasto del destinatario
 - Lato servente: *Filtra i duplicati (Duplicate filtering – DF)*
 - Il servente scarta gli eventuali duplicati di richieste provenienti dallo stesso cliente
 - Lato servente: *Ritrasmetti le risposte (Result Retransmit – RR2)*
 - Il servente conserva le risposte per poterle ritrasmettere senza ricalcolarle
 - Fondamentale per calcolo non idempotente!

Corso di Laurea Magistrale in Informatica, Università di Padova 16/37

Sistemi distribuiti: comunicazione

Semantica della comunicazione – 2

- ❑ Semantica "*maybe*"
 - 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 nessun meccanismo
 - All'arrivo di una risposta il cliente non sa quante volte sia stata calcolata dal servente e dunque non conosce per certo lo stato del servente
- ❑ 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 (p.es. replicazione trasparente) per tollerare guasti di lato servente

Corso di Laurea Magistrale in Informatica, Università di Padova 17/37

Sistemi distribuiti: comunicazione

RMI – 1

- ❑ Il paradigma RPC può essere facilmente esteso al modello a oggetti distribuiti
 - Soluzioni storiche – standard di riferimento
 - CORBA (*Common Object Request Broker Architecture*) – OMG
 - DCOM (*Distributed Component Object Model*) – .NET – Microsoft
 - J2EE (*Java 2 Platform, Enterprise Edition*) – Enterprise JavaBeans – Sun Microsystems
- ❑ La separazione logica tra interfaccia e oggetto consente anche la loro separazione fisica
 - Lo stato interno di un oggetto non viene distribuito!
 - Al *binding* di un cliente con un oggetto servente distribuito, una copia dell'interfaccia del servente (*proxy*) viene caricata nello spazio del cliente
 - Ruolo del tutto analogo a quello del *client stub* in ambiente RPC
 - La richiesta in arrivo all'oggetto remoto viene trattata da un "agente" del cliente, locale al servente (*skeleton*)
 - Ruolo del tutto analogo a quello del *server stub* in ambiente RPC

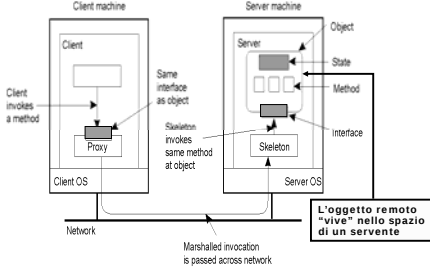
Corso di Laurea Magistrale in Informatica, Università di Padova 18/37



Sistemi distribuiti: comunicazione

RMI – 2

Realizzazione di oggetti distribuiti



Corso di Laurea Magistrale in Informatica, Università di Padova

19 / 37



Sistemi distribuiti: comunicazione

RMI – 3

- ❑ **Oggetti di tipo *compile-time***
 - Realizzazione completamente determinata dal linguaggio di programmazione
 - Ambiente e protocollo d'uso noti e uniformi
- ❑ **Oggetti di tipo *run-time***
 - Ciò che si vuole far apparire come oggetto senza che la sua concreta realizzazione lo sia effettivamente
 - L'entità concreta, spesso solo la sua interfaccia, viene incapsulata in un "adattatore" (*object wrapper*) che interagisce con l'esterno come un normale oggetto distribuito

Corso di Laurea Magistrale in Informatica, Università di Padova

20 / 37



Sistemi distribuiti: comunicazione

RMI – 4

- ❑ **Oggetti persistenti**
 - 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
- ❑ **Oggetti transitori**
 - Cessano di esistere insieme al processo servente che li contiene
- ❑ **Modelli diversi fanno scelte diverse**

Corso di Laurea Magistrale in Informatica, Università di Padova

21 / 37



Sistemi distribuiti: comunicazione

RMI – 5

- ❑ **Maggior trasparenza rispetto a RPC**
 - I riferimenti a oggetti distribuiti solo validi e possono essere scambiati a livello sistema
 - *System-wide* (in RMI) vs. *scoped* (in RPC) ←
- ❑ **2 modalità di utilizzo dei riferimenti**
 - *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 del cliente all'oggetto distribuito (C++ *Distr_object*)
- ❑ **Un analogo del *daemon* di RPC media tra il cliente e il nodo del servente dell'oggetto richiesto**
 - Riferimento a oggetto distribuito (scarsamente scalabile)
 - <indirizzo di rete del *daemon*, identificatore di livello sistema del servente>

Corso di Laurea Magistrale in Informatica, Università di Padova

22 / 37



Sistemi distribuiti: comunicazione

RMI – 6

- ❑ **Invocazione statica**
 - 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*)
- ❑ **Invocazione dinamica**
 - Deve essere 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 sulla struttura del cliente

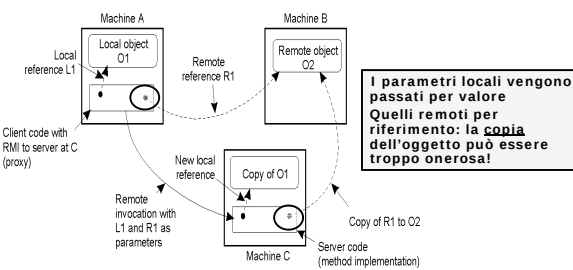
Corso di Laurea Magistrale in Informatica, Università di Padova

23 / 37



Sistemi distribuiti: comunicazione

RMI – 7



Corso di Laurea Magistrale in Informatica, Università di Padova

24 / 37



Sistemi distribuiti: comunicazione

Scambio messaggi – 1

- ❑ **Comunicazione persistente**
 - Il messaggio inviato dal mittente viene trattenuto e preservato dalla infrastruttura di comunicazione fino alla sua consegna al destinatario
- ❑ **Comunicazione transitoria**
 - Fragile rispetto ai possibili guasti (temporanei o permanenti)
 - Non garantisce la consegna del messaggio al destinatario
 - Corrisponde al modello di servizio offerto dal protocollo UDP
- ❑ **Comunicazione asincrona**
 - Il mittente attende solo fino alla prima memorizzazione del messaggio
- ❑ **Comunicazione sincrona**
 - Il mittente attende fino alla ricezione del destinatario (o del suo nodo di residenza)

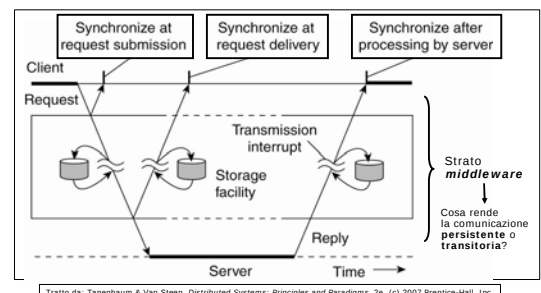
Corso di Laurea Magistrale in Informatica, Università di Padova

25 / 37



Sistemi distribuiti: comunicazione

Scambio messaggi – 2



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

Corso di Laurea Magistrale in Informatica, Università di Padova

26 / 37



Sistemi distribuiti: comunicazione

Scambio messaggi – 3

- ❑ **6 possibili combinazioni reali**
 - **Comunicazione persistente e asincrona**
 - Esempio: posta elettronica
 - **Comunicazione persistente e sincrona**
 - Mittente bloccato fino alla copia (garantita) del messaggio presso il destinatario
 - **Comunicazione transitoria e asincrona**
 - Mittente non attende ma messaggio perso se destinatario non raggiungibile → **UDP**
 - **Comunicazione transitoria e sincrona**
 - 1: mittente bloccato fino alla copia del messaggio nel nodo 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**

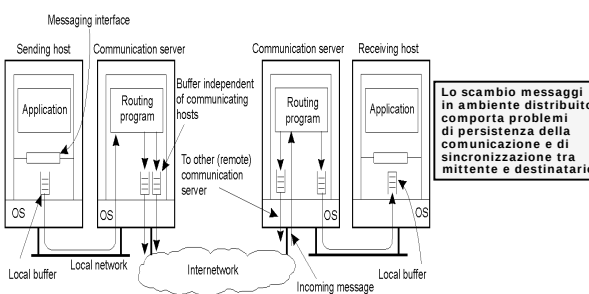
Corso di Laurea Magistrale in Informatica, Università di Padova

27 / 37



Sistemi distribuiti: comunicazione

Scambio messaggi – 4



Corso di Laurea Magistrale in Informatica, Università di Padova

28 / 37



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

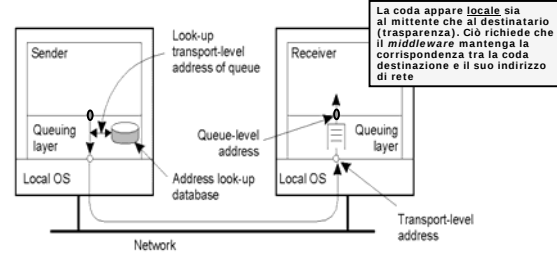
Corso di Laurea Magistrale in Informatica, Università di Padova

29 / 37



Sistemi distribuiti: comunicazione

Scambio messaggi – 6



Corso di Laurea Magistrale in Informatica, Università di Padova

30 / 37



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
- Un adattatore (**broker**) fornisce trasparenza di accesso a messaggi il cui formato aderisca a standard di trasporto diversi nel suo percorso
 - La natura del **middleware** è adattativa e non intrusiva

Corso di Laurea Magistrale in Informatica, Università di Padova

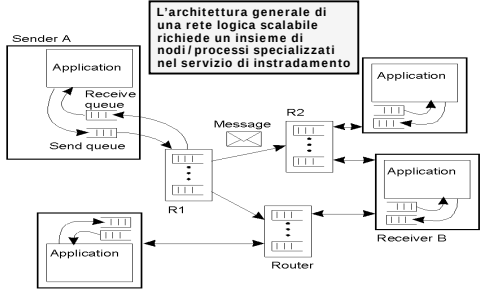
31 / 37



Sistemi distribuiti: comunicazione

Scambio messaggi – 8

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



The diagram illustrates a message exchange architecture. On the left, 'Sender A' contains an 'Application' box, a 'Receive queue' box, and a 'Send queue' box. An arrow points from the 'Send queue' to a 'Message' box. This message is then sent to a 'Router' box, which is connected to two other router boxes, 'R1' and 'R2'. From 'R1', an arrow points to an 'Application' box. From 'R2', an arrow points to a 'Receiver B' box, which contains an 'Application' box. The 'Router' box is also connected to 'R1' and 'R2'.

Corso di Laurea Magistrale in Informatica, Università di Padova

32 / 37



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)

Corso di Laurea Magistrale in Informatica, Università di Padova

33 / 37



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**)
- Vincoli temporali sulla modalità trasmissiva
 - Asincrona**
 - Preserva l'ordinamento, non la distanza temporale tra unità dati
 - Sincrona**
 - Preserva l'ordinamento e garantisce un **tempo massimo di trasmissione** di ogni unità dati
 - Isocrona**
 - Aggiunge la garanzia di un tempo minimo di trasmissione → **bounded (delay) jitter**

Corso di Laurea Magistrale in Informatica, Università di Padova

34 / 37



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 3

- Gli **stream** possono essere composti (internamente strutturati) con requisiti temporali tra le parti che li compongono
 - Problema di sincronizzazione
- 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 (QoS)

Corso di Laurea Magistrale in Informatica, Università di Padova

35 / 37



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 4

- 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

Corso di Laurea Magistrale in Informatica, Università di Padova

36 / 37



Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 5

□ 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 (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

Corso di Laurea Magistrale in Informatica, Università di Padova

37 / 37