Comunicazione e sistemi distribuiti

Flooding e Shortest Path Tree

Roberto Marino, PhD

Università di Messina — CdL in Informatica — Reti e Sistemi Distribuiti Mod.B

17 maggio 2026



UniMe
1548

Il modello di rete

In un sistema distribuito i processori non condividono memoria: comunicano *esclusivamente* scambiando messaggi lungo canali diretti. Modelliamo la struttura fisica con un digrafo.

Definizione 1.38 — Rete sincrona

Il componente fisico di una **rete sincrona** $\mathcal{S}$ è un digrafo (I, E_{cmm}) dove:

- $I = \{1, \ldots, n\}$ è l'insieme degli **identificatori unici (UID)** dei processori
- E_{cmm} è un insieme di archi diretti su I, detti **communication links**

Il termine *sincrono* indica che **tutti** i processori comunicano nello stesso istante (stesso round). Esiste un clock globale implicito.

Reti asincrone (per confronto)

In una rete **asincrona** non esiste un clock globale: ogni messaggio può impiegare un tempo arbitrario (ma finito) per arrivare. I processori non possono distinguere tra un vicino lento e un vicino in crash.

Stato complessivo della rete

Fissato un algoritmo distribuito $\mathcal{DA}$, ogni processore i ha un insieme di stati $W^{[i]}$ e un sottoinsieme di stati iniziali ammissibili $W_0^{[i]} \subseteq W^{[i]}$.

Stato complessivo della rete

Lo **stato complessivo** (o *configurazione globale*) della rete al tempo ℓ è la tupla

$$\mathrm{w}(\ell) \;=\; \left(w^{[1]}(\ell),\; w^{[2]}(\ell),\; \ldots,\; w^{[n]}(\ell)\right)$$

dove $w^{[i]}(\ell) \in W^{[i]}$ è lo stato del processore i al round ℓ.

Lo stato iniziale $\mathrm{w}(0)$ assegna a ciascun processore un valore in $W_0^{[i]}$.

Due funzioni per ogni processore i:

- $\mathrm{msg}^{[i]} : W^{[i]} \times I \to \mathbb{A}$
  genera il messaggio da inviare al vicino j
- $\mathrm{stf}^{[i]} : W^{[i]} \times \mathbb{A}^n \to W^{[i]}$
  aggiorna lo stato in base ai messaggi ricevuti

Ad ogni round ℓ, in quest'ordine:

1. **Trasmissione**: i invia $\mathrm{msg}^{[i]}(w^{[i]}, j)$ a ogni out-neighbor j
2. **Aggiornamento**: i calcola il nuovo stato con $\mathrm{stf}^{[i]}$ applicata allo stato corrente e ai messaggi appena ricevuti

Algoritmo distribuito: definizione formale

Definizione 1.39 — Algoritmo distribuito

Un **algoritmo distribuito** $\mathcal{DA}$ per una rete $\mathcal{S}$ consiste degli insiemi $\mathbb{A}$, $\{W^{[i]}\}_{i\in I}$, $\{W_0^{[i]}\}_{i\in I}$ e delle mappe:

$$\mathrm{msg}^{[i]} : W^{[i]} \times I \;\to\; \mathbb{A}, \qquad \mathrm{stf}^{[i]} : W^{[i]} \times \mathbb{A}^n \;\to\; W^{[i]}, \qquad i \in I$$

dove $\mathbb{A}$ è l'alfabeto di comunicazione (`null` $\in \mathbb{A}$; `null` indica assenza di messaggio).

Se $W^{[i]} = W$, $\mathrm{msg}^{[i]} = \mathrm{msg}$ e $\mathrm{stf}^{[i]} = \mathrm{stf}$ per ogni i, l'algoritmo è detto **uniforme**.

Remarks 1.41 — Modello di comunicazione

- La comunicazione è **point-to-point**: i può inviare messaggi diversi a vicini diversi e conosce il mittente di ogni messaggio ricevuto
- I messaggi codificano valori dell'alfabeto $\mathbb{A}$: numeri, booleani, ...
- In algoritmi uniformi si usa spesso la *standard mgf*: $\mathrm{msg}_{\mathrm{std}}(w, j) = w$

Perché la distinzione uniforme / non uniforme?

Un algoritmo uniforme è più robusto: non richiede che i processori conoscano a priori la propria identità o il numero di nodi. Gli algoritmi che vedremo (Flooding e Bellman-Ford distribuito) sono entrambi **uniformi**.

Broadcast e BFS Tree Computation

Il primo scenario classico negli algoritmi distribuiti è la comunicazione *uno-a-tutti*: come propaga l'informazione un singolo nodo all'intera rete?

Problema 1.50 — Broadcast

Un processore, detto **sorgente**, possiede un messaggio (*token*) μ. Trasmettere il token a *tutti* gli altri processori della rete.

Problema 1.52 — BFS Tree Computation

Data $\mathcal{S}$, calcolare una **rappresentazione distribuita** di un BFS tree radicato in v: ogni nodo figlio memorizza l'identità del proprio `parent`; la radice sa di non avere parent.

Breadth-First Search (BFS)

La BFS visita i nodi di un grafo per livelli: prima tutti i nodi raggiungibili dalla sorgente v in 1 hop, poi quelli in 2 hop, e così via. Il cammino da v a qualsiasi nodo nell'albero è il più corto possibile in numero di archi.

Flooding Algorithm — Idea e stato

Il **Flooding Algorithm** risolve *Broadcast* e *BFS Tree Computation* contemporaneamente.

Idea. La sorgente ($v = 1$) invia il token ai propri out-neighbor al round 0. In ogni round successivo, ogni nodo che riceve il token per la prima volta: (1) lo memorizza, (2) registra il mittente come `parent`, (3) lo propaga ai propri out-neighbor nel round seguente. Chi ha già ricevuto il token, o non ne ha ancora uno, non trasmette nulla.

Stato di ogni processore:

$$w = (\underbrace{\texttt{parent}}_{\in\{0,\ldots,n\}},\ \underbrace{\texttt{data}}_{\in\mathbb{A}},\ \underbrace{\texttt{snd-flag}}_{\in\{\texttt{false},\texttt{true}\}})$$

`parent`: predecessore nel BFS tree
`data`: il token (o `null` se non ancora ricevuto)
`snd-flag`: `true` esattamente nel round in cui si propaga

Stato iniziale:

	Sorgente $i = 1$	Ogni $j \neq 1$
`parent`	1	0
`data`	μ	`null`
`snd-flag`	`true`	`false`

La sorgente ha già il token e lo trasmette al primo round.

Flooding Algorithm — Formalismo completo

Specifica dell'algoritmo

Rete sincrona: $\mathcal{S} = (\{1, \ldots, n\}, E_{\text{cmm}})$

Algoritmo distribuito: FLOODING

Alfabeto: $\mathbb{A} = \{\alpha, \ldots, \omega\} \cup \{\texttt{null}\}$

Stato: $w = (\texttt{parent}, \texttt{data}, \texttt{snd-flag})$

function `msg`(w, i)
  if `parent` $\neq i$ **and** `snd-flag` $=$ `true` **then**
    return `data`
  else **return** `null`

(si trasmette a tutti tranne il parent)

function `stf`(w, y)

case `data` $=$ `null` **and** y contiene solo `null`
nessun token ricevuto ancora
new-parent := `null`; new-data := `null`
new-snd-flag := `false`

case `data` $=$ `null` **and** y ha un msg $\neq$ `null`
token ricevuto per la prima volta
new-parent := UID minimo tra i mittenti non-`null`
new-data := il messaggio ricevuto
new-snd-flag := `true`

case `data` $\neq$ `null`
token già posseduto: non ritrasmettere
new-parent := `parent`; new-data := `data`
new-snd-flag := `false`

return (new-parent, new-data, new-snd-flag)

Shortest-paths tree: il problema

Aggiungiamo i pesi. Un digrafo pesato associa a ogni arco $(j \to i)$ un peso *strettamente positivo* $a_{ji} > 0$. Il **cammino minimo** da 1 a i è quello che minimizza la somma dei pesi degli archi percorsi.

Problema — Shortest-paths tree computation

Data una rete sincrona $\mathcal{S} = (\{1, \ldots, n\}, E_{\text{cmm}}, A)$ con pesi $A = (a_{ji})$, calcolare in forma distribuita un **albero dei cammini minimi** radicato nella sorgente 1: ogni nodo i deve conoscere il proprio `parent` nell'albero e la propria distanza `dist` dalla sorgente.

Caso centralizzato vs. distribuito.

Il caso centralizzato si risolve con l'algoritmo di **Dijkstra**, che esplora i nodi in ordine di distanza crescente e richiede una struttura dati condivisa (coda di priorità). Questo è impossibile in un sistema distribuito.

Il **Bellman-Ford distribuito** rimpiazza la coda di priorità con lo scambio iterativo di stime tra vicini.

Idea chiave: rilassamento degli archi.

Ogni nodo i mantiene una stima $\texttt{dist}^{[i]}$ della propria distanza dalla sorgente. Ad ogni round riceve le stime dei vicini e verifica se esiste un cammino più corto passando per uno di loro:

$$\texttt{dist}^{[i]} \leftarrow \min_j \left(y_j + a_{ji} \right)$$

se questo valore è minore della stima corrente.

Distributed Bellman-Ford — Formalismo completo

Specifica dell'algoritmo

Rete: $\mathcal{S} = (\{1, \ldots, n\}, E_{\text{cmm}}, A)$

Algoritmo: DISTRIBUTED BELLMAN-FORD

Alfabeto: $\mathbb{A} = \mathbb{R}_{>0} \cup \{\texttt{null}\} \cup \{+\infty\}$

Stato: $w = (\texttt{parent}, \texttt{dist})$

Stato iniziale:

$\texttt{parent}^{[j]} = j$ per ogni j

$\texttt{dist}^{[1]} = 0$ (sorgente)

$\texttt{dist}^{[j]} = +\infty$ per $j \neq 1$

function `msg`(w, i)

if round $< n$ **then** **return** `dist`

else **return** `null`

(si trasmette la propria stima per n round)

function `stf`(w, y)

$i :=$ UID del processore corrente

$k := \arg\min\{ y_j + a_{ji} \mid \forall\, y_j \neq \texttt{null} \}$

k è il vicino che offre il cammino più corto verso i

if $\texttt{dist} \leq y_k + a_{ki}$ **then**

la stima corrente è già ottima: non aggiornare

return $(\texttt{parent}, \texttt{dist})$

else

trovato un cammino più corto passando per k

return $(k,\ y_k + a_{ki})$

Convergenza

Dopo al più $n - 1$ round le stime `dist` sono quelle ottimali. Ciò segue dal fatto che ogni cammino minimo ha al più $n - 1$ archi.

Riepilogo

Il modello

Una rete sincrona $\mathcal{S} = (I, E_{\text{cmm}})$ è un digrafo di processori che comunicano per round. Un algoritmo distribuito $\mathcal{DA}$ è specificato da $(W, \mathbb{A}, \text{msg}, \text{stf})$. Lo **stato complessivo** $\text{w}(\ell)$ cattura la configurazione globale al round ℓ.

Flooding Algorithm

Problema: Broadcast + BFS tree su reti non pesate.

Meccanismo: ogni nodo propaga il token esattamente una volta (`snd-flag`), registrando il mittente come `parent`.

Prestazioni: ottimale, $\text{radius}(v, \mathcal{S})$ round.

Distributed Bellman-Ford

Problema: Shortest-path tree su reti con pesi positivi.

Meccanismo: ogni nodo aggiorna `dist` se un vicino offre un cammino più corto ($y_j + a_{ji} <$ `dist`).

Prestazioni: converge in $n - 1$ round, $\Theta(n^2)$ messaggi.

Lo stesso framework ($\mathbb{A}$, W, msg, stf) si adatta a problemi diversi modificando solo lo stato e la logica di transizione.