

Lezione n.8

ALGORITMI EPIDEMICI/GOSSIP

Laura Ricci
21/3/2012

acknowledgment: Alberto Montresor

ALGORITMI EPIDEMICI/DI GOSSIP

Protocolli epidemici o di gossip:

- paradigma computazionale e di comunicazione orientato a sistemi distribuiti su larga scala con caratteristiche di alta dinamicità
- approccio probabilistico
- Caratteristiche principali:
 - semplicità
 - scalabilità
 - efficienza
 - robustezza: resistenza rispetto a fallimenti di nodi/perdita di messaggi/partizionamento della rete
 - convergenza ed accuratezza della soluzione garantite in modo probabilistico

ALGORITMI EPIDEMICI/DI GOSSIP

- Schema generale di un protocollo gossip:
 - ogni nodo della rete contatta uno o alcuni nodi(pochi) scelti uniformemente in **modo casuale** e scambia informazione con questi nodi
 - richiesto un **servizio di peer sampling** (campionamento) che aggiorni costantemente l'insieme di vicini di un nodo
 - lo scambio delle informazioni è periodico
- Diffusione della informazione
 - **bio inspired**: dinamica analoga a quella della **diffusione di una epidemia** nella popolazione
 - **autostabilizzante**
 - **self-repair**: non richiede meccanismi espliciti per la gestione del churn
- Svantaggi:
 - proprietà garantite con una certa probabilità, manca garanzia assoluta

GOSSIP: UN PO' DI STORIA

- Alan Demers ed altri di Xerox nel 1987 propongono un protocollo epidemico per l'**aggiornamento dello stato** di un data-base replicato
 - "Epidemic Algorithms for Replicated Database Management" (più di 1400 citazioni)
 - Applicazioni precedenti: Network News Transport Protocol
- Anni '90: diffusione della informazione in sistemi distribuiti
- Anni 2000, applicazioni nel campo delle reti P2P
- Attualmente, esistono implementazioni reali in:
 - **Amazon S3** (Simple Storage System): diffusione della informazione
 - gossip per individuare i server disponibili e lo stato dei nodi
 - **Amazon Dynamo**: failure detection e membership service
 - **Cassandra**: database distribuito utilizzato nelle prime versioni di Facebook
 - **Bittorrent**: diffusione della informazione all'interno di uno swarm

GOSSIP: APPLICAZIONI

- Diffusione epidemica della informazione
 - anti-entropia:
 - diffusione epidemica degli aggiornamenti dello stato dei nodi
 - ogni nodo sceglie periodicamente in modo casuale uniformemente un altro nodo e i due nodi coinvolti aggiornano il loro stato
 - rumor mongering
 - diffusione di un 'rumore'
 - i nodi che sono venuti a conoscenza del rumore, i 'nodi infetti', periodicamente scelgono un nodo in modo casuale uniformemente e diffondono quel rumore
- Calcolo di valori aggregati
 - Es: media, contare il numero di nodi totali sulla rete,...
- Overlay bootstrap
 - A partire da una configurazione casuale dei nodi, costruire un overlay strutturato

ALGORITMI EPIDEMICI: STRUTTURA GENERALE

Algoritmo epidemico

- si ispirano a comportamenti che si presentano in natura
- esiste una **popolazione** composta da un insieme di entità comunicanti
- esiste un insieme di **semplici regole** che definiscono come ogni entità può diffondere una informazione alle altre
- ogni entità ha esclusivamente una **visione locale** dell'ambiente

Processi epidemici: meccanismi di diffusione di entità di tipo diverso su reti complesse

- diffusione di virus in una popolazione
- diffusione di 'pettegolezzi'
- diffusione di un incendio in una foresta

PROCESSI EPIDEMICI

Esempio classico: **diffusione di virus in una popolazione**

- semplici regole: una persona infetta che viene a contatto con una non infetta ha una probabilità dell'85% di infettarla
- classificazione della popolazione:
 - **Susceptible** : infettabile, non sono ancora infetti, ma possono infettarsi se vengono a contatto con un infetto
 - **Infective** : hanno la malattia e possono trasmetterla
 - **Recovered** : sono guariti dalla malattia ed hanno acquisito una immunità permanente, per cui non possono più diventare infettati o passare la malattia (in alcune varianti del modello, individui deceduti)

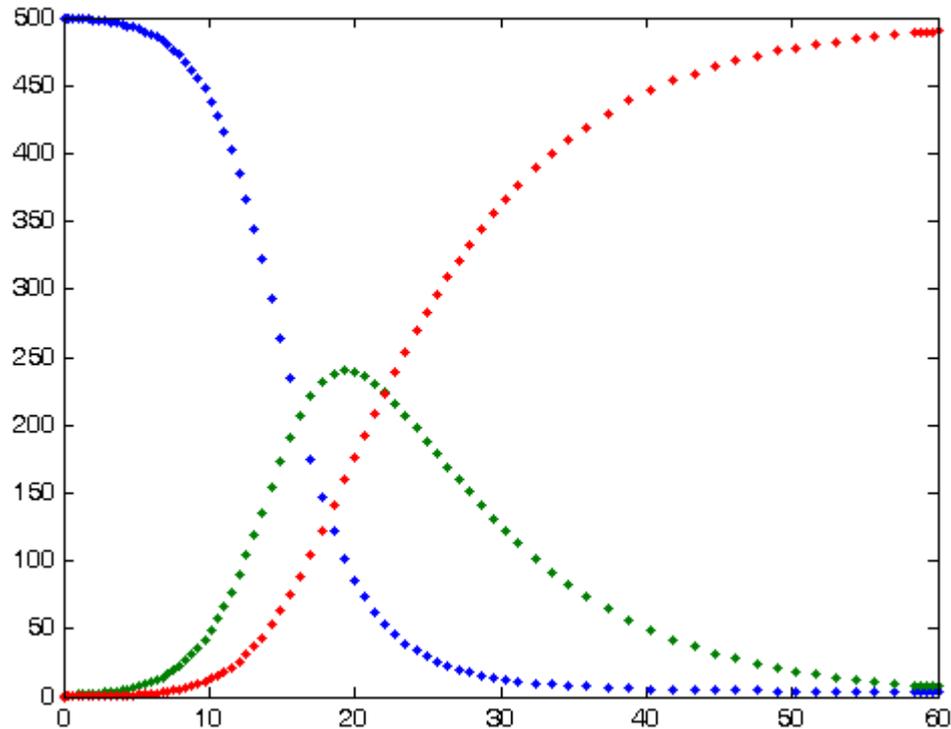


Il sistema può essere modellato mediante un insieme di **equazioni differenziali** la cui soluzione descrive la diffusione del virus nel tempo

PROCESSI EPIDEMICI

Modello

- inizialmente, un singolo individuo infettivo
- i suscettibili diventano infetti
- gli infetti, con il tempo, diventano removed



MODELLI EPIDEMICI

- **Susceptible-Infective (SI)**. inizialmente **infettabile**, diventa **infetto** e rimane tale finchè l'intera popolazione viene infettata



- **Susceptible-Infective-Removed-Susceptible(SIRS)** : immunità temporanea, un individuo può essere infettato, guarire e contrarre di nuovo la malattia



- **Susceptible-Exposed-Infective-Removed-Susceptible(SEIRS)**: prende in considerazione un nuovo stato, **exposed**, in cui la malattia è in incubazione



DAI PROCESSI EPIDEMICI AL GOSSIP

- i virus si diffondono rapidamente
 - possiamo adottare la stessa idea per la diffusione delle informazioni in un sistema distribuito?
- analogia individui/nodi di un sistema distribuito, ogni nodo può essere:
 - **susceptible**; il nodo non è ancora venuta a conoscenza della informazione da diffondere, ma è in grado di riceverla
 - **infective**: il nodo è venuta a conoscenza di una specifica informazione e può diffonderla, rispettando un insieme di regole
 - **removed**; non diffonde più l'informazione
- i modelli precedenti possono essere utilizzati come base per la definizione di algoritmi di gossip
 - rumor spreading è basato su questi concetti

DAI PROCESSI EPIDEMICI AL GOSSIP

Susceptible-Infective-Removed(SIR):

un nodo può decidere di interrompere la diffusione di un'informazione.

Non riprendere successivamente la diffusione dell'informazione.

una unità che non è riuscita a diffondere l'informazione negli ultimi x cicli di diffusione, decide che l'intera popolazione è stata infettata (tutti i nodi l'hanno ricevuta) ed interrompe la diffusione dell'informazione

Susceptible-Infective-Susceptible(SIS):

un nodo può decidere autonomamente di interrompere la diffusione dell'informazione, prima che la popolazione sia completamente infettata.

il nodo può riprendere in seguito la diffusione dell'informazione

DIFFUSIONE DELLA INFORMAZIONE

- **Anti-entropia:** algoritmo epidemico il cui scopo è quello di 'riconciliare' le diverse repliche di una variabile condivisa
 - Push
 - Pull
 - Push/Pull
- Rumor mongering

ANTI ENTROPIA

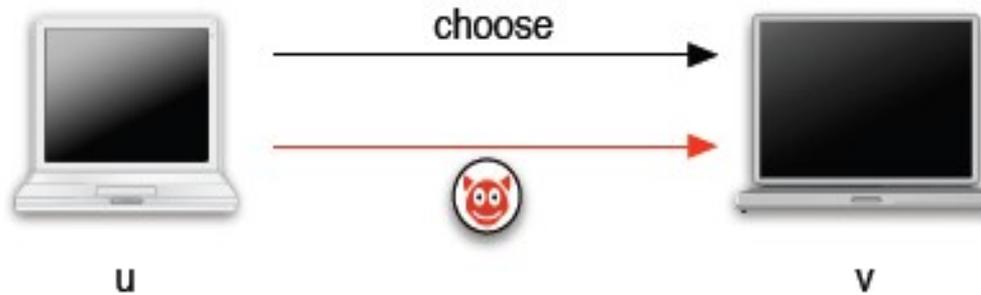
Modelli di comunicazione per algoritmi epidemici SI

- la computazione avviene secondo una **sequenza di cicli**
- ad ogni ciclo i nodi si contattano a coppie, diffondendo l'informazione

Strategie per la diffusione della informazione: u contatta v

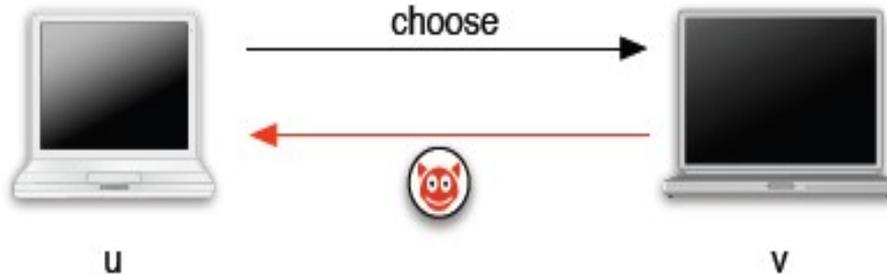
- **push:** u può infettare v , ma non viceversa
- **pull:** u può essere infettato da v , ma non viceversa
- **pushpull:** se una delle due unità (u o v) è infetta, e l'altra suscettibile, entrambe le unità diventano infette

STRATEGIA PUSH

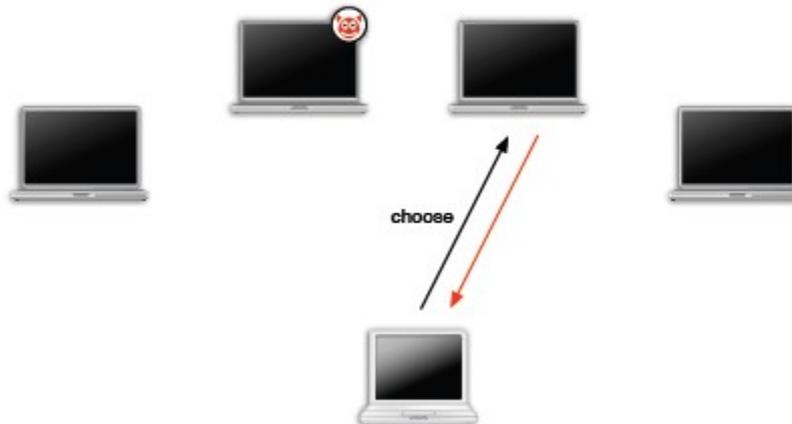


- All'inizio, solo pochi nodi sono infetti e la probabilità di scegliere un nodo non infetto è alta
- Successivamente la probabilità di trovare un nodo non infetto diminuisce

STRATEGIA PULL

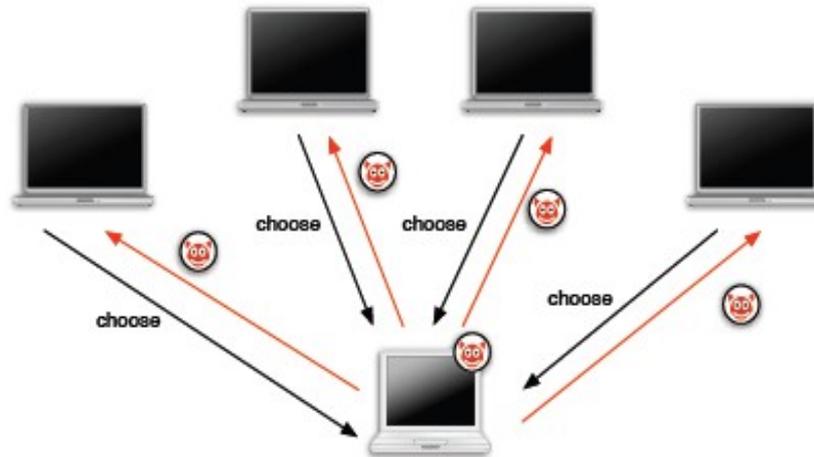


- all'inizio la probabilità di essere infettati è bassa
- non esiste garanzia che la diffusione dell'informazione inizi al primo ciclo

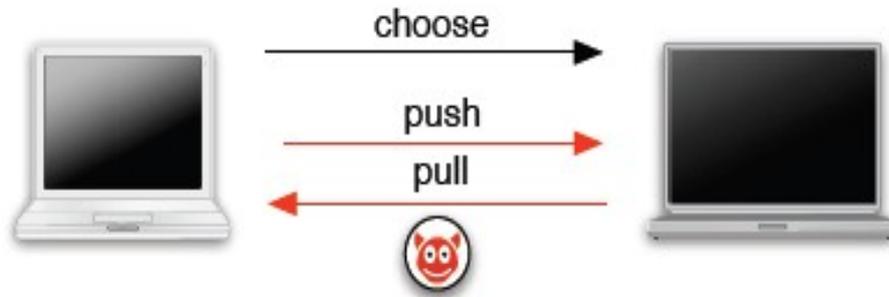


STRATEGIA PULL

- Esiste anche la possibilità che l'intera popolazione venga infettata in un solo ciclo



STRATEGIA PUSH-PULL



- unisce i vantaggi degli approcci precedenti
- all'inizio dominano le **operazioni di push**, perchè il numero di unità non infettate è maggiore di quelle infettate, nell'ultima parte quelle di pull, perchè è più probabile individuare nodi infettati
- la probabilità che un nodo non infetto contatti un nodo infetto cresce ad ogni ciclo, mentre quella che un nodo infetto contatti uno non infetto diminuisce ad ogni round

MODELLO DI RIFERIMENTO

- $\Pi = \{p_1, \dots, p_k\}$ insieme di nodi che possiedono una informazione condivisa x
- $Value_i$ funzione time-dependent restituisce il valore della copia di x memorizzata su p_i

$$Value_i: \Pi \rightarrow V \times T$$

V : insieme dei valori

T : insieme dei timestamps

- $Value(p_i) = (v, t)$, valore memorizzato nel nodo p_i al tempo t
- $Value(p_i) = (nil, t)$, eliminazione di x al tempo t da parte di p_i
- $Value(p_i) = (v, now)$, aggiornamento di x , now restituisce un timestamp globalmente univoco
- **scopo della diffusione:**
 - **eventual consistency**: se nessun nodo effettua un aggiornamento dopo il tempo t , allora prima o poi, dopo t , tutti i nodi avranno lo stesso valore di x
 - diffusione aggiornamenti di stato per ottenere eventual consistency

ANTI ENTROPIA PROTOCOLLO PUSH

Anti-entropy, Push protocol executed by process p_i :

repeat every Δ time units

┌ PROCESS $p_j \leftarrow \text{random}(\Pi)$ % Select a random neighbor
└ send $\langle \text{PUSH}, \text{value}_i \rangle$ to p_j

upon receive $\langle \text{PUSH}, (v, t) \rangle$ do

┌ if $\text{value}_i.\text{time} < t$ then
└ ┌ $\text{value}_i \leftarrow (v, t)$

- ogni nodo sceglie periodicamente in modo casuale un altro nodo e gli invia il valore della variabile condivisa
- l'aggiornamento avviene solo se il nodo che esegue la PUSH ha un valore più aggiornato del partner prescelto

ANTI ENTROPIA PROTOCOLLO PULL

Anti-entropy, Pull protocol executed by process p_i :

repeat every Δ time units

┌ PROCESS $p_j \leftarrow \text{random}(\Pi)$ % Select a random neighbor
└ send $\langle \text{PULL}, p_i, \text{value}_i.\text{time} \rangle$ to p_j

upon receive $\langle \text{PULL}, p_j, t \rangle$ do

┌ if $\text{value}_i.\text{time} > t$ then
└ ┌ send $\langle \text{REPLY}, \text{value}_i \rangle$ to p_j

upon receive $\langle \text{REPLY}, (v, t) \rangle$ do

┌ if $\text{value}_i.\text{time} < t$ then
└ ┌ $\text{value}_i \leftarrow (v, t)$

- ogni nodo sceglie regolarmente in modo casuale un altro nodo e gli invia il valore della variabile condivisa
- L'aggiornamento avviene solo se il nodo che ha ricevuto la richiesta di PULL ha un valore più aggiornato del nodo che ha fatto la richiesta di pull

ANTI ENTROPIA PROTOCOLLO PUSH PULL

Anti-entropy, Push-Pull protocol executed by process p_i :

repeat every Δ time units

┌ PROCESS $p_j \leftarrow \text{random}(\Pi)$ % Select a random neighbor
└ send $\langle \text{PUSHPULL}, p_i, \text{value}_i \rangle$ to p_j

upon receive $\langle \text{PUSHPULL}, p_j, (v, t) \rangle$ do

┌ if $\text{value}_i.\text{time} < t$ then
│ | $\text{value}_i \leftarrow (v, t)$
└ else if $\text{value}_i.\text{time} > t$ then
 ┌ send $\langle \text{REPLY}, \text{value}_i \rangle$ to p_j

upon receive $\langle \text{REPLY}, (v, t) \rangle$ do

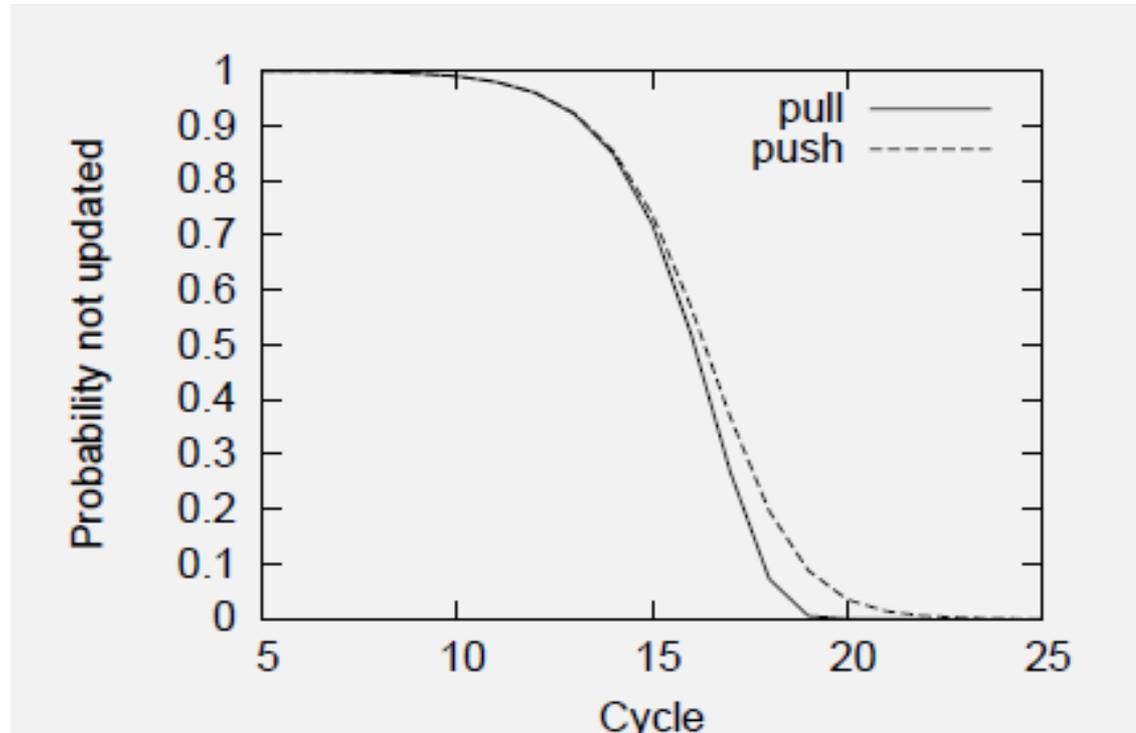
┌ if $\text{value}_i.\text{time} < t$ then
└ | $\text{value}_i \leftarrow (v, t)$

- ogni nodo sceglie regolarmente in modo casuale un altro nodo e gli invia il valore della variabile condivisa
- il nodo che ha il valore più aggiornato della variabile (nodo infetto) infetta l'altro nodo

ANTI ENTROPIA: ANALISI

- Valutazione del protocollo epidemico sulla base della popolazione dei suscettibili e degli infetti
 - p_i probabilità che un nodo sia suscettibile dopo l' i -esimo round
 - $i_i = 1 - p_i$ probabilità che un nodo sia infetto dopo l' i -esimo round
- Vogliamo valutare p_{i+1}
 - **Condizione iniziale:** $p_0 = n-1/n$, un solo nodo possiede l'aggiornamento
 - **Pull:** $p_{i+1} = (p_i)^2$
 - un nodo rimane suscettibile al round $i+1$ se lo era al round i e se ha contattato un nodo suscettibile al round i
 - **Push:** $p_{i+1} = p_i \left(1 - \frac{1}{n}\right)^{n(1-p_i)}$ ovvero, approssimativamente $p_{i+1} = p_i e^{-1}$
 - un nodo rimane suscettibile al round $i+1$ se lo era al round i e se nessun nodo infetto al round i lo ha contattato

ANTI ENTROPIA: ANALISI



Rapida convergenza

Pull migliore di push

ANTI ENTROPIA: ANALISI

- Sia T_n il primo ciclo in cui il numero di nodi suscettibili è uguale a 0, per una popolazione di n individui
- Push: $T_n = \log n + \ln n + O(1)$
- PushPull: $T_n = \log \log n$
- Bibliografia
 - B. Pittel On spreading a rumor, SIAM Journal of Applied Mathematics 47(1), 1987
 - R. Karp, C.Schindelhauer, S.Shenker, B. vocking, Randomized rumor spreading, Proc. 41 Symposium on foundation of Computer Science, 2000

RUMOR MONGERING

- n individui, inizialmente tutti in stato susceptible
- uno di essi viene a conoscenza di una notizia (rumor), telefona ad un altro scelto in modo casuale e 'condivide il rumore'
- ogni persona diventa infetta ogni volta che è venuta a conoscenza del rumore lo condivide a sua volta
- quando un individuo effettua una telefonata non necessaria (il destinatario conosce già la notizia), con una certa probabilità l'individuo 'perde interesse' nella diffusione della notizia (diventa removed)
- si vuole studiare
 - convergenza del sistema, dopo quanti cicli i nessuno è più infetto
 - percentuale di individui che conoscono la notizia (removed) quando viene raggiunto questo stato

RUMOR MONGERING

- Tutti i nodi inizialmente suscettibili
- Un nodo diventa infetto quando riceve un aggiornamento (rumor)
- Un nodo che possiede il 'rumor' sceglie periodicamente in maniera casuale un altro nodo e diffonde il rumore
- Dopo un certo numero di round, il nodo 'perde interesse' nella diffusione del rumore, cessa di diffonderlo e passa nello stato 'removed'
 - dopo k contatti
 - con probabilità $1/k$
- Blind vs. Feedback
 - destinatario già infettato
 - nessuna informazione sul destinatario

RUMOR MONGERING: IL MODELLO

Siano

- S_t numero di individui suscettibili al tempo t
- I_t numero di individui infetti al tempo t
- R_t numero individui rimossi al tempo t
- N dimensione della popolazione
- $s_t = S_t/N$ frazione di suscettibili
- $i_t = I_t/N$ frazione di infetti
- $r_t = R_t/N$ frazione di rimossi
- Valgono le seguenti relazioni

$$S_t + I_t + R_t = N$$

$$s_t + i_t + r_t = 1$$

RUMOR MORGERING: IL MODELLO

Parametri del sistema:

- γ numero di persone contattate in media da un individuo infetto
- α percentuale di individui susceptible che se, contattati, si infettano a loro volta
- $\beta = \gamma \alpha$
- κ = probabilità che un individuo infetto diventi immune
- i tre gruppi di individui (susceptible, infected, removed) hanno contatti tra di loro distribuiti in modo uniforme
 - un infetto incontra un infetto o un immune: nessun cambiamento nel sistema
 - un infetto incontra un suscettibile: avviene un nuovo contagio
- Il numero medio di individui contagiati da un infetto è $\beta \times s_+$

RUMOR MORGERING: ANALISI

- Se si discretizza il tempo in una sequenza di intervalli, otteniamo il sistema

$$\left\{ \begin{array}{l} S_{t+1} = S_t - \beta s_t I_t \\ I_{t+1} = I_t + \beta s_t I_t - \kappa I_t = I_t(1 + \beta s_t - \kappa) \\ R_{t+1} = R_t + \kappa I_t \end{array} \right.$$

- Nella seconda equazione si tiene di conto degli infetti che sono passati in uno stato di immunità
- Modellazione del sistema mediante un **sistema di equazioni differenziali** (derivate calcolate rispetto al tempo)

RUMOR MORGERING: MODELLO

- analisi degli stati di equilibrio(steady states): una situazione in cui il sistema(o alcune parti del sistema) non subiscono variazioni.
 - non esiste uno stato di equilibrio per gli individui infatti se la popolazione è costante, a parte quello banale in cui $i_t=0$ per $t=0$
 - supponiamo $i_0>0$: i_{t+1} sarà maggiore o minore di i_t , ma mai uguale
- Consideriamo l'equazione $i_{t+1} = i_t(1+\beta s_t - v)$
- $\rho_t = (1+\beta s_t - v)$ **epidemic threshold**
 - $\rho_t > 1$, il numero degli individui infettati aumenta
 - $\rho_t < 1$ il numero di individui infettati diminuisce
 - $\rho_t = 1$ impossibile
- i_t decresce a velocità via via crescente
- se la popolazione è costante, la popolazione infettata diventa nulla

PROTOCOLLI GOSSIP: FUNZIONI DI AGGREGAZIONE

Data una rete di N nodi, si supponga che il nodo i memorizza il valore x_i

- **Aggregazione:** ogni nodo deve determinare il valore di una funzione globale di aggregazione

$$f(\) = f(x_0, x_1, \dots, x_{n-1})$$

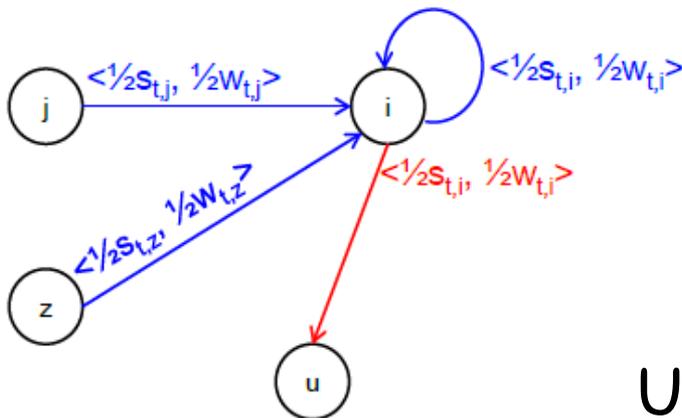
- esempi di funzioni di aggregazione: somma, media, massimo, minimo, quantili,..
- applicazioni
 - calcolare il numero di nodi sulla rete
 - minimo, massimo
 - media
 - quantili

IL PROTOCOLLO PUSH SUM

- calcolare il valore medio dei valori posseduti dai peer
- ogni nodo i memorizza una somma locale $sum_{t,i}$ ed un peso locale $w_{t,i}$
- Inizializzazione: il nodo i si invia la coppia $\langle x_i, w_{oi} \rangle$
 - Algoritmo eseguito al ciclo t dal nodo i

Algorithm 1 Protocol Push-Sum

- 1: Let $\{(\hat{s}_r, \hat{w}_r)\}$ be all pairs sent to i in round $t - 1$
 - 2: Let $s_{t,i} := \sum_r \hat{s}_r$, $w_{t,i} := \sum_r \hat{w}_r$
 - 3: Choose a target $f_t(i)$ uniformly at random
 - 4: Send the pair $(\frac{1}{2}s_{t,i}, \frac{1}{2}w_{t,i})$ to $f_t(i)$ and i (yourself)
 - 5: $\frac{s_{t,i}}{w_{t,i}}$ is the estimate of the average in step t
-

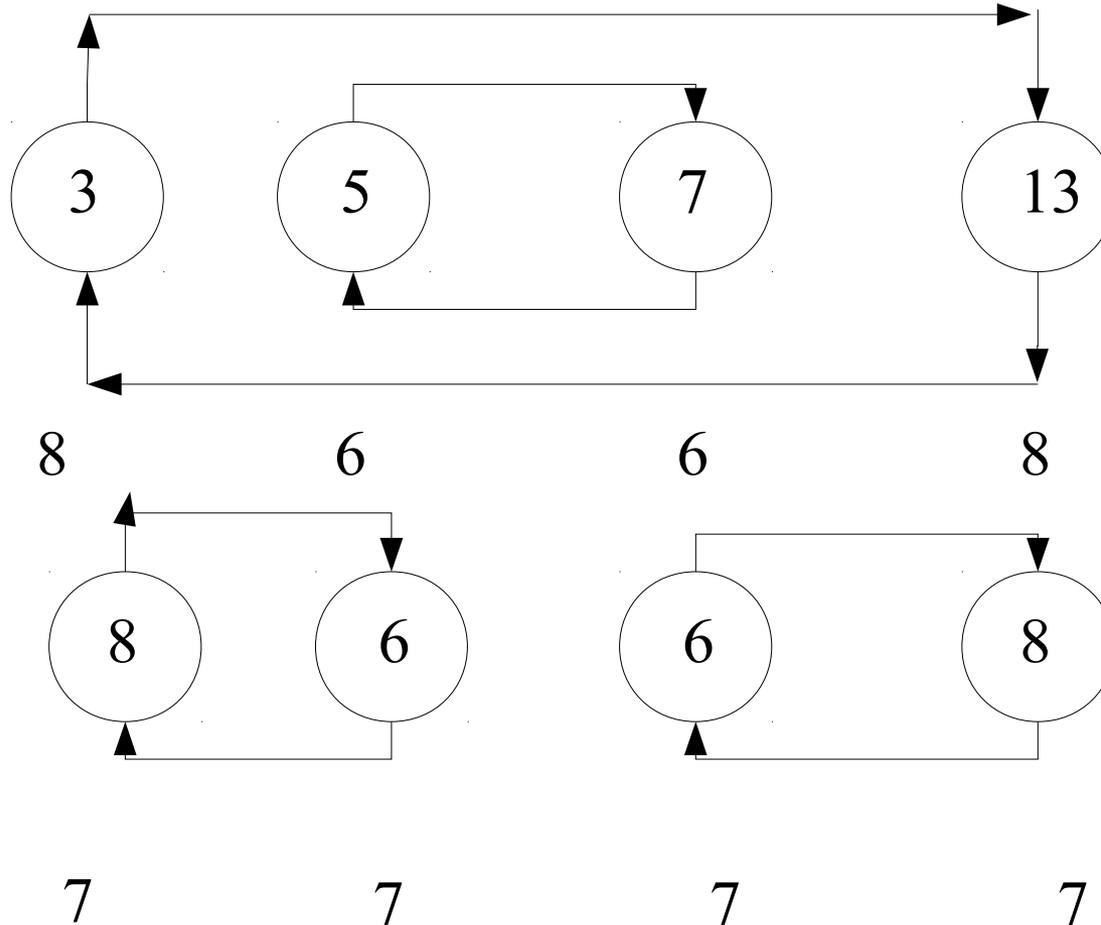


Aggiornamento al ciclo i

$$\begin{cases} s_{t+1,i} = \frac{1}{2}s_{t,j} + \frac{1}{2}s_{t,i} + \frac{1}{2}s_{t,z} \\ w_{t+1,i} = \frac{1}{2}w_{t,j} + \frac{1}{2}w_{t,i} + \frac{1}{2}w_{t,z} \end{cases}$$

Un passo di riduzione della varianza

IL PROTOCOLLO PUSH SUM: ESEMPIO



Calcolo della media
Media= 14

IL PROTOCOLLO PUSH SUM

- Cambiando opportunamente le inizializzazioni si possono ottenere diverse funzioni da aggregazione

Function	Description
Sum	$v_i = \text{local value}$ $w_i = 1$ at a single node, 0 at all other nodes
Count	$v_i = 1$ $w_i = 1$ at a single node, 0 at all other nodes
Average	$v_i = \text{local value}$ $w_i = 1$
Weighted Average	$v_i = \text{local value} \times \text{local weight}$ $w_i = \text{local weight}$

Convergenza: con probabilità $1-\delta$ l'errore nella approssimazione del valore aggregato è minore di ε , in al più $O(\log(N) + \log(1/\varepsilon) + \log(1/\delta))$ rounds

PEER SAMPLING SERVICE

- nel lavoro originale di Demers ogni nodo periodicamente effettua un ciclo di 'gossip' con un **nodo casuale scelto dall'insieme completo dei nodi**
 - rete statica
 - di piccola dimensione
- nel nostro caso
 - i nodi hanno una visione parziale della rete
 - la vista parziale è dinamica a causa del churn
- **Random Peer Sampling Service**
 - Deve definire un metodo `getpeer` che restituisce un peer scelto dalla visione locale del nodo, ma che approssimi una scelta casuale effettuata sull'intero insieme di nodi
- **Idea base: i nodi effettuano gossip con i loro vicini e l'argomento del gossip sono....gli altri vicini!**
 - random shuffling delle viste
 - rimozione dei nodi obsoleti

ALGORITMI EPIDEMICI

Modello di riferimento

- un insieme dinamico di nodi distribuiti
- ogni nodo partecipa al protocollo epidemico
 - i nodi possono unirsi/lasciare la rete dinamicamente
 - i nodi possono fallire in un qualsiasi istante
- comunicazione:
 - ogni nodo può comunicare con un sottoinsieme degli altri nodi di cui conosce l'indirizzo IP
 - i messaggi possono essere persi. Il protocollo deve funzionare anche in presenza di un alta percentuale di perdita di messaggi
- gli algoritmi epidemici 'classici' si basano su una assunzione importante:
 - ad ogni ciclo dell'algoritmo un nodo P può selezionare un nodo Q scelto in **modo casuale uniforme** tra l'insieme di tutti i nodi che partecipano al protocollo

PEER SAMPLING SERVICE

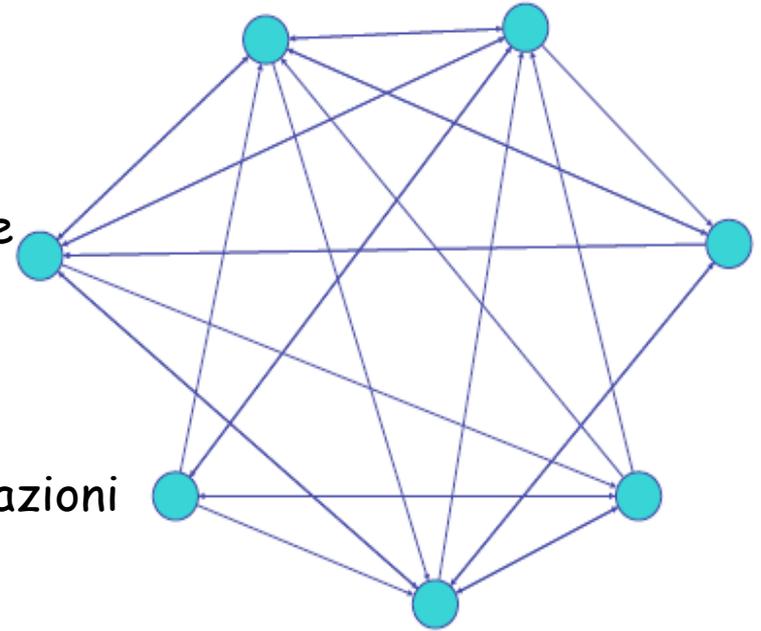
- in ogni algoritmo epidemico
 - **scelta vicini**: un peer sceglie in modo casuale un insieme di vicini con cui scambiare l'informazione (vicini da **infettare**)
 - **gossip**: scambia un insieme di informazioni con tali vicini. Esempio: propagazione degli update per riconciliare le repliche in un database
- **Peer sampling service classico**: `getPeer()`
 - **input**: l'insieme P dei peer che eseguono l'algoritmo epidemico
 - **output**: insieme di peer scelti **casualmente in modo uniforme** tra quelli di P
- In un ambiente P2P il Peer Sampling Service deve garantire
 - **scalabilità**
 - **accuratezza in presenza di alta dinamicità**: l'insieme dei peer restituiti dovrebbe essere scelto tra quelli **attualmente presenti** sulla rete
 - **indipendenza**: peer diversi dovrebbero ottenere 'campioni' indipendenti

PEER SAMPLING SERVICE DISTRIBUITO

- Un servizio di campionamento accurato richiede la presenza di un server centralizzato che registri la presenza dei peer e restituisca un campione di peer scelto in **modo casuale uniforme**
 - questa soluzione non è realizzabile in un ambiente P2P perchè non scalabile
- In un ambiente P2P il servizio di sampling può essere implementato mediante un **algoritmo epidemico**
 - ogni peer possiede una **propria vista della rete**
 - i peer si **scambiano frequentemente le proprie viste** (gossip) cambiando continuamente quindi la propria visione della rete
 - il sampling service restituisce un sottoinsieme dei peer presenti nella vista del peer che invoca la `getPeer()`
 - questa soluzione consente di **approssimare accuratamente la soluzione** in cui il 'campione' restituito dal servizio è scelto dall'insieme di tutti i **peer della rete**
- Il servizio di sampling viene utilizzato dagli algoritmi epidemici ai livelli superiori

PEER SAMPLING MEDIANTE ALGORITMI EPIDEMICI

- ogni nodo possiede una propria vista contenente C vicini
- per ogni vicino:
indirizzo IP
informazioni necessarie per implementare il servizio di campionamento
- ogni nodo contatta periodicamente un vicino nella propria vista e scambia con esso informazioni relative alla propria vista (gossip)
- i nodi coinvolti nel gossip aggiornano la propria vista in base alle informazioni ricevute
- la vista di un nodo cambia continuamente: **overlay altamente dinamico**



STRUTTURA GENERALE DI ALGORITMI DI GOSSIP

Active thread

```
selectPeer (&Q);  
selectToSend (&bufs);  
sendTo (Q, bufs);  
  
receiveFrom (Q, &bufr);  
selectToKeep (p_view, buf);
```

Passive thread

```
receiveFromAny (&P, &buf);  
selectToSend (&bufs);  
sendTo (P, bufs);  
selectToKeep (p_view, buf);
```

- **SelectPeer** : seleziona in modo casuale un peer dalla vista locale
- **SelectToSend** : seleziona alcune entrate dalla vista locale
- **SelectToKeep** :
 - aggiunge le informazioni ricevute alla vista locale,
 - elimina i duplicati e seleziona un sottoinsieme della vista risultante che definisce la nuova vista locale

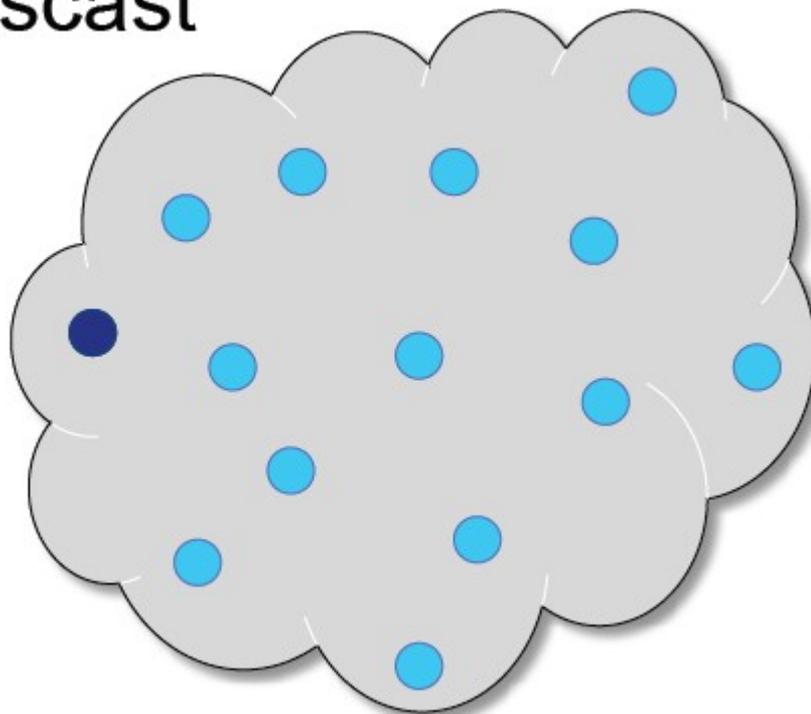
UN ESEMPIO: IL PROTOCOLLO NEWSCAST

- Decrittore di un peer: identificatore del peer + timestamp
- **SelectPeer**: seleziona un peer **in modo random** un peer dalla vista locale
- **SelectToSend**: restituisce tutti i descrittori appartenenti alla vista locale + un descrittore del peer associato ad un timestamp aggiornato all'ultimo ciclo di gossip
 - Nota: il timestamp indica quanto è 'datata' l'informazione relativa ad un peer nella lista locale
- **SelectToKeep**:
 - unisce la vista locale con quella ricevuta
 - restituisce i descrittori **più recenti**, cioè quelli con timestamp maggiore



UN ESEMPIO: IL PROTOCOLLO NEWSCAST

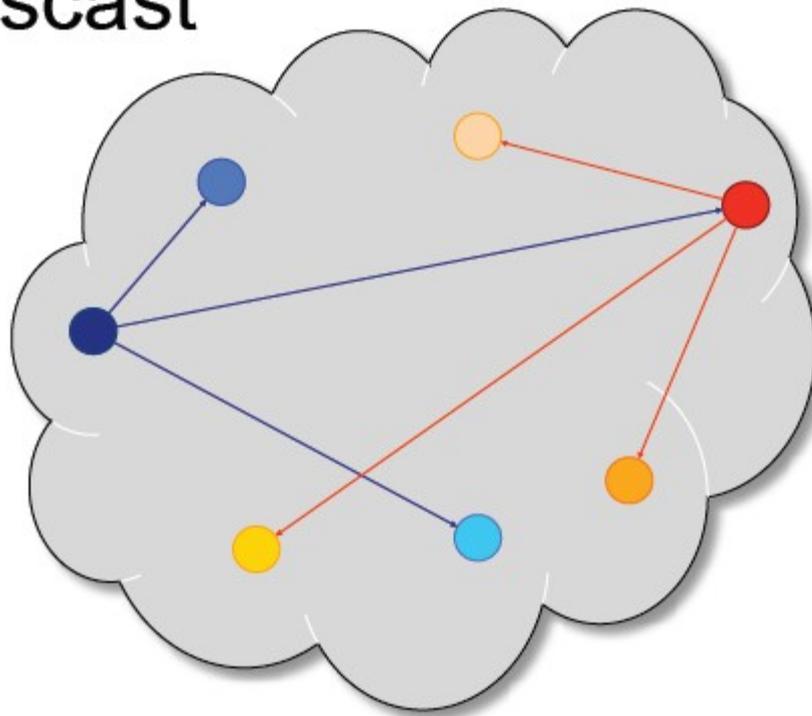
Newscast



UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16

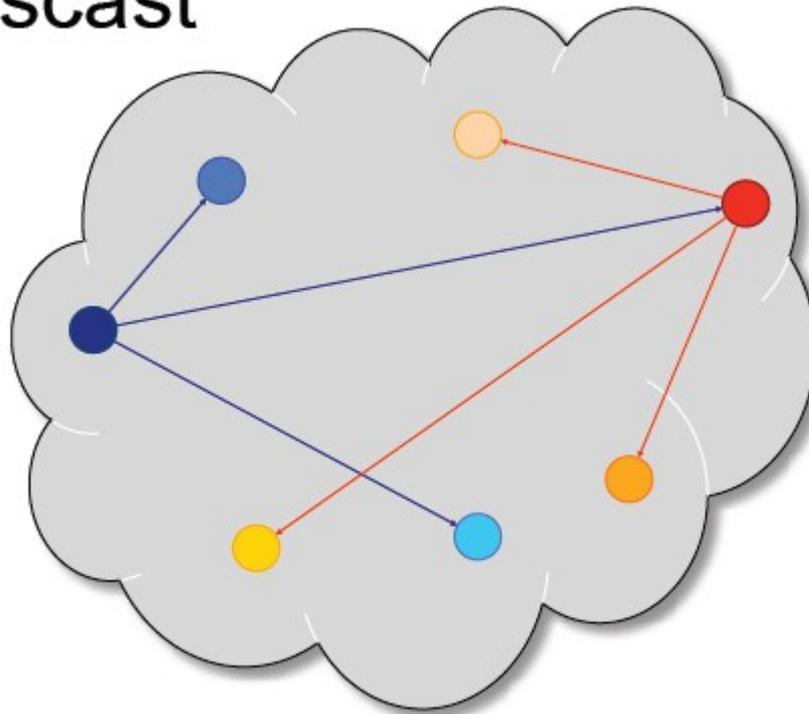


ID & Address	Time stamp
	7
	10
	14

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
● 9	9
● 12	12
● 16	16



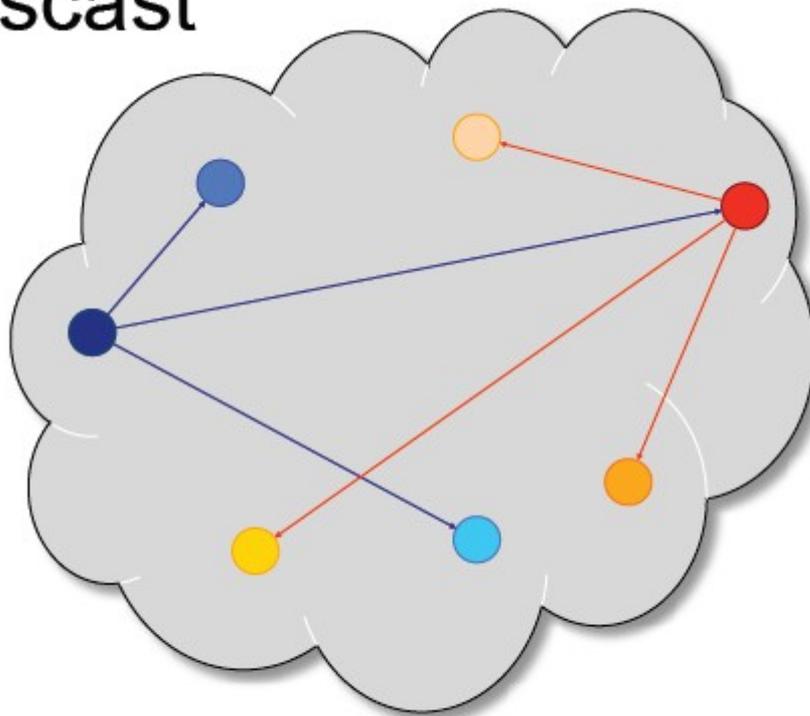
ID & Address	Time stamp
● 7	7
● 10	10
● 14	14

1. Pick random peer from my view

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newsicast

ID & Address	Time stamp
	9
	12
	16



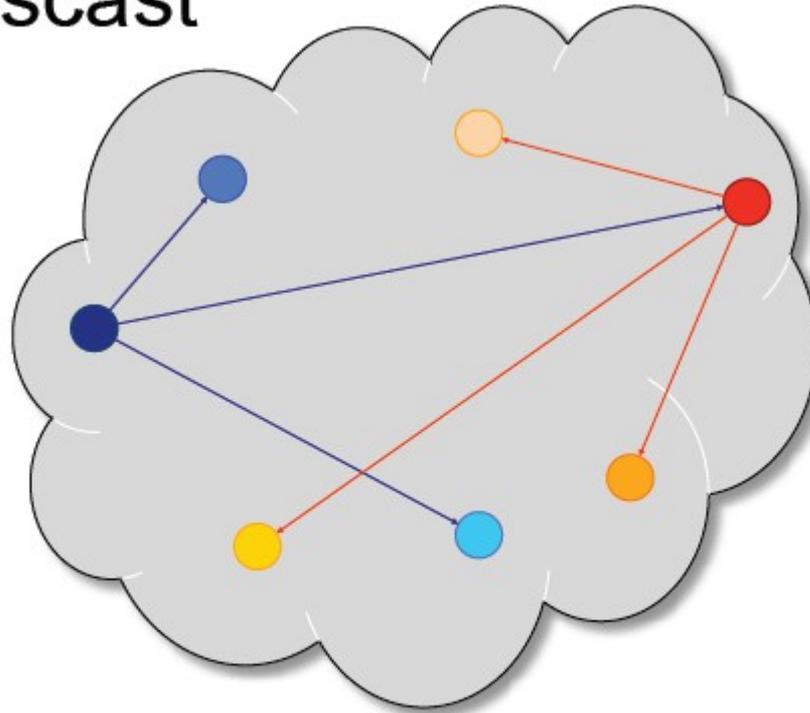
ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16



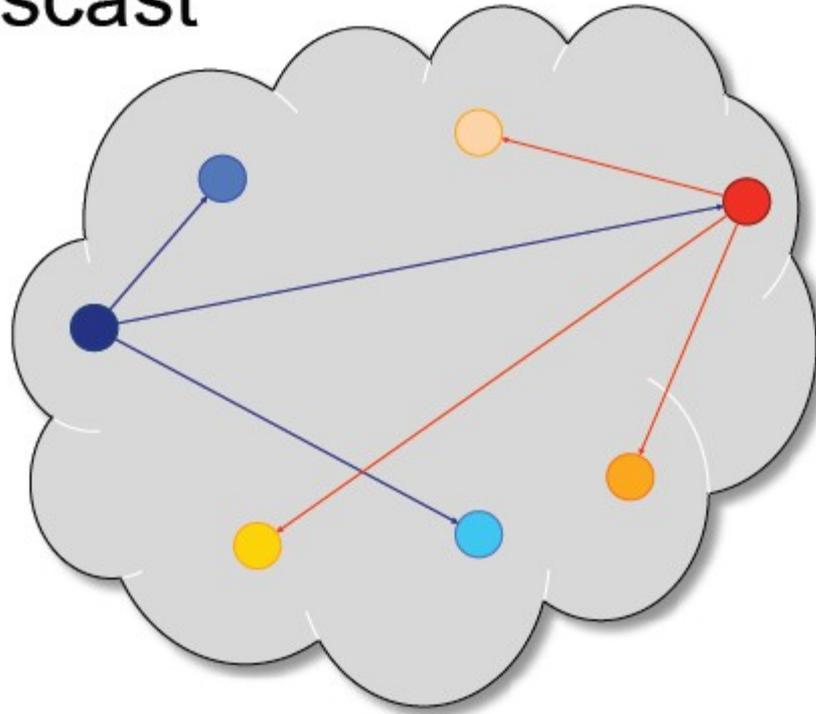
ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view
2. Send each other view + own fresh link

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
● 9	9
● 12	12
● 16	16



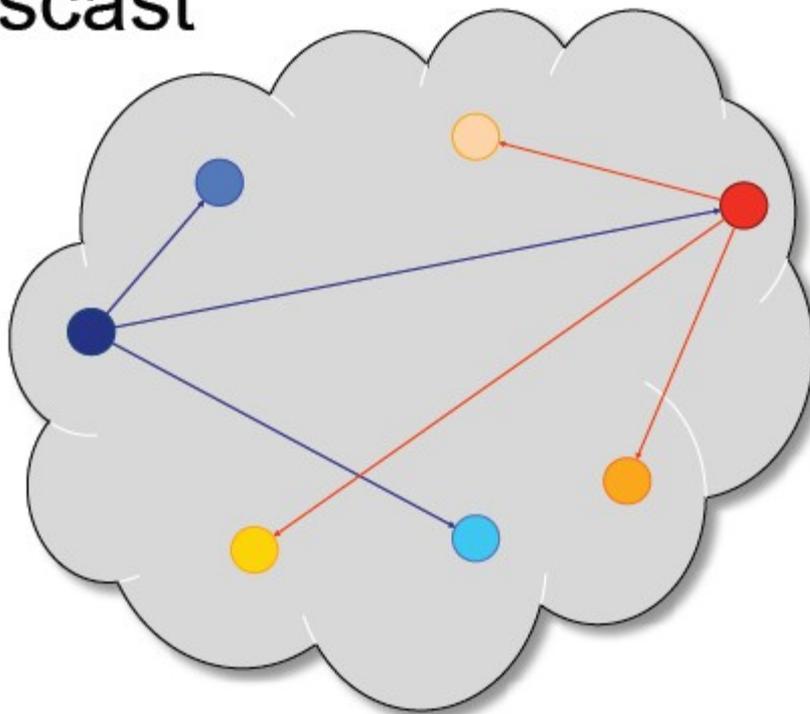
ID & Address	Time stamp
● 7	7
● 10	10
● 14	14
● 9	9
● 12	12
● 16	16
● 20	20

1. Pick random peer from my view
2. Send each other view + own fresh link

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16
	7
	10
	14
	20



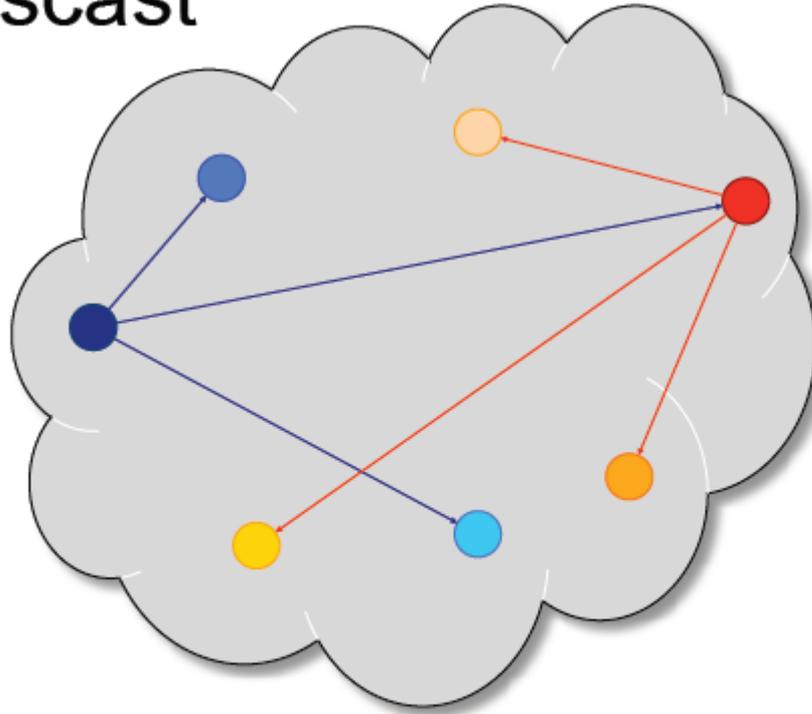
ID & Address	Time stamp
	7
	10
	14
	9
	12
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16
	7
	10
	14
	20



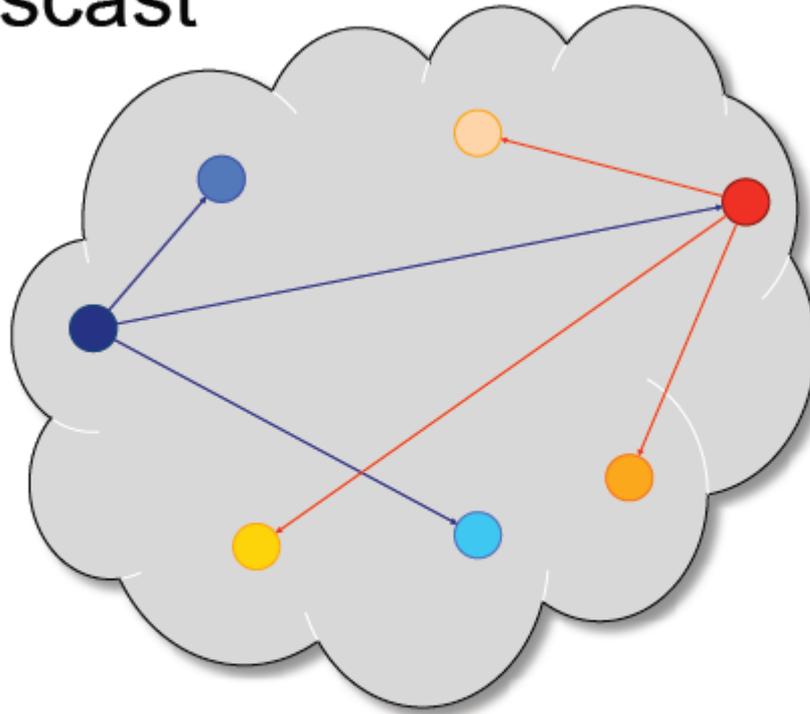
ID & Address	Time stamp
	7
	10
	14
	9
	12
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest links (remove own info, duplicates)

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16
	7
	10
	14
	20



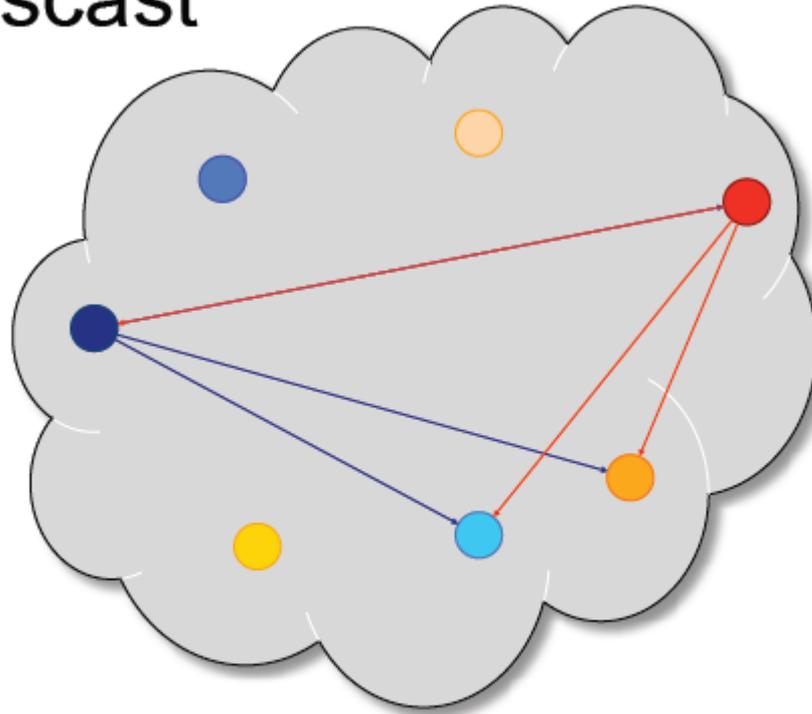
ID & Address	Time stamp
	7
	10
	14
	9
	12
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest links (remove own info, duplicates)

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
Orange	14
Light Blue	16
Red	20



ID & Address	Time stamp
Orange	14
Light Blue	16
Dark Blue	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest link (remove own info, duplicates)

GOSSIP BASED PEER SAMPLING

- Illustriamo ora il protocollo di peer sampling proposto in

Gossip-based Peer Sampling, ACM Transactions on Computers

Jelasyty, Voulgaris, Guerraoui, Kermarrec, Van Steen

- Scopo del protocollo, garantire che le viste parziali dei nodi:
 - contengano i descrittori di un sottoinsieme di nodi variabili dinamicamente
 - riflettano la dinamica del sistema in presenza di churn
- Analizzeremo poi in seguito (dopo aver introdotto gli strumenti necessari per l'analisi di reti complesse) diverse proprietà dell'overlay creato mediante questo protocollo

GOSSIP BASED PEER SAMPLING

- n numero di nodi della rete, ciascun nodo identificato da un indirizzo
- ogni nodo possiede una 'visione parziale' della rete: una **lista locale** contenente **c** descrittori
- descrittore del nodo = < indirizzo nodo, età del nodo >
- operazioni definite sulla vista parziale del nodo:
 - **SelectPeer ()** : restituisce un peer della vista
 - **permute ()** : shuffle casuale dei peer della vista
 - **IncreaseAge ()** : aggiunge 1 all'età di ogni descrittore
 - **Append ()** : aggiunge un insieme di elementi in coda alla vista
 - **RemoveDuplicates ()** : rimuove i duplicati (solito indirizzo), mantenendo le versioni più nuove
 - **removeOldItems (n)** : rimuove gli n descrittori 'più vecchi'
 - **removeHead (n)** : rimuove i primi n descrittori
 - **removeRandom (n)** : rimuove n descrittori scelti in modo casuale

ACTIVE THREAD

```
do forever
  wait(T time units) // T is called the cycle length
   $p \leftarrow$  view.selectPeer() // Sample a live peer from the current view
  if push then // Take initiative in exchanging partial views
    buffer  $\leftarrow$  ( $\langle$  MyAddress,0  $\rangle$ ) // Construct a temporary list
    view.permute() // Shuffle the items in the view
    move oldest H items to end of view // Necessary to get rid of dead links
    buffer.append(view.head(c/2)) // Copy first half of all items to temp. list
    send buffer to  $p$ 
  else // empty view to trigger response
    send (null) to  $p$ 
  if pull then // Pick up the response from your peer
    receive buffer $_p$  from  $p$ 
    view.select(c,H,S,buffer $_p$ ) // Core of framework – to be explained
  view.increaseAge()
```

un active thread per ogni nodo

ACTIVE THREAD

- **View:** vista locale, **Buffer:** buffer da inviare
- effettua uno **shuffle casuale** della vista, dopo aver aggiunto ad essa il proprio descrittore
- sposta i descrittori più vecchi in coda alla vista
- **Nota Bene:** a differenza del protocollo Newscast:
 - ad ogni iterazione aumenta il timestamp dei peer nella vista
 - quindi i descrittori più vecchi ora sono quelli con **timestamp maggiore**
- selezione i primi $c/2$ elementi dalla vista e li inserisce nel buffer da inviare
 - **da tenere a mente:** i primi $c/2$ elementi della vista sono quelli inviati al partner (servirà in seguito)
 -
- **$H \leq c/2 = \text{healing}$:** gli ultimi H elementi non vengono inviati
 - determina quanto il protocollo risulta 'aggressivo' nella eliminazione di links ritenuti obsoleti

PASSIVE THREAD (UNO PER NODO)

do forever

receive buffer_p from p // *Wait for any initiated exchange*

if pull then // *Executed if you're supposed to react to initiatives*

buffer ← (⟨ MyAddress,0 ⟩) // *Construct a temporary list*

view.permute() // *Shuffle the items in the view*

move oldest H items to end of view // *Necessary to get rid of dead links*

buffer.append(view.head(c/2)) // *Copy first half of all items to temp. list*

send buffer to p

view.select(c,H,S,buffer_p) // *Core of framework – to be explained*

view.increaseAge()

VIEW SELECTION

- `view_select(c,H,S,bufferp)` restituisce la nuova 'vista locale' del peer
- Parametri
 - `c` : lunghezza della vista locale
 - `H(Healing)` : numero degli elementi più vecchi spostati in fondo alla lista
 - `S(Swapped)` : numero di elementi da eliminare dalla testa della lista. Controlla quanti degli elementi scambiati con il partner vengono eliminati dalla vista
 - `Bufferp` lista di descrittori ricevuta dal partner

```
method view.select( c, H, S, bufferp )  
  view.append(bufferp) // expand the current view  
  view.removeDuplicates() // Remove by duplicate address, keeping youngest  
  view.removeOldItems( min(H,view.size-c) ) // Drop oldest, but keep c items  
  view.removeHead( min(S,view.size-c) ) // Drop the ones you sent to peer  
  view.removeAtRandom(view.size-c) // Keep c items (if still necessary)
```

PARAMETRI DELL'ALGORITMO

Il comportamento dell'algorithm e quindi le caratteristiche dell'overlay generato, sono determinati da diversi parametri

- scelta del peer con cui effettuare lo scambio delle viste
- strategia push/push pull
- strategia utilizzata per scegliere i peer da inserire nella nuova vista

SELEZIONE DEL PARTNER

- **SelectPeer**() restituisce un peer scelto in modo random dalla vista corrente
- Scelte possibili:
 - **Rand**: sceglie un peer in modo casuale
 - **Tail**: sceglie il peer con cui si è avuto un contatto meno recentemente
 - **Head**: sceglie il peer con cui si è avuto un contatto più di recente
 - i peer che sono stati selezionati recentemente hanno poche possibilità di fornire nuova informazione
 - Questa strategia tende a creare overlay statici

STRATEGIA DI PROPAGAZIONE DELLE VISTE

- **Push:** il peer invia un insieme di descrittori al peer selezionato
- **Pull:** il peer riceve i descrittori dai nodi selezionati
- **PushPull:** il peer ed il peer selezionato si scambiano i descrittori

- vedremo che i risultati sperimentali dimostrano che la strategia Pull, anche se diffonde più rapidamente l'informazione non consente di ottenere risultati soddisfacenti
- un nodo non ha l'opportunità di iniettare nella rete il suo descrittore
 - contatta gli altri e riceve le loro informazioni, ma non propaga il suo descrittore
 - deve aspettare di essere contattato dagli altri
 - la perdita di connessioni in entrata isola il nodo dalla parte rimanente della rete
- in generale, si considerano solamente strategie di tipo **push/pushpull**

SELEZIONE DI UNA NUOVA VISTA

Parametri critici in view.select: c , H , S

Si assuma, per semplicità, c pari

- dopo aver aggiunto alla propria vista i descrittori appartenenti alla vista del vicino, il numero dei descrittori nella vista locale è $c+c/2$. (c vecchia vista + $c/2$ descrittori ricevuti dal partner)
- devo ridurre la vista risultante, elimino:
 - gli H più vecchi
 - ricordare: incima alla lista ci sono i descrittori che ho scambiato con il vicino
 - elimino S descrittori in cima alla lista
 - maggiore è S più rinnovo la vista

SELEZIONE DI UNA NUOVA VISTA

Scelte possibili per definire la nuova vista

- **Blind**: $H=0, S=0$ i descrittori scelti in modo random
- **Healer (guaritore)**: $H=c/2, S=0$, si selezionano i descrittori più recenti
- **Swapper** : $H=0, S=c/2$, si eliminano i descrittori scambiati con il vicino, si favorisce il 'rinnovo continuo' dei links

Healer:

- eliminando i descrittori 'più vecchi' si cerca di eliminare i links a peer che probabilmente non sono più presenti sulla rete

SELEZIONE DI UNA NUOVA VISTA

Swapper

- i primi $c/2$ descrittori della lista sono quelli scambiati con il partner
- dopo aver eliminato dalla vista i descrittori più vecchi, si eliminano i primi S descrittori
- S = swap, il parametro controlla il numero di descrittori scambiati tra i due peer
- Il parametro S controlla la priorità data ai descrittori ricevuti dal partner
 - un valore alto di S corrisponde ad una probabilità alta di inserire i descrittori ricevuti nella nuova vista
 - un valore basso di S consente di mantenere nella propria vista molti dei valori scambiati con l'altro peer. Come conseguenza, le viste dei due peer diventano simili