Mobile and Social Sensing Systems

Autore: Gabriele Frassi

Università di Pisa
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Artificial Intelligence and Data Engineering

Appunti sulla parte del prof. Marco Avvenuti

SOMMARIO DISPENSA

1 UNIMAP

1. Mer 28/02/2024 15:30-18:30 (*3:0 h*) lezione: Course general overview and concepts. Wireless (non-GPS) Localisation in Cyber-Physical-Systems: definition, taxonomy, approaches. Anchor-based localization: range-based, range-free approaches. (MARCO AVVENUTI)
2. Mer 06/03/2024 15:30-18:30 (*3:0 h*) lezione: Range-based localization techniques: RSSI,TDoA,AoA. Range-free localization techniques: Centroid, APIT, DV-HOP. Discussion. (MARCO AVVENUTI)
3. Mar 12/03/2024 10:30-12:30 (*2:0 h*) lezione: Humans as Sensors: participatory and opportunistic crowdsensing, the HaS paradigm, the Reliable Sensing Problem. A Binary Model of Human Sensing: Unknown Reliability, Binary Observations, Uncertain Provenance. (MARCO AVVENUTI)
4. Mar 19/03/2024 10:30-12:30 (*2:0 h*) lezione: Humans as Sensors: Estimation Problem. Example. Methodology. Validation Method. Precision Evaluation. Observations and Limitations. (MARCO AVVENUTI)
5. Mer 15/05/2024 16:30-18:30 (*2:0 h*) lezione: Distributed Hash Table. The Publish-Subscribe problem, centralized vs Peer-to-peer architectures. Distributed Hash Table: properties, structure, operations, applications, API. Pastry: overlay network, prefix routing, routing table (MARCO AVVENUTI)
6. Mar 21/05/2024 10:30-11:30 (*1:0 h*) lezione: Pastry: routing table, node arrival/departure, API. Example. (MARCO AVVENUTI)

2 INTRODUZIONE

Il corso affronta la tematica dei sensori da due punti di vista:

- **Punto di vista tradizionale, ma pur sempre innovativo**.
  Dispositivi fisici dotati di sensori che captano informazioni.
- **Punto di vista nuovo**.
  Utilizzare i social network come reti di sensori, dove i sensori sono esseri umani (*human sensors*).

2.1 Punto di vista tradizionale, ma pur sempre innovativo

Abbiamo dispositivi mobili che sono alimentati da batterie (dai più classici ai più recenti dispositivi indossabili). Due aspetti tipicamente li caratterizzano:

- un collegamento wireless alla rete (una LAN/WAN o addirittura Internet);
- il dialogo con una infrastruttura centrale.

Questi dispositivi, soprattutto quelli indossabili, sono caratterizzati da una vasta gamma di sensori. L'uso maggiore e continuo (durante il giorno) di questi dispositivi apre a nuove possibilità. I dispositivi hanno autonomia maggiore, sono molto più piccoli e collocati in posti come maglie, corpo umano, automobili case… Si pensi all'ingente quantità di dati raccolti, che possono andare ad allenare un modello di machine learning. Si parla di ***pervasive/ubiquitous computing***:

> *The purpose of a computer is to help you do something else (beyond computing)*
> *The best computer is a quiet, invisible servant*
> *The more you can do by intuition the smarter you are*
> *The computer should extend your unconscious*
> *Technology should create calm*
>
> **Mark Weiser**
> **The computer for the 21th century**
> **1991**

- Due aspetti caratterizzano il *pervasive computing*:
  - creazione di un ambiente dove la connettività è sempre disponibile e non è ostruita;
  - uso su larga scala di tecnologie per creare un ambiente con *computing and communication capabilities*, tecnologie che risultano perfettamente integrate con l'uomo e con l'ambiente.
- **Coperta corta**. Un *pervasive computing system* deve essere anche *context-aware*, cioè deve conoscere lo stato di utenti e ambiente circostante per modificare il proprio comportamento e arricchire le risposte agli utenti. Fare ciò è difficile: non si concilia col minimizzare l'intrusività, è difficile gestire la molteplicità di aspetti che definiscono l'utente (posizione fisica, stato fisiologico, stato emozionale, comportamenti tipici, …).
- **Altro problema**. L'altra questione è essere in grado di estrarre dati affidabili per mezzo dei sensori.

2.2 Punto di vista nuovo

I social network possono essere considerati a tutti gli effetti reti di sensori, dove i sensori sono gli esseri umani (*human sensors / social sensors*): sono fonti di informazioni sugli utenti nello specifico e sull'ambiente da essi frequentanti in generale.

- Il concetto di pervasività rimane valido in quanto tali piattaforme sono utilizzate soprattutto per mezzo di dispositivi mobili.
- I dati riguardano qualunque tipologia di argomento e sono ingenti in quantità (situazioni emergenziali, politica…).
- Problema: dati maggiori in quantità e contenuto, ma aumenta l'inaffidabilità a differenza dei sensori tradizionali. Parleremo di come questi dati sono raccolti (*social media crawling*) e come vengono analizzati. Analizzeremo i cosiddetti comportamenti coordinati inautentici (*Coordinated Inauthentic Behaviors*).

3 WIRELESS LOCALIZATION TECHNIQUES

3.1 Definizione di *Cyber-Physical-Systems* (CPS)

Un Cyber-Physical-Systems è un sistema caratterizzato da reti e sistemi embedded.

- È caratterizzato da un componente fisica e una componente software.
- Queste due componenti sono profondamente legate tra loro e interagiscono alterando il *contesto*.
- Abbiamo un *"a computer system in which a mechanism is controlled or monitored by some algorithms".*

3.2 Definizione di Wireless Sensor Networks (WSN)

Una Wireless Sensor Network è l'esempio principale di *Cyber-Physical-System*.

- È costituita da un numero significativo di nodi, ciascuno caratterizzato dalla presenza di sensori.
- I nodi sono connessi per mezzo di una connessione wireless.
- Ciascun nodo è caratterizzato da una ricetrasmittente (per comunicare con gli altri nodi, sia ricevere che trasmettere informazioni), un microcontrollore e in generale un'interfaccia per interloquire coi sensori, una fonte di energia (solitamente una batteria)
- Ogni nodo monitora l'ambiente in cui è posizionato: raccoglie informazioni (Esempio: temperatura, suono, livelli di inquinamento, umidità, vento…) e le trasmette a una *base station*.

La batteria e gli algoritmi eseguiti sui nodi sono congeniati per massimizzare la durata dell'autonomia del nodo (pensiamo che in alcuni contesti cambiare la batteria potrebbe essere molto difficile).

3.3 Localization: concetto

Un aspetto fondamentale di queste applicazioni è la localizzazione. Il punto fondamentale non è solo sapere dove si trova il dispositivo in un particolare istante temporale, ma associare tali informazioni (istante temporale e collocazione) a particolari eventi.

Localization is the ability to determine the positions of Components, Sensors and Events.

La localizzazione permette la creazione di *location-aware services* su un'applicazione:

- collocazione di prodotti all'interno di un magazzino;
- collocazione di personale e attrezzatura medica all'interno di un ospedale;
- collocazione del vigile del fuoco all'interno di una struttura in fiamme;
- determinazione di monumenti/negozi più vicini che potrebbero interessarci;
- …

Ma anche di supportare *network services*:

- geographical routing, dove i messaggi vengono trasmessi indicando come destinatario una posizione geografica e non un indirizzo di rete;
- valutazione della qualità della copertura della rete;
- tracciare oggetti;
- riportare il luogo in cui specifici eventi sono avvenuti;
- …

3.4 Tassonomia per la localization

Un sistema di localizzazione è caratterizzato da due blocchi principali:

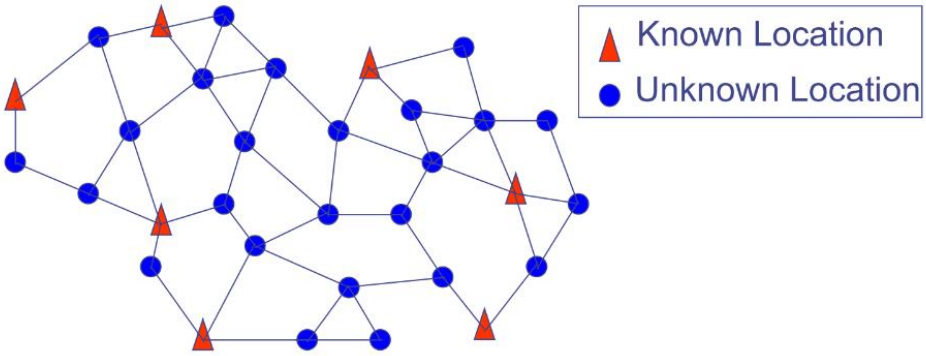
- un set di nodi distribuiti, e
- un algoritmo di localizzazione.

Ogni nodo si trova in uno stato:

- ***unknown***. Nodo che non conosce alcuna informazione sulla sua posizione geografica.
- ***beacon***. Nodo (detto anche *anchor*) che conosce già la sua posizione in virtù di un posizionamento manuale (nel senso che il dispositivo è stato collocato in un certo posto e il software è progettato tenendo conto di ciò) o per calcolo attraverso il sistema GPS.

- ***settled.*** Nodi inizialmente nello stato unknown che hanno acquisito (o semplicemente stimato) la propria posizione.

Al lancio dell'algoritmo di localizzazione un nodo si trova nello stato *unknown* o *beacon*. Lo scopo dell'algoritmo di localizzazione è individuare la posizione dei nodi in *unknown state*. All'interno di questi algoritmi possono avere un ruolo fondamentale i *beacon nodes* e i *settled nodes* (li prendiamo come riferimenti, in un certo senso).



Il GPS (Global Positioning System) è la soluzione più intuitiva, ma non è detto sia la più valida: è costosa da un punto di vista energetico, non funziona in certi ambienti (ad esempio al chiuso).

Questo ci porta all'analisi di tecniche che fanno uso della comunicazione wireless per determinare la posizione, senza ricorrere al GPS.

3.5 Topologia di un sistema di localizzazione

Quattro topologie possibili:

- ***Remote positioning***.
  La posizione del nodo è calcolata presso un dispositivo centrale (*base station*). Il nodo trasmette segnali radio, questi vengono captati dagli anchor nodes e inoltrati da questi al dispositivo centrale. Il dispositivo centrale sfrutta questi segnali inoltrati per calcolare la posizione.
- ***Self-positioning***.
  La posizione di un nodo è calcolata dal nodo stesso, sfruttando i segnali che riceve dagli anchor nodes.
- ***Indirect remote positioning***.
  La posizione di un nodo è calcolata dal nodo stesso, come prima, ma la posizione è successivamente inviata a un dispositivo centrale attraverso un *wireless back channel*.
- ***Indirect self*-positioning**.
  La posizione del nodo è calcolata presso un dispositivo centrale, ma la posizione è successivamente inviata al nodo attraverso un *wireless back channel*.

3.6 Centralized VS Localized

Nelle topologie appena introdotte si distinguono due approcci:

- ***Centralized approach***.
  Un dispositivo centrale stima la posizione degli unknown nodes utilizzando i segnali inoltrati dagli anchors. L'accuratezza è maggiore, ma il sistema non è scalabile. Se il numero di anchors non è sufficiente è necessario introdurre una comunicazione multi-hop.
- ***Localized approach***.
  Ogni nodo stima la sua posizione sfruttando i segnali ricevuti dagli anchor nodes vicini e le informazioni relative agli anchor nodes. Molto più semplice rispetto all'approccio centralizzato, ma l'esecuzione di tali computazioni sul nodo comporta maggior consumo di energia.

3.7 Sistemi di coordinate

Quattro tipologie di coordinate:

- **Coordinate assolute**.
  La posizione è determinata rispetto agli *anchor nodes*, in generale nodi che conoscono la loro posizione.
- **Coordinate relative**.
  La posizione è determinata rispetto a nodi diversi dagli *anchor nodes /absolute nodes* citati parlando di coordinate assolute.
- **Coordinate fisiche**.
  Rappresentazione del dispositivo come un punto collocato all'interno di un sistema di coordinate a due o tre dimensioni. Esempi: Universe Transverse Mercator (UTM) o Degree/Minutes/Seconds (DMS, usato in GPS e basato su longitudine e latitudine).
- **Coordinate simboliche**.
  Rappresentazione della posizione del dispositivo per mezzo di informazioni come il numero dell'ufficio, l'edificio o il nome della strada.

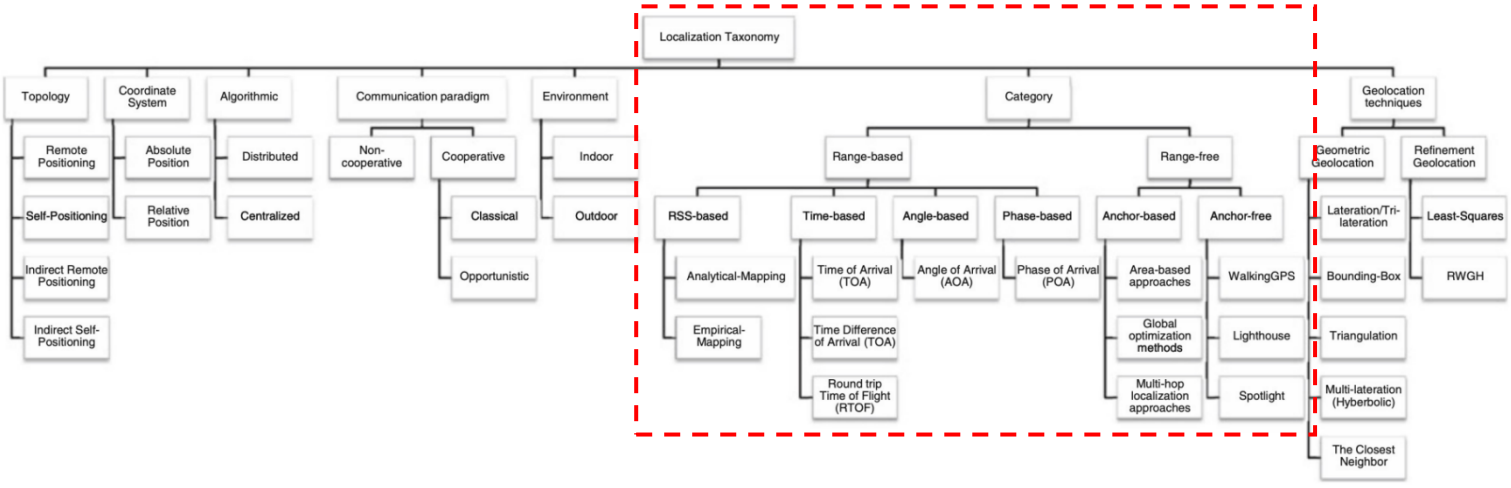
3.8 Paradigmi di comunicazione

Gli algoritmi di localizzazione si basano sulla comunicazione tra nodi. Quali sono i paradigmi possibili?

- ***Paradigma non-cooperativo***.
  La comunicazione avviene unicamente tra un nodo in stato unknown e gli anchor nodes. Nodi in unknown state non dialogano tra di loro. Si scarica la complessità sugli anchors, ma affinchè il paradigma funzioni correttamente è necessaria un'elevata densità di ancor o una trasmissione a lungo raggio affinchè l'unknown node si trovi nel raggio di comunicazione di almeno tre anchors.
- ***Paradigma cooperativo***.
  La comunicazione avviene anche tra unknown nodes. La complessità è scaricata sugli unknown nodes.
  - ***Paradigma opportunistico***.
    Forma particolare di paradigma cooperativo. La comunicazione avviene con i nodi che passano in prossimità, sfruttando le interazioni che questi hanno con altri nodi.

3.9 Schema e principali tipologie di algoritmi di localizzazione

Introduciamo le tipologie di algoritmi di localizzazione:



- ***Range-based algorithms***.
  Gli algoritmi range-based prevedono il calcolo di distanza e angolo per la localizzazione del nodo. Tecniche accurate, ma costose.
- ***Range-free algorithms***.
  Gli algoritmi range-free non prevedono il calcolo di distanza e angolo per la localizzazione. Fanno affidamento a prossimità e informazioni sulla connettività per individuare la posizione del dispositivo. Tecniche decisamente meno costose, ma anche meno precise.

3.10 Metriche per valutare la performance degli algoritmi di localizzazione

- **Accuracy**.
  Errore medio della distanza tra la posizione stimata e la posizione reale.
- **Precision**.
  Variazione della performance dell'algoritmo in varie esecuzioni.
- **Complexity**.
  Questioni legate all'hardware, al software, alle operazioni svolte.
- **Robustness**.
  Capacità del sistema di lavorare quando alcuni segnali non sono disponibili.
- **Scalability**.
  La performance dell'algoritmo degrada quando la distanza tra trasmettitore e ricevitore aumenta. Due assi: geografia e densità (Cosa succede se colloco i nodi in posizioni diverse? Cosa succede se aumento la densità dei nodi?).
- **Cost**.
  Costi infrastrutturali necessari.

3.11 Range-based methods

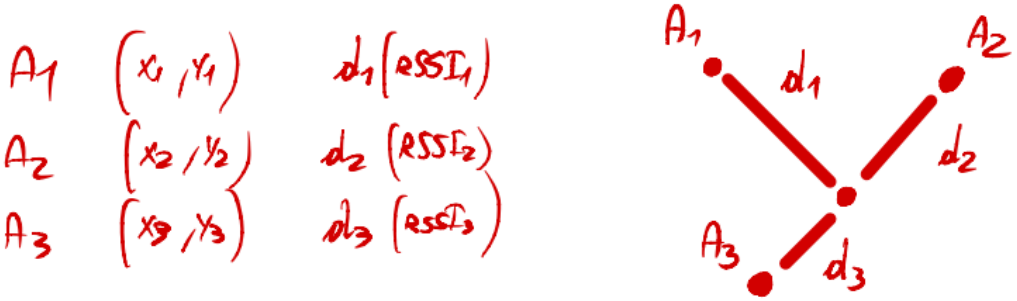
3.11.1 Fasi che caratterizzano questi algoritmi

Gli algoritmi range-based permettono la stima della posizione in due fasi:

- ***Ranging phase***
  Il nodo calcola la sua distanza e angolo dagli anchors vicini.
- ***Estimation phase***.
  Il nodo utilizzare le informazioni individuate al passaggio precedente e quelle degli anchors per stimare la propria posizione (*self-positioning method*, sì).

3.11.2 Ranging phase

La prima fase consiste nel calcolo della distanza, ed eventualmente dell'angolo, rispetto agli *anchors* vicini. I dati ottenuti saranno utilizzati nella *estimation phase* per determinare la posizione del nodo.



La distanza è calcolata rispetto agli anchors vicini. Attenzione all'area considerata: se gli anchors A_1, A_2, A_3 "delimitano" un'area grande allora l'accuratezza sarà minore (e quindi l'errore maggiore).

- **Received Signal Strength (RSS)**.
  I nodi stimano la distanza dagli anchors utilizzando la potenza del segnale ricevuto, che risulta attenuato rispetto alla potenza del segnale emesso (proprietà della fisica, come sappiamo da Reti Informatiche). Calcoliamo l'RSS con la seguente formula matematica

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2} \Leftrightarrow \boxed{P_{RSSI} = \frac{X}{r^n}}$$

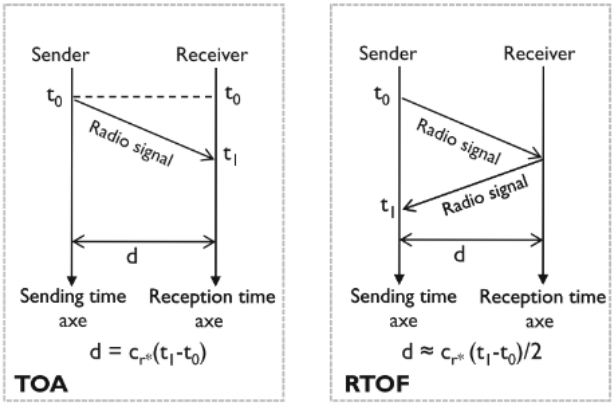
  La potenza del segnale ricevuto è inversamente proporzionale al quadrato della distanza, il che non ci sorprende. Inoltre:
  - P_t è la potenza del segnale trasmesso;
  - G_t e G_r sono costanti di guadagno, rispettivamente del trasmettitore e del ricevitore (dipendono dalle antenne del sistema);

- λ è un'altra costante.

L'idea è di calcolare d a partire da $P_r(d)$. Purtroppo questa formula è solo teorica: *Due to multipath fading and shadowing (reflection, refraction, diffraction, scattering) present in the indoor environment, background interference, irregular signal propagation, path-loss models do not always hold*. Altra questione rilevante è che questo modello è site-specific: abbiamo delle costanti che possono essere ottenute solo misurando il sito dove è presente il nostro sistema.

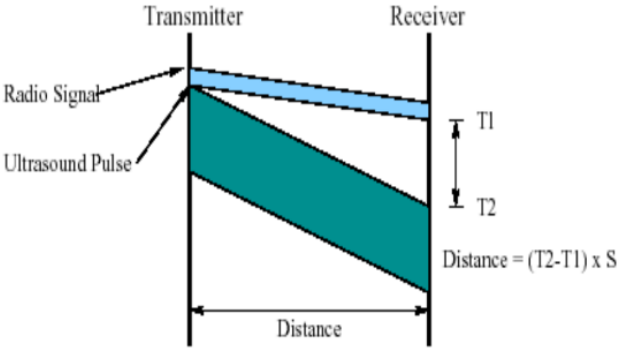
Technique	Advantages	Limitations
Analytical-mapping models	Simple to implement Useful for simulators design	Parameters are environment-dependent Coarse accuracy
Empirical-mapping models	Can achieve high accuracy level	Need extensive environment profiling High off-line computation overhead Poor scalability Unreliable if the environment is continually changing

- **Time of Arrival (ToA)**.
  - Gli anchors trasmettono al nodo un messaggio che indica l'istante temporale in cui tale messaggio è stato trasmesso. Il nodo riceve questo segnale e per mezzo di una differenza trova il ToA.
  - Calcolato il ToA e conosciuta la velocità di propagazione c (velocità della luce, supponendo di utilizzare i segnali radio) del segnale possiamo trovare la distanza: $d = c \cdot \mathrm{ToA}$.
  - Problema: è necessario che il clock del trasmettitore e del ricevitore siano sincronizzati adeguatamente, piccoli errori nel ToA potrebbero dare luogo a errori di centinaia di chilometri!
  - Esempio: GPS, basato su una costellazione di almeno 24 satelliti con *atomic clocks*, trilateration.
  - Altra via: RTT (Round Trip Time).



- **Time Difference of Arrival (TDoA)**.

Qualora non sia possibile avere una sincronizzazione precisa si può ricorrere al TDoA. In questo metodo non è richiesta la presenza del tempo di invio all'interno del messaggio trasmesso dall'anchor node.



- Si sfrutta la differenza temporale tra la ricezione del segnale trasmesso e gli ultrasuoni (quindi si fa riferimento non alla velocità della luca c, ma alla velocità del suono s – cosa buona perché l'errore diventa molto più piccolo rispetto a ToA).
- Individuati gli istanti temporali di ricezione procediamo col seguente calcolo

$$d = s \cdot (t_2 - t_1)$$

La cosa più vantaggiosa è sicuramente la precisione e la non rilevanza della sincronizzazione tra clock. Problemi: necessità di hardware aggiuntivo (per gestire il canale ultrasuoni) e conseguentemente maggiore consumo energetico.

- **Angle of Arrival (AoA)**.
  Gli anchors trasmettono beacons al nodo e questo misura l'angolo da cui arrivano i messaggi: si parla di *Angle of Arrival*.
  - Tre nodi necessari per una misurazione 3D, due per una misurazione 2D.
  - Necessario hardware aggiuntivo: un array di antenne o un'antenna direzionale.
  - Non è necessaria *time synchronization* (vantaggio significativo rispetto a Time of Arrival).
  - *Location derived from the intersection of several pairs of angle direction lines*, come nella figura a sinistra

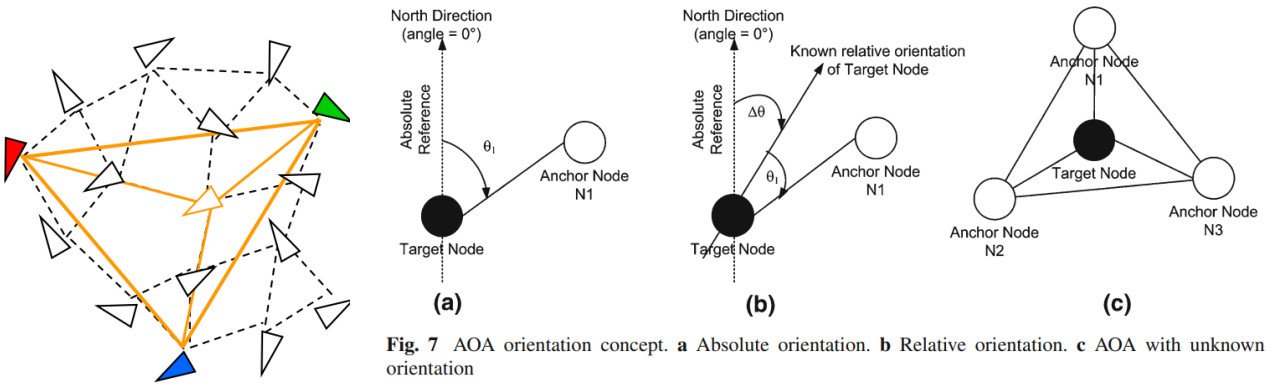


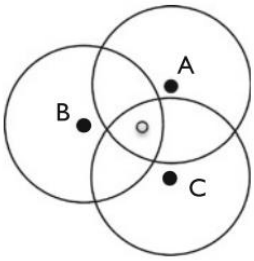
Fig. 7 AOA orientation concept. **a** Absolute orientation. **b** Relative orientation. **c** AOA with unknown orientation

So, assuming that we can mount that extra hardware on our nodes, the methods rely on the angle of arrival, which can be estimated by using the index of the antennae of the specific message (if we have many antennas pointed on many different directions, at the receiving node we switch on the right one in the array and that gives us the receiving angle, which we can employ in geometric methods to, for example, draw triangles, or use trigonometry). It is not easy to see such a solution, because it requires additional hardware and is very impractical when we have a large sensor network.

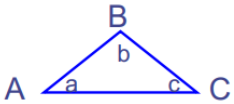
3.11.3 Estimation phase

Determinata la distanza possiamo procedere nel determinare la posizione! Possibili tre metodi:

- **Trilaterazione**.
  Si determina la collocazione del nodo misurando la sua distanza da tre riferimenti (gli anchors). Tali distanze sono state ottenute nella *ranging phase*. Per ogni anchor node si disegna una circonferenza avente per raggio la distanza rispetto al target node. A quel punto è possibile determinare la posizione del target node sfruttando le posizioni (note) dei tre *anchor nodes*.



- **Triangolazione**.
  Si misurano gli angoli tra i tre nodi di riferimento (gli anchors). Tali angoli sono ottenuti nella *ranging phase*. Anche qua calcoliamo la posizione del target node a partire dalle posizioni (note) dei tre anchor nodes. Minore è la dimensione del triangolo, maggiore sarà accurata la stima.

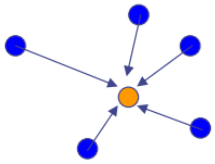


Sines Rule $\frac{A}{\sin a} = \frac{B}{\sin b} = \frac{C}{\sin c}$

Cosines Rule
$C^2 = A^2 + B^2 - 2AB\cos c$
$B^2 = A^2 + C^2 - 2BC\cos b$
$C^2 = B^2 + C^2 - 2BC\cos a$

.

- **Multilaterazione**.
  Generalizzazione dei metodi precedenti: si sfruttano tutti i *beacon node* presenti, non solo tre. Utilizzando in sistemi caratterizzati da elevata densità di beacons: maggiore è il numero di beacon maggiore è l'accuratezza della stima.



3.11.4 Limiti

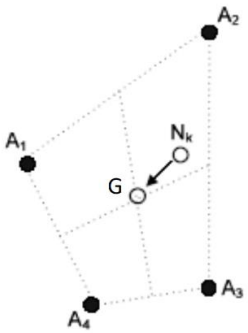
Questi metodi hanno dei limiti importanti:

- Il radio range non è perfettamente circolare a causa dell'ambiente
- Non è detto che la comunicazione sia simmetrica: se io ricevo un messaggio da un anchor non posso assumere, ahimè, che se invio un messaggio di risposta l'anchor lo riceva.
- Necessario hardware aggiuntivo.
- Non devono essere presenti ostacoli
- *we need a clear line of sight;*
- *no multipath and flat terrain, which introduce noise and problems*

3.12 Range-free methods

Negli algoritmi range-free non si calcola la distanza o l'angolo per determinare la posizione del nodo. Introduciamo i seguenti approcci, che sono decisamente più economici e con minori requisiti di hardware, ma meno precisi dei metodi precedenti.

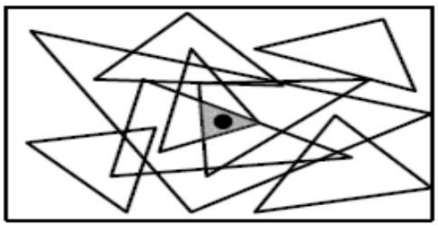
- ***Area-based approaches***.
  - ***Centroid Localization***
    Gli anchors rappresentano i vertici di un poligono. Un nodo presente all'interno di questo poligono riceve dagli anchor un messaggio contenente la loro posizione: sfruttando tali dati calcola la sua posizione come centroide/baricentro del poligono.



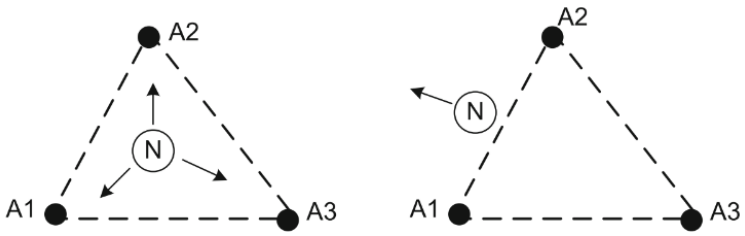
$$(x_g, y_g) = \left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n}\right)$$

    - Il numero di anchors non è fissato. La precisione dipende soprattutto dalla dimensione dell'area del poligono. Possibile intersezione tra più poligoni.
    - Semplice da implementare, non complesso da un punto di vista computazionale.
    - Poco accurato e si basa sull'assunzione che il nodo si trovi all'interno dei range di trasmissione degli anchors.

  - ***APIT*** (*Approximate Point-In-Triangle test*)
    Approccio simile al precedente dove le aree consistono in triangolo. La rete risulta divisa in triangoli e la posizione del nodo consiste nel centroide di un triangolo, definito come l'intersezione dei triangoli formati dagli anchors dove risiede il nodo. Risultati sperimentali confermano che questo metodo è molto più preciso del precedente e che l'errore rimane contenuto all'aumentare della densità dei nodi. Si segnala un basso overhead (buono)!



    Il *Point-In-Triangulation Test* (PIT) determina se il nodo, di cui non conosciamo la posizione, si trova all'interno del triangolo formato dai tre anchors A1, A2, A3 o all'esterno



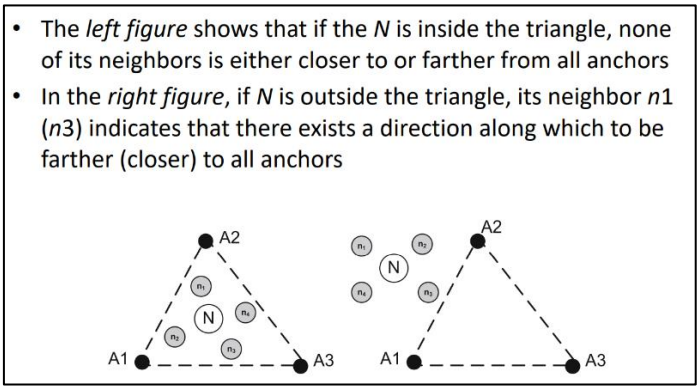
L'algoritmo eseguito da ogni nodo ha la seguente forma

Receive location beacons (X_i, Y_i) from N anchors;
$InsideSet = \emptyset$;
for each triangle $T_i \in \binom{N}{3}$ triangles do
  if Point-In-Triangle-Test (T_i) == TRUE then
    $InsideSet = InsideSet \bigcup T_i$;
  end if
end for
Estimated Position = CenterOfGravity($\bigcap T_i \in InsideSet$);

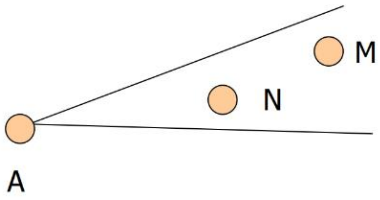
Approfondiamo il PIT Test!

- **PIT Test perfetto (non possibile)**.
  Possiamo compiere un test immaginando di muovere il nodo lungo tutte le possibili direzioni. Facciamo ciò considerando quanto segue:
  - se il nodo è dentro il triangolo allora questo, muovendosi in qualunque direzione, si avvicinerà ad almeno uno dei tre anchors;
  - se il nodo è esterno al triangolo allora esiste una direzione che, se intrapresa dal nodo, lo porterà ad allontanarsi da tutti e tre i nodi allo stesso tempo.

  Le proposizioni sono giuste, ma il test non è possibile: i nodi non si muovono tipicamente e non è possibile eseguire tutte le possibili direzioni.
- **PIT Test approssimato**.
  A causa di quanto detto si introduca una versione approssimata del PIT Test.
  - Il nodo chiede ai suoi vicini (non agli anchors) di trasmettere la loro distanza rispetto a tre anchor nodes. Il nodo computa anche la sua distanza rispetto agli anchor nodes.
  - Ricevute le distanze dai nodi vicini fa un confronto con le proprie distanze. Il nodo si trova all'interno del triangolo se, prese le distanze di un qualunque nodo vicino rispetto agli anchor nodes, osserviamo che:
    - non esistono nodi vicini dove tutte e tre le distanze sono simultaneamente maggiori delle relative distanze del target node;
    - non esistono nodi vicini dove tutte e tre le distanze sono simultaneamente minori delle relative distanze del target node.
  - Se quanto detto non è valido allora il nodo è fuori dal triangolo.



Rispetto alla distanza si utilizza la potenza del segnale ricevuto (RSS, *Received Signal Strength*) che in assenza di ostacoli cala all'aumentare della distanza tra ricevitore e trasmettitore



RSS(A,M) < RSS(A,N)

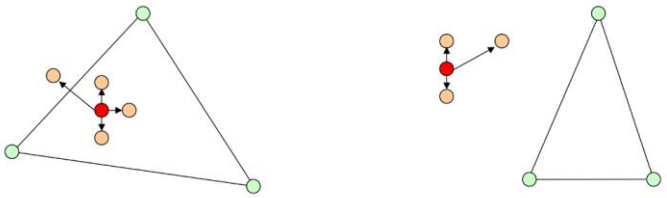
DIST(A,M) > DIST(A,N)

Per rappresentare l'area ottenuta dall'intersezione dei triangoli utilizziamo un algoritmo *grid scan*:

- l'area dove sono presenti i nodi è divisa in celle, ciascuna rappresentata da un valore numerico (inizialmente zero per tutte le celle);
- le aree dei triangoli sono associate alle celle dette prima;
- ogni volta che il PIT segnala che un nodo è all'interno di un certo triangolo incrementiamo di 1 le relative celle, se invece non è presente decrementiamo di 1;
- le celle coi valori più grandi rappresentano l'area all'interno della quale è posizionato il nodo.

0	0	0	0	0	0	1	0	0	0
0	0	1	0	1	1	1	0	0	0
0	0	1	1	1	1	1	0	0	0
0	1	2	2	1	1	0	-1	0	0
1	1	2	2	1	1	0	-1	-1	0
0	0	2	2	2	1	0	-1	-1	-1
0	0	1	1	1	0	0	-1	-1	-1

In tutto questo dobbiamo stare attenti ai possibili errori:



In-to-out error

Out-to-in error

- *in-to-out error*, il nodo è all'interno del triangolo e viene segnalato come esterno al triangolo (causa: uno dei nodi coinvolti è esterno al triangolo, come evidente in figura)
- *out-to-in error*, il nodo è all'esterno del triangolo e viene segnalato come interno al triangolo (causa: il nodo è più lontano dal triangolo di tutti i suoi vicini)

- ***Multihop localization approaches***

Se il potere di trasmissione è insufficiente potrebbe essere impossibile avere all'interno del range di trasmissione tre anchor node per costruire un triangolo. Segue che il metodo precedente è insufficiente! Vogliamo estendere la localizzazione oltre il range di trasmissione, a tal proposito introduciamo meccanismi di localizzazione multihop.

  - **DV-HOP (*Distance Vector – HOP*).**
    - **Diffusione dei dati all'interno della rete**.
      - Ogni anchor node trasmette indipendentemente dei beacon con all'interno la propria posizione e un hop-counter che inizialmente è settato ad 1 e

viene incrementato ad ogni hop (cioè ogni volta che il beacon passa da un altro nodo e viene a sua volta inoltrato).

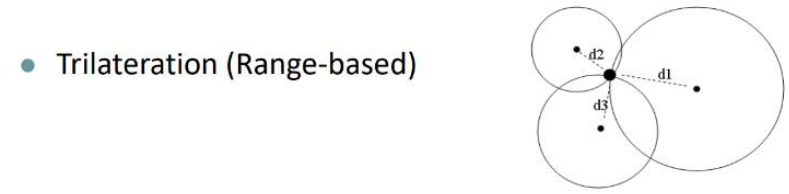
- Il nodo che deve calcolare la sua posizione riceve questi beacon e sfrutta i dati ricevuti per determinare lo shortest-path per raggiungere ogni anchor node.
    - Quando un nodo N riceve un beacon da un anchor A_i memorizza il record $\langle X_i, Y_i, h_i \rangle$, dove X_i, Y_i è la posizione dell'anchor e h_i è il numero di hop necessari per passare dal nodo N all'anchor A_i. Fatto questo il nodo N incrementa h_i e inoltra il beacon a tutti i nodi presenti nel suo range di trasmissione.

- **Stima distanze dagli anchor nodes, anche quelli esterni al transmission range**.
  Si calcolare il numero di hop e si stima la distanza fisica "legata a un singolo hop" dividendo la somma delle distanza fisiche con la somma delle distanze logiche.
    - Calcoliamo la *average 1-hop distance* con la seguente formula matematica

$$c_i = \frac{\sum_j \left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right)}{\sum_j h_j}$$

      Chiamiamo c_i *correction factor*! i è l'anchor che calcola la correzione, j identifica gli anchors che sono noti all'anchor i. Il valore viene calcolato da un anchor node quando riceve un beacon con la distanza da altri anchor node!
    - Calcolato il *correction factor* l'anchor node lo diffonde nella rete. Un nodo che riceve il valore lo inoltra ai suoi vicini e smette di inoltrare successive correzioni.

- **Stima della posizione del nodo.**
  Ogni nodo utilizza il *correction factor* ricevuto per stimare la distanza: moltiplica la *1-hop distance* per il numero di hop necessari per raggiungere gli anchors. Fatto questo e stimata la distanza di almeno tre nodi è possibile stimare la posizione del nodo utilizzando la trilaterazione!



- **Conclusioni**.
  L'algoritmo permette di risolvere il problema segnalato all'inizio e calcolare distanze di anchor nodes superiori al transmission range del target node. Questo non significa che l'algoritmo è esente da drawbacks:
    - il numero di anchor nodes incide sulla qualità della stima;
    - overhead nella comunicazione, gli anchor nodes propagano un numero significativo di messaggi;
    - assunzione che le distanze fisiche tra nodi distanti 1-hop siano uniformi in tutta la rete (*isotropic vs anisotropic networks*, nell'ultima tipologia di rete la distanza fisica cambia in base alla direzione intrapresa).

4 USING HUMANS AS SENSORS (SOCIAL NETWORKS AS SENSOR NETWORKS)

4.1 Introduzione

Il punto di vista nuovo che si vuole introdurre con questo corso è quello dell'uso degli esseri umani come sensori, attraverso i social network!

A large group of individuals having mobile devices capable of sensing and computing collectively share data and extract information to measure, map, analyze, estimate or infer any process of common interest.

- Si parla di mobile crowdsensing (dove crowd sta per folla): un numero ingente di persone che fornisce informazioni. Gli utenti pubblicano informazioni utilizzando i loro dispositivi mobili: localizzazione e aggiornamenti in tempo reale.
- Questi dati possono essere usati per educare modelli di machine learning, in generale per sostenere la collettività.
- Due tipologie di mobile crowdsensing:
  - Partecipatorio, l'utente decide volontariamente di fornire informazioni
  - Opportunistico, l'utente non è coinvolto nel processo di raccolta dati e molto spesso non è neanche consapevole.
- I social network sono difficili da classificare: formalmente sono partecipatori in quanto l'utente si iscrive volontariamente alla piattaforma e pubblica post volontariamente; nei fatti sono opportunistico perché non siamo consapevoli di come i nostri dati verranno utilizzati.

Un esempio vantaggioso di come questi dati possono essere utilizzati? Pensiamo ad eventi emergenziali (disastri naturali in primis): Fukushima nel 2011, Emilia-Romagna nel 2012… I social media possono essere utilizzati per capire dove sono concentrati maggiormente i danni e conseguentemente prendere decisioni su come intervenire e gestire l'emergenza.

Il professore è autore di uno dei primi paper sull'argomento

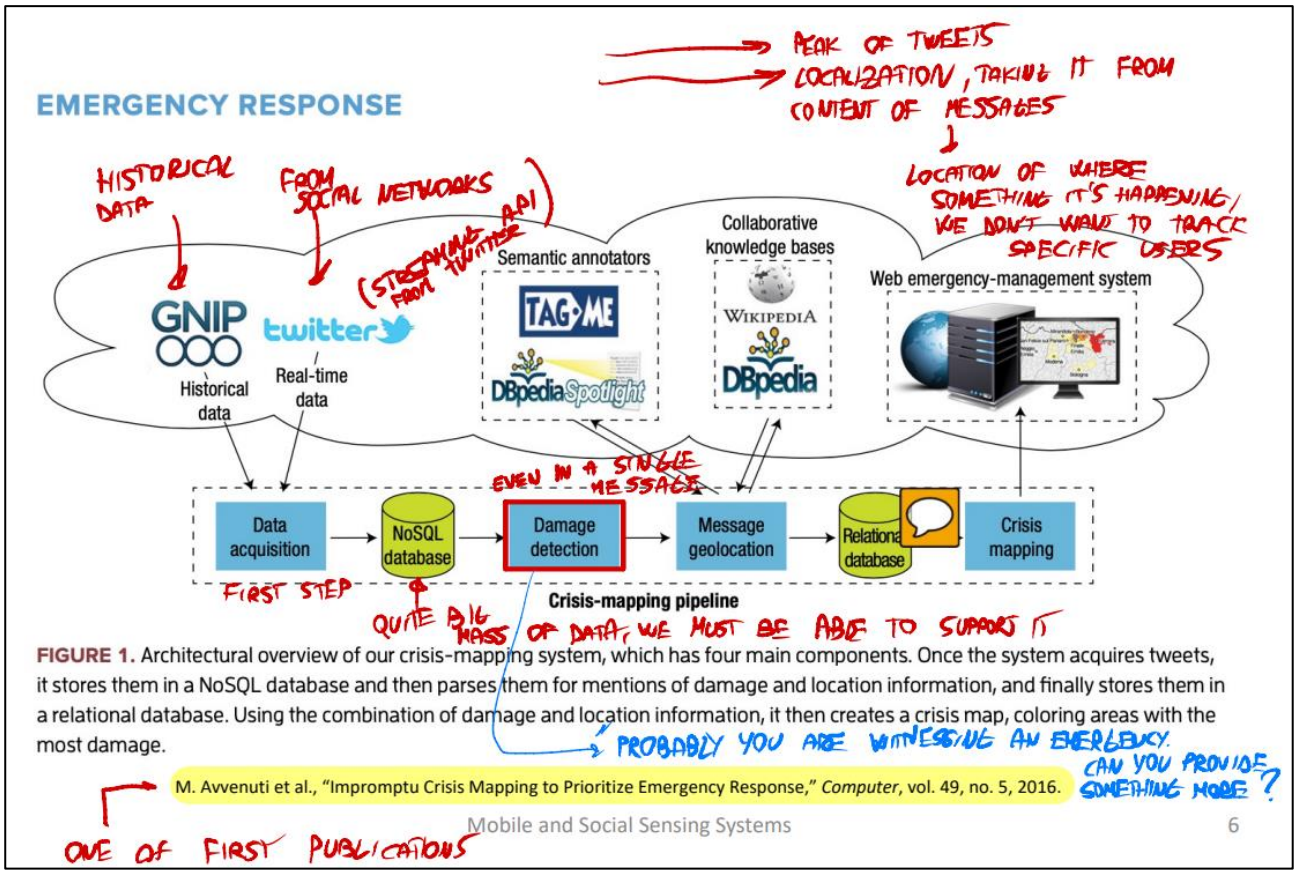


FIGURE 1. Architectural overview of our crisis-mapping system, which has four main components. Once the system acquires tweets, it stores them in a NoSQL database and then parses them for mentions of damage and location information, and finally stores them in a relational database. Using the combination of damage and location information, it then creates a crisis map, coloring areas with the most damage.

PROBABLY YOU ARE WITNESSING AN EMERGENCY. CAN YOU PROVIDE SOMETHING MORE?

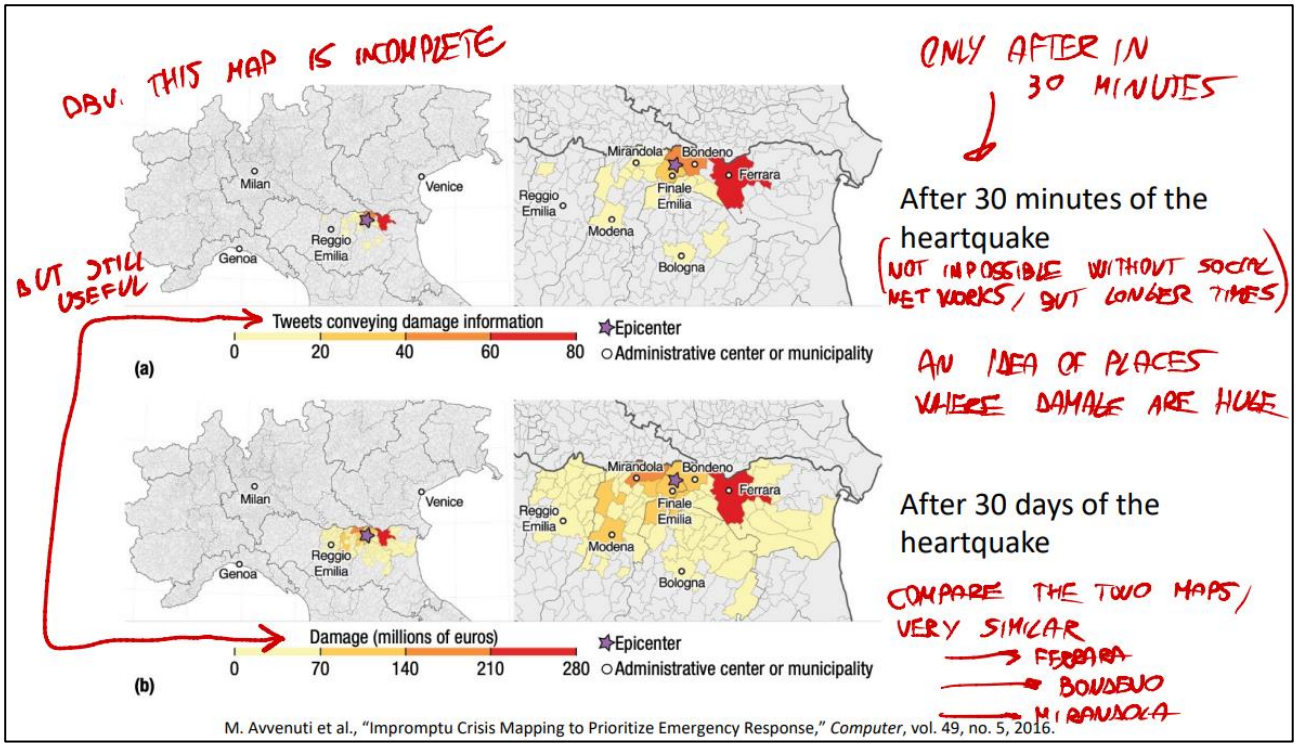
M. Avvenuti et al., "Impromptu Crisis Mapping to Prioritize Emergency Response," *Computer*, vol. 49, no. 5, 2016.

ONE OF FIRST PUBLICATIONS

Mobile and Social Sensing Systems 6

- Acquisizione dei dati storici per mezzo di dump e di dati recenti dai social network. Si fa ricorso, in particolare, alle API di Twitter.
- Le Informazioni individuate, ingenti ed eterogenee, sono memorizzate all'interno di un database non relazionale.
- Le informazioni sono passate al setaccio per individuare danni (damage detection).
- Se si individua una situazione di emergenza si cerca di ottenere maggiori informazioni, in primis la posizione dell'evento. Si fa affidamento anche a siti come Wikipedia, i cui collaboratori aggiornano i contenuti in tempo reale.
- I risultati di tutto questo vengono memorizzati in un database relazionale e possono essere utilizzati dall'unità di crisi per assumere decisioni.

Possiamo osservare come vi sia una relazione tra la quantità di Tweet pubblicati e le zone con maggiori danni (danni quantificati in termini di milioni di euro).



M. Avvenuti et al., "Impromptu Crisis Mapping to Prioritize Emergency Response," *Computer*, vol. 49, no. 5, 2016.

Quanto fatto è possibile anche in vie "classiche", tuttavia l'analisi dai social network permette una riduzione drastica dei tempi: dopo soli 30 minuti dall'evento è stato possibile avere un'idea delle zone maggiormente coinvolte!

4.2 Paradigma "Humans as Sensors" (HaS) e problematiche da gestire

La grande disponibilità di contenuti sui social network ci suggerisce che la più grande rete di sensori è costituita da esseri umani!

- Pensiamo alla Wireless Sensor Network vista precedentemente, ci sono delle assonanze con quanto stiamo dicendo ora: utilizziamo i social networks come reti di sensori.
- La presenza di sensori permette di analizzare cosa sta accadendo nel mondo reale e, eventualmente, fare anche valutazioni su cosa accadrà nel futuro.
- **N.B**. Analisi del mondo reale attraverso mere osservazioni degli utenti: *claims*! Da Wikipedia: *state or assert that something is the case, typically without providing evidence or proof*.

Emerge il *reliable sensing problem*!

- Si adotta un modello binario: escluendo contenuti di carattere soggettivo ("Oggi è una bella giornata") ed emozioni personali ("Sono depresso") possiamo dire che l'ambiente è caratterizzato da uno stato univoco. **Segue un'unica base di verità**: le osservazioni rispetto all'ambiente sono caratterizzate come vere o false.

We model humans as sources of unknown reliability, generating binary observations of uncertain provenance.

- Dobbiamo classificare i claims come veri o falsi. Ciò non è facile perché il tutto dipende da una molteplicità di aspetti. Sottolineiamo che:
    - ***Unknown reliability***.
      La reliability dei sensori umani è sconosciuta a priori (non sappiamo nulla su questi sensori umani, non possiamo dare per scontato che un soggetto sia affidabile e un altro no)
    - ***Uncertain provenance***.
      La provenienza delle osservazioni è incerta: non possiamo dare per scontato che un'osservazione dell'utente su un evento derivi dall'aver partecipato a tale evento! L'utente potrebbe aver fatto sua l'osservazione di un altro utente.

      $$\text{Reporting} \neq \text{Witnessing}$$
- Tenendo conto di quanto detto (*model human participants as sources of unknown reliability generating binary measurements of uncertain provenance*) dobbiamo utilizzare algoritmi e teoria probabilistica per individuare e rimuovere cattive osservazioni sul mondo reale.

Approfondiamo gli aspetti evidenziati ("grassettate" le cose che di solito non dico nei discorsi precedenti, quando ripeto):

- ***Unknown reliability***.
    - I sensori fisici tradizionali sono noti in tutte le loro caratteristiche, inclusa la reliability. Conosciamo il loro comportamento, che rimarrà sempre quello salvo malfunzionamenti.
    - Non possiamo dire lo stesso sugli esseri umani: non conosciamo la reliability, non mantengono lo stesso comportamento e non possiamo agire su di essi affinchè si comportino come da noi desiderato.
    - Tutto molto più difficile dei classici sensori: minore accuratezza, ma siamo interessati in quanto *humans are much broader in what they can observe*.
    - **Definiamo la *reliability* come la probabilità che un'utente esprima *true claims*.**
    - **Un *true claim* è un'osservazione che contribuisce a una corretta rappresentazione dello stato del mondo reale.**
- ***Binary observation***.
    - Il mondo reale può essere immaginato come una collezione di fatti che le persone desiderano segnalare.
    - I sensori umani segnalano alcuni dei fatti che sono in grado di osservare.
    - Associamo a ciascun fatto una variabile binaria: true o false. Misuriamo una molteplicità di variabili binarie, e se non siamo sicuri della reliability della fonte ricorriamo alla teoria della probabilità!
    - ***Reliable sensing* significa individuare tra le osservazioni umane quelle che rappresentano adeguatamente il mondo reale**.
- ***Uncertain provenance***.
    - I sensori fisici sono caratterizzati da rumore, stessa cosa quelli umani.
    - Un utente può pubblicare un'osservazione, ma questa osservazione non è dovuta a un suo diretto coinvolgimento in un evento: potrebbe aver fatto sua l'osservazione di un altro utente.
    - Il problema dietro questa cosa è che l'utente potrebbe far sue osservazioni non veritiere e contribuire a diffondere una rappresentazione falsa del mondo reale (condivide un contenuto senza verificarne la veridicità). Tale "diffusione di rumore" è tipica degli esseri umani e non ha similarità con i sensori fisici tradizionali.
    - Detta in altre parole: non sappiamo se una fonte è indipendente da altre!

"Let model human participants as sources of unknown reliability generating binary measurements of uncertain provenance"

4.3 Architettura “Human as Sensors” per gestire gli aspetti precedenti

4.3.1 Introduzione

Per ricostruire informazioni veritiere a partire dai dati riportati dagli utenti sono necessari i seguenti passaggi:

- Data collection from sensor networks
- Data structuring to enable analysis
- Understanding source relationship
- Estimating probability of correctness of individual observation

4.3.2 Data collection

Dobbiamo raccogliere i dati dalla rete di sensori, quindi dai social networks.

- Supponiamo di fare questo su Twitter.
- Le API permettono di estrarre i dati desiderati ponendo alcuni filtri: keyword particolari, regione geografica, periodo temporale….

4.3.3 Data structuring

I dati raccolti nella fase precedente devono essere organizzati in modo tale da poter essere analizzabili.

- **Somiglianza tra osservazioni**.
  Utilizziamo una funzione di tokenizzazione su ogni tweet: questo significa ottenere da ogni tweet un set di parole chiave che lo caratterizzano.
  - Ogni tweet rappresenta un’osservazione individuale.
  - Lo spazio che stiamo analizzando è definito da un set di parole chiave: i token!
  - Associamo ad ogni tweet un vettore i cui elementi restituiscono il numero di occorrenze dei token all’interno del messaggio.
  - Utilizziamo una funzione `distance(t1, t2)` che date due osservazioni `t1` e `t2` restituisce un valore che rappresenta la somiglianza tra le due osservazioni. Maggiore è il valore restituito, maggiore è la differenza tra le due osservazioni.
  - Adottiamo in questo contesto una cosine similarity function[1]

$$\text{similarity} = cos\,\theta = \frac{A \cdot B}{||A||\,||B||}$$

  Il valore restituito è compreso tra -1 e $+1$: maggiore è la differenza maggiore è θ.

```
doc_1 = "Data is the oil of the digital economy"
doc_2 = "Data is a new oil"

# Vector representation of the document
doc_1_vector = [1, 1, 1, 1, 0, 1, 1, 2]
doc_2_vector = [1, 0, 0, 1, 1, 0, 1, 0]
```

	data	digital	economy	is	new	of	oil	the
doc_1	1	1	1	1	0	1	1	2
doc_2	1	0	0	1	1	0	1	0

[1] Da ScienceDirect: *Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors* ***are pointing in roughly the same direction****. It is often used to measure document similarity in text analysis*.

$$A \cdot B = \sum_{i=1}^{n} A_i B_i$$
$$= (1*1) + (1*0) + (1*0) + (1*1) + (0*1) + (1*0) + (1*1) + (2*0)$$
$$= 3$$

$$\sqrt{\sum_{i=1}^{n} A_i^2} = \sqrt{1+1+1+1+0+1+1+4} = \sqrt{10}$$

$$\sqrt{\sum_{i=1}^{n} B_i^2} = \sqrt{1+0+0+1+1+0+1+0} = \sqrt{4}$$

$$cosine\ similarity = cos\theta = \frac{A \cdot B}{|A||B|} = \frac{3}{\sqrt{10}*\sqrt{4}} = 0.4743$$

- **Individuazione dei claims da considerare**.
  Dobbiamo individuare quali sono i claims da analizzare. Costruiamo un grafo dove:
  - i nodi sono le osservazioni individuali (i tweet);
  - gli archi sono pesati e rappresentano le somiglianze tra nodi.

  A questo punto facciamo clustering: ciascun cluster ottenuto rappresenta un *claim,* cioè un pezzo di informazione riportato da una molteplicità di fonti.

- ***Source-Claim graph*** **(SC)**.
  Passaggio successivo è la costruzione di un *Source-Claim graph* caratterizzato da due tipologie di nodi:
  - Fonti S_i
  - Claim C_l

  Si pone un arco $S_i C_l = 1$ se la fonte S_i ha sostenuto il claim C_l. Il passaggio finale dell'architettura sarà comprendere se il claim è *true* o *false*.

4.3.4 Sources relationship

Il *Source-Claim graph* permette di avere una visione quantitativa del supporto al *claim*, ma non ci segnala il rapporto tra le fonti che hanno sostenuto tale *claim*, cosa necessaria per poter rispondere all'uncertain provenance. Serve a tal proposito un information ***dissemination graph*** (SD) che permette di stimare come le informazioni potrebbero propagarsi da un utente a un altro.

- **Epidemic Cascade network**
  Ogni osservazione è modellata come una cascata. Il "tempo di contagio" permette di analizzare: chi è il primo ad aver affermato un certo "claim" e cv hi lo ha affermato successivamente; la velocità con cui il claim si è diffuso sul social network; dove è stato affermato il *claim* all'interno del social network (gruppi/pagine particolari?).
- **Grafici specifici per Twitter.**
  - **Follower-Followee network** (FF)
    Ogni nodo rappresenta una fonte. Un arco $S_i S_k$ esiste all'interno del grafo se la fonte S_k è un follower dalla fonte S_i
  - **Re-Tweeting network** (RT)
    Ogni nodo rappresenta una fonte. Un arco $S_i S_k$ esiste all'interno del grafo se la fonte S_k ha retweettato messaggi dalla fonte S_i
  - **Combined network** (RT+FF)
    Ogni nodo rappresenta una fonte. Un arco $S_i S_k$ esiste all'interno del grafo se la fonte S_k è un follower dalla fonte S_i o la fonte S_k è un follower dalla fonte S_i (combinazione dei criteri delle due reti precedenti).

4.3.5 The Estimation problem

Il passaggio conclusivo è determinare la veridicità dei claim.

- **Una prima idea (sbagliata): voting algorithm**.
  Sia N il numero di claim considerati. Per ogni claim $C_j, 1 \leq j \leq N$ contiamo tutte le fonti S_i tali che $S_i C_j = 1$. Quello che facciamo è affermare che maggiore è il numero di fonti che sostengono un claim, maggiore è la veridicità del claim. Approccio problematico:
  - le fonti non hanno tutte lo stesso livello di affidabilità, il metodo appena detto considera tutte le fonti affidabili allo stesso modo;
  - le fonti non è detto siano indipendenti, se una fonte ripete quello che sente da altri non si aggiunge credibilità al claim.

  Due versioni:
  - ***Voting***, considera sia retweet che tweet originali come voti
  - ***Voting-NoRT***, considera solo i tweet originali come voti (miglioramento, considerare retweet significa considerare voti non indipendenti)

- **Cosa abbiamo per poter determinare la veridicità del claim?**
  Due grafi: il *Source-Claim Graph* (SC) e il *Social Dissemination Graph* (SD), che sono cose oggettive e veritiere.

- **Primo metodo: *likelihood estimation*.**
  Calcoliamo la probabilità della correttezza di un claim, visti SC e SD (cioè tenendo in considerazione unknown source reliability e uncertain provenance).
  $$\forall j, 1 \leq j \leq N: p\big(C_j = 1 \big| SC, SD\big)$$
  Calcolati questi valori il sistema inoltra all'utente solo i tweet che soddisfano una minima probabilità di correttezza (soglia stabilità dall'utente).

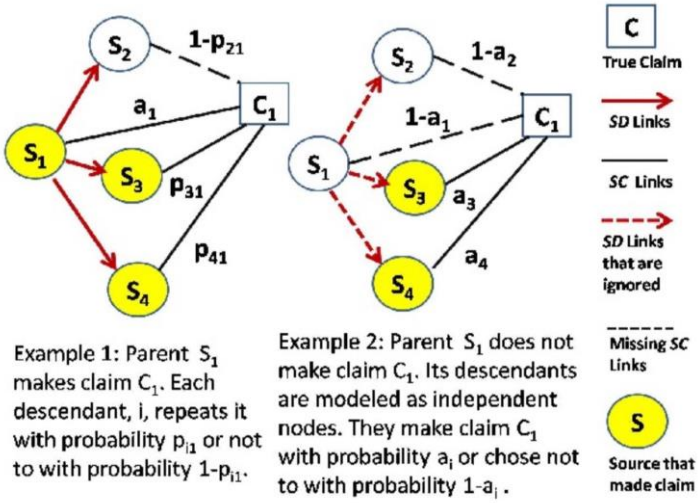
- **Secondo metodo: Maximum Likelihood Estimation (MLE)**.
  Come prima dobbiamo tenere in considerazione *unknown reliability* e *uncertain provenance*.
  - Sia m il numero totale di fonti da cui abbiamo dati.
    - Consideriamo le fonti $S_i, 1 \leq i \leq m$
    - Per ogni fonte i-esima ci interessano due valori, non conosciuti a priori:
      - La probabilità di avere un vero positivo: $a_i = p\big(S_i C_j = 1 \big| C_j = 1\big)$, questi valori ci interessano di più perché sono quelli che esprimo la reliability delle fonti.
      - La probabilità di avere un falso positivo: $b_i = p\big(S_i C_j = 1 \big| C_j = 0\big)$
    - Consideriamo anche $d = p\big(C_j = 1\big)$, anch'essa non nota
    - Costruiamo con i valori detti il vettore $\theta = (a_1 \dots a_m \, b_1 \dots b_m \, d)$
  - Vogliamo trovare un vettore θ tale per cui la probabilità di osservazione del SC graph, dato il social network SD, risulta massimizzata.
    $$p(SC|SD, \theta)$$
    Supponendo che i claim siano l'uno indipendente dall'altro applichiamo il teorema della probabilità totale per scrivere
    $$p(SC|SD, \theta) = \sum_z p(SC, z|SD, \theta)$$
    Dove z è un vettore il cui elementi z_j è uguale a 1 se $C_j = \text{true}$, 0 altrimenti. Se individuiamo z e θ abbiamo finito: conosciamo l'affidabilità delle fonti (vettore θ) e sei claim sono veritieri (vettore z).
  - Nei nostri ragionamenti mancano ancora considerazioni sul dissemination graph. Cosa possiamo fare rispetto all'uncertain provenance e quindi alla relazione tra fonti?
    - Il SD è un grafo orientato, si stabiliscono relazioni dove definiamo alcuni nodi padri e altri figli. Ogni nodo rappresenta una fonte, nel caso di Twitter un utente.

- Sia nodi padri che figli sostengono claims.
- Se il nodo padre k sostiene un claim e anche il figlio i lo sostiene allora il figlio lo ha ripetuto con una probabilità p_{ik}

$$p_{ik} = \frac{\text{Numero di volte che i nodi i e k hanno sostenuto lo stesso claim}}{\text{Numero di claim sostenuti dal nodo k}}$$

- Se il nodo padre sostiene un claim e il figlio no allora si suppone che il figlio si sia mosso in maniera indipendente. In tal caso la probabilità è pari a $1 - p_{ik}$

Rendiamo meglio l'idea con la seguente figura, dove andiamo a mettere insieme *SC graph e SD graph.*



- Si parla di ***Expectation Maximization*** (EM) se si intende l'algoritmo precedente tranne la parte finale sul Social Dissemination graph: questo significa non poter distinguere *reliable sources* da *independent sources*. Due tipologie:
  - ***Regular-EM***, tutte le fonti sono considerate osservatori indipendenti
  - ***Regular-EM-AD***, le fonti dipendenti da altre fonti sono rimosse sfruttando approcci euristici.

Riprendendo l'esempio di paper con cui si è affrontata la gestione delle emergenze. Ci si muove con la seguente metodologia:

- Si sceglie la tipologia di evento di interesse;
- Si individua tutto ciò che permette di identificare i tweet come relativi alla tipologia di evento scelta (*keywords*, posizione geografica, etc…)
- Si estraggono i messaggi utilizzando le API
- Si memorizzano i tweet raccolti in database.
- Si applicano tecniche di clustering per trovare i *claims* da analizzare.
- Si costruiscono i grafi SC ed SD.
- Si applica il *Maximum Likelihood Estimation algorithm*.

4.4 Validation methods

La validazione di un metodo è un processo manuale: vogliamo verificare se la base di verità restituita dall'algoritmo coincide con la rappresentazione della realtà.

- Essendo un processo manuale è difficile considerare un numero elevato di claims: ci limitiamo per comodità a 150 claims.
- Per ciascun claim verifichiamo se quanto ottenuto col metodo adottato è corretto. Facciamo questo prendendo a riferimento una molteplicità di fonti (ad esempio i mass media).
  - True claims: *"claims that describe a physical or social event that is generally observable by multiple indepedent observers and corroborated by sources external to the experiment"*

    - Unconfirmed claims: *"claims that do not meet the definition of true claims. May include true claims that cannot be verified"*.
- Rispetto agli ultimi, che includono anche i true claims non verificabili, dobbiamo decidere se assumerci la responsabilità di dire se il label attribuito al claim è veritiero oppure adottare un approccio pessimistico rimuovendo tutti gli unconfirmed claims.
- I metodi che abbiamo visto tendono a ignorare informazioni di carattere introspettivo (emozioni ed opinioni soggettive): si ipotizza che questa cosa sia legata al fatto che emozioni e slogan tendono ad essere retweettati e quindi esclusi dallo schema. Fatti esterni sono invece monitorati da più fonti in maniera indipendente.

5 DISTRIBUTED HASH TABLE

5.1 Publish-Subscribe pattern

5.1.1 Ruoli

Il pattern è caratterizzato dalla presenza di due figure:

- ***Publishers***
  I publishers sono coloro che inviano messaggi nella rete. Questi messaggi saranno ricevuti da subscriber, ma a differenza dei pattern tradizionali il sender non specifica l'indirizzo del destinatario: si limita a classificare i messaggi in classi senza conoscere i subscriber.
  - Come possiamo gestire la consegna dei messaggi ai subscriber in maniera efficiente?
- ***Subscribers***
  I subscribers sono coloro che ricevono i messaggi. Segnalano interesse in particolari classi di messaggi e ricevono solo questi senza essere a conoscenza del publisher.
  - Quali publisher esistono? Quali topic esistono?

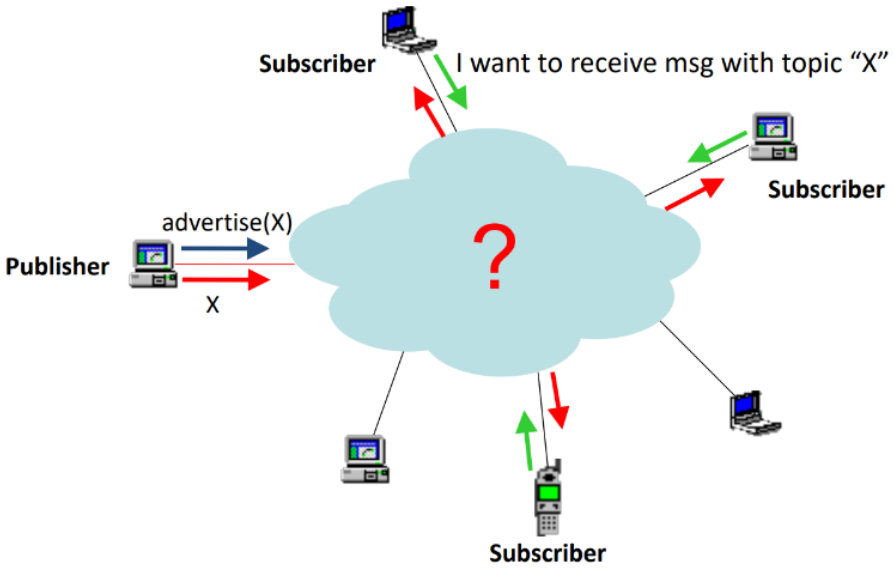
5.1.2 Filtering

Decidere quali messaggi verranno recapitati a un certo nodo è determinato da un'operazione di filtering, che può essere di due tipologie:

- ***topic-based system***
  I messaggi sono associati a dei topic (*logical channels*), identificati da una chiave. I subscriber riceveranno tutti i messaggi pubblicati sui topic che hanno sottoscritto.
  - La complessità del problema è scaricata sui publisher, che classificano i messaggi!
- ***content-based system***
  I messaggi non sono associati a topic, ma sono classificabili sulla base di keywords presenti al loro interno. Il subscriber determina se il messaggio è di suo interesse analizzando le keyword.

5.2 Publish-Subscribe problem

Consideriamo un contesto caratterizzato da un certo numero di *publishers* e un certo numero di *subscribers*. Adottare il Publish-Subscribe pattern richiede di risolvere alcuni problemi.



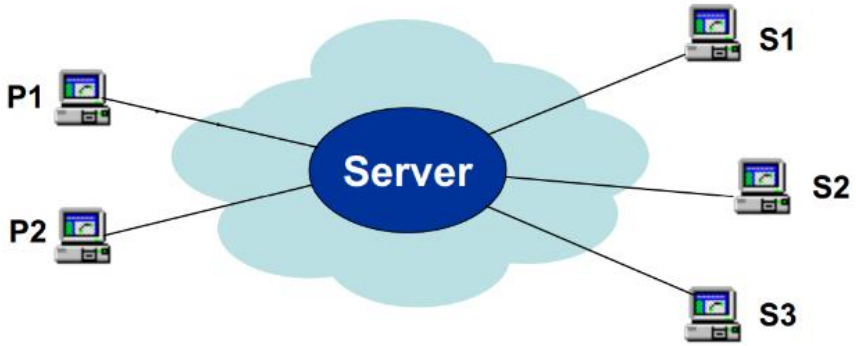
- Il sistema deve permette ai publisher di fare *advertise*, cioè informare il sistema che seguirà uno stream di messaggi relativo a un topic X.
- Dove vengono memorizzate le informazioni relative ai messaggi, considerato che non indichiamo l'indirizzo di uno o più nodi? Dobbiamo implementare un efficiente *message delivering*, le informazioni devono essere memorizzate da qualche parte fino al termine dell'operazione.

- Il subscriber, in maniera asincrona rispetto al publisher, segnalerà che vuole ricevere messaggi relativi a un certo topic. Come fa il subscriber a conoscere quali sono i topic che può sottoscrivere? Come trova il nodo del sistema dove sono memorizzati i messaggi?

Come possiamo ottenere una trasmissione efficiente dei messaggi? Facciamo delle proposte!

5.3 Prima proposta: soluzione client-server

La prima proposta è un'architettura client-server:



- I nodi dialogano sempre col server.
- Il server riceve gli advertise dai publisher, memorizza tutte le informazioni che necessitano essere memorizzate e trasmette i messaggi ai subscriber.

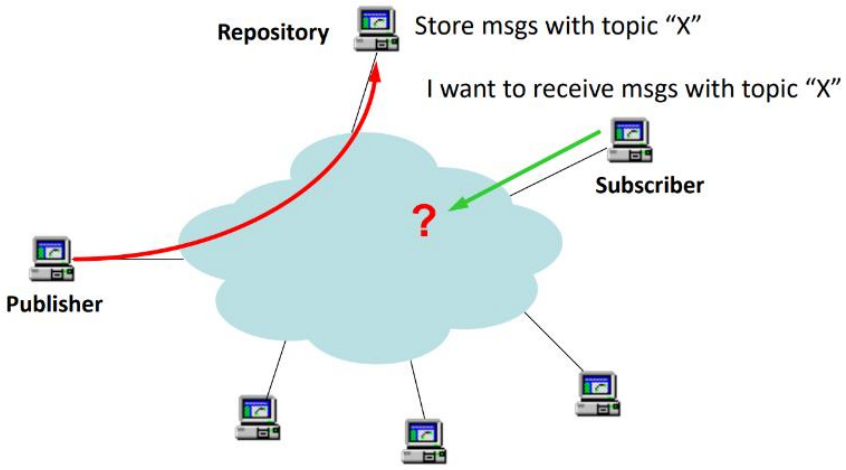
La complessità viene scaricata interamente sul server, tuttavia emergono problemi significativi: la soluzione non è scalabile (immaginiamoci un contesto con migliaia di nodi publisher e nodi server) e il server costituisce un single point of failure (se non funziona il sistema non funziona).

5.4 Seconda proposta: soluzione peer-to-peer e Lookup problem

La seconda proposta è un'architettura peer-to-peer

- Tutti i nodi sono in grado di svolgere tutte le funzioni utili per il sistema, senza un'assegnazione di ruoli a priori (asymmetric in function). Non si ha un controllo centralizzato.
- Numero elevato di nodi: questi sono eterogenei e *unreliable*.

In questa soluzione il "*Publish-Subscribe problem*" assume un altro nome: "Lookup problem"!



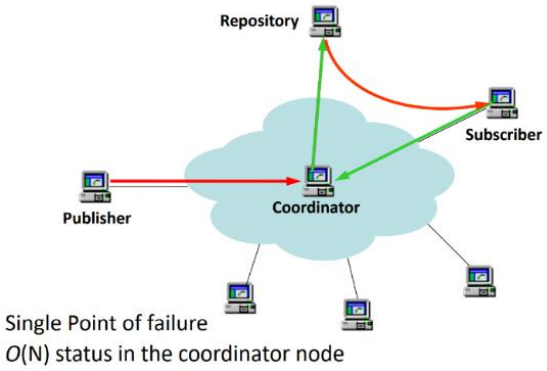
- **Dove sono memorizzate le informazioni?** Questa è la domanda principe in un sistema peer-to-peer
- Problema simmetrico:
  - Abbiamo peer che si comportano da *Publisher* e che vogliono fare advertise rispetto a un particolare topic X. Il topic X è identificato da una chiave (ad esempio un valore numerico) che identifica un particolare topic: dove memorizziamo la chiave associata al topic?
  - Abbiamo peer che si comportano da *Subscriber* che desiderano trovare il nodo a cui rivolgersi per sottoscrivere un topic X, e sapere dove i messaggi relativi al topic X sono memorizzati.

- Alcuni dei nodi partecipanti, in certe proposte, si comportano come *repository nodes*, cioè nodi che memorizza le chiavi associate ai topic! Domanda: dove trovo il repository node?
    - **Lato *publisher***
      Il publisher deve poter fare *advertise* sul topic X, associare una chiave a quel topic e memorizzare la chiave generata su uno dei repository node (ovviamente non ce ne sarà uno solo di questi nodi)!
    - **Lato *subscriber***
      Situazione ancora più complicata: grande numero di *repository nodes*. Vogliamo trovare dove la chiave è memorizzata.

  La complessità è scaricata sul *subscriber*, non è una cattiva idea. Capiremo meglio più avanti, consideriamo che dobbiamo ancora pensare a come implementare la consegna dei messaggi.

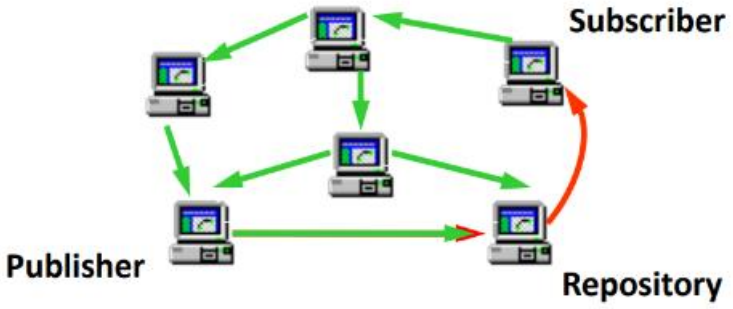
Due approcci possibili

- **Approccio centralizzato**.
  Si introducono dei coordinator nodes a cui si rivolgono sia i publishers che i subscribers.
    - Il publisher invia messaggi a un particolare coordinatore, quando questi li riceve assegna una chiave al topic X e decide quale repository memorizzerà la chiave. L'identità della repository è nota solo al coordinator, non al subscriber.
    - Il subscriber invia messaggi a un particolare coordinatore per richiedere l'accesso ai messaggi. Il coordinatore, che conosce la repository, la avverte: fatto ciò repository e subscriber dialogheranno direttamente senza il coinvolgimento del coordinatore.



  Il coordinatore non è paragonabile a un server in quanto il primo, a differenza del secondo, non memorizza informazioni. Permangono i problemi tipici dell'approccio centralizzato.

- ***Flooding approach***.
  Il publisher trasmette l'advertise a tutti i peer presenti nel suo raggio di trasmissione. I peer inoltreranno a loro volta il messaggio ad altri peer presenti nella rete: dopo un po' di tempo tutta la rete sarà raggiunta dall'*advertise message* e quindi tutti i nodi nel sistema sapranno che c'è un nuovo topic e che verranno pubblicati messaggi a riguardo.

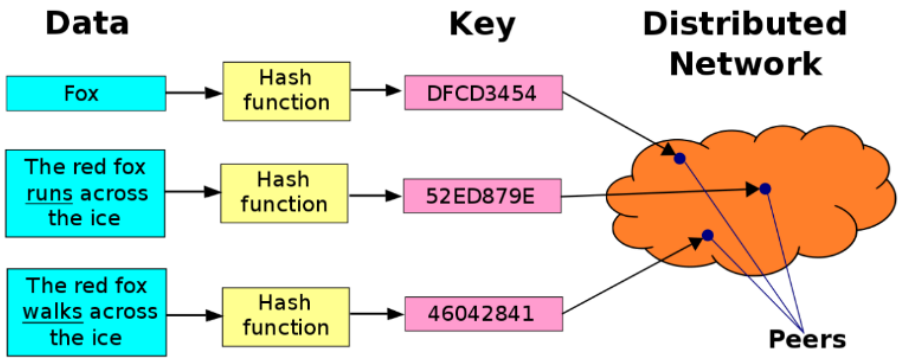


  I *repository nodes* sono sempre presenti, che possono essere usati per la consegna dei messaggi. Non abbiamo i problemi tipici dell'approccio centralizzato, ma il numero di messaggi che il subscriber deve inviare per individuare il repository node è veramente elevato!

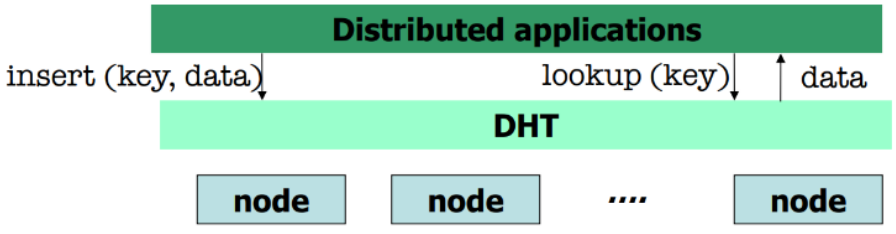
5.5 Terza proposta (la migliore): Distributed Hash Table (DHT)!

5.5.1 Introduzione

DHT è acronimo di Distributed Hash Table.



- Sono memorizzate coppie $\langle \text{key}, value \rangle$ che l'utente può recuperare (lookup) in maniera efficiente.
- Approccio distribuito: la tabella è distribuita su più nodi!
- La chiave è un identificatore univoco generato a partire da una funzione hash. Value può essere di tutto: indirizzi, documenti, qualunque genere di dato…
- La responsabilità sul mantenimento delle corrispondenze chiave-valori è distribuita tra i suoi nodi, in modo tale da minimizzare problemi nel caso in cui cambino i partecipanti della rete: le chiavi possono essere redistribuite senza problema tra gli altri nodi della rete nel caso in cui un nodo smetta di funzionare (per qualunque ragione).
- Presente un ***abstract keyspace*** che ospiterà le chiavi: dato che una singola chiave può essere una stringa di 160bit è necessario che questo spazio sia di dimensioni significative. Questo spazio viene ripartito tra i nodi che partecipano alla rete.
- Si definisce un ***overlay network***, livello logico superiore a quello fisico, che permetta di connettere i nodi e permettere a questi di individuare il proprietario di una particolare chiave presente nel keyspace.
- Una DHT è caratterizzata dalle seguenti proprietà:
  - ***Autonomy and Decentralization***
    I nodi lavorano collegialmente e non esiste alcuna autorità centrale.
  - ***Scalability***
    Il sistema è scalabile: funziona anche con un numero estremamente elevato di nodi.
  - ***Fault tolerance***
    Il sistema è in grado di gestire gli errori, in particolare è in grado di gestire l'abbandono della rete da parte dei nodi (che ricordiamo essere di base *unreliable*)
  - ***Load balancing***
    Le chiavi sono distribuite in maniera bilanciata tra i nodi che costituiscono la rete.
  - ***Anonimity***
    Si garantisce l'anonimato dei nodi, nel senso che la loro posizione fisica non è nota. Ricordiamo che nel Publish-Subscribe model il Publisher non indica indirizzi di rete.
- **Funzioni garantite dal DHT**.
  Una generica implementazione del DHT deve fornire almeno due funzioni: insert per aggiungere all'interno della rete una coppia $\langle \text{key}, value \rangle$, lookup per individuare il valore associato a una particolare chiave.



Attenzione alle proprietà della funzione hash:

- La chiave deve identificare i dati in maniera univoca, dato che è il valore utilizzato dalla funzione lookup per recuperare dati precedentemente memorizzati nella DHT;
- La DHT è pensata in maneira tale da distribuire le chiavi tra i nodi in maniera bilanciata, questo per evitare che particolari nodi presenti all'interno della rete siano soggetti a un carico maggiore.

L'applicazione che esegue le funzioni insert e lookup ha l'impressione di lavorare su un abstract keyspace, in realtà l'overlay network attua quanto già detto precedentemente:

- Rispetto alla funzione insert il messaggio viene inoltrato da nodo a nodo fino a quando non viene individuato un nodo disponibile a memorizzare la coppia $\langle \mathrm{key}, value \rangle$. Si memorizza la coppia su un nodo, ma questo nodo non è indicato a priori dall'utente.
- Rispetto alla funzione lookup si svolge una ricerca all'interno dei nodi e si restituisce i dati richiesti se individuati, il tutto con un meccanismo che garantisce il non invio di pacchetti a tutti i nodi presenti nella rete (niente flooding, ribadiamo).
  - Complessità della lookup con approccio di tipo flooding: $\mathcal{O}(n)$
  - Complessità della lookup con approccio DHT: $\mathcal{O}(log\, n)$

5.5.2 Proposta di overlay network: Pastry

5.5.2.1 Introduzione

In Pastry la Distributed Hash Table risulta implementata con paradigma peer-to-peer e rete con ridondanze.

- Non si ha flooding.
- Ogni nodo è caratterizzato da una tabella di routing, che è costruita e riparata in maniera dinamica.
- Il primo passaggio avviene contattando un nodo di cui conosciamo l'indirizzo IP e che sappiamo essere presente all'interno della rete.
- Non ci sono single point of failure, ogni singolo nodo può abbandonare la rete senza provocare perdita di dati.

5.5.2.2 Pastry Design

La tabella è implementata attraverso un key-space circolare.

- I nodi sono identificati da ***NodeIDs***, unsigned integer di 128 bit.
- Il NodeID di un particolare nodo è generato quando questo entra a far parte della rete: si applica una funzione hash ponendo in ingresso il suo indirizzo IP.
- Anche le chiavi dei dati (***key***) memorizzati sono unsigned integer di 128 bit.
- NodeID e key sono sequenze di cifre in base 2^b (tipicamente $b = 4$)
- Una chiave è memorizzata in un nodo il cui nodeID è quello numericamente più vicino alla chiave, (messo a confronto con tutti i nodi).

5.5.2.3 Routing in Pastry

Con Pastry routing intendiamo l'insieme dei passaggi che permettono di stabilire il percorso che un pacchetto deve percorrere da un nodo sorgente a un nodo destinatario, nodi presenti all'interno della rete.

- Se la rete consiste di N nodi allora pastry può trasmettere il messaggio al nodo destinatario con un numero di step pari a $\lceil log_{2^b} N \rceil$
- Nel caso in cui i nodi falliscano la consegna del messaggio è comunque garantita, a condizione che non falliscano $\frac{l}{2}$ nodi con NodeIDs adiacenti, dove l è la dimensione dei *leaf sets* (tipico valore $l = 16$).

Il routing si basa sulla presenza di tre tabelle: ***routing tables***, ***leaf sets*** e ***neighborhood sets***.

- ***Routing table***.
  Ogni nodo in Pastry è caratterizzato da una tabella di routing, che presenta la seguente struttura:
  - Ha un numero di righe pari a $\lceil log_{2^b} N \rceil$ e un numero di colonne pari a 2^b. Nella tabella nella pagina seguente abbiamo supposto $b = 4$ (valore tipico)

- Ogni riga n della tabella rappresenta un prefisso condiviso con altri nodi. Per ogni elemento della riga si pone un nodeID relativo a un nodo che ha lo stesso prefisso rappresentato dalla riga, ma i successivi valori diversi ($2^b - 1$ combinazioni possibili)! Si pongono solo i nodeID dei nodi, non gli indirizzi IP!

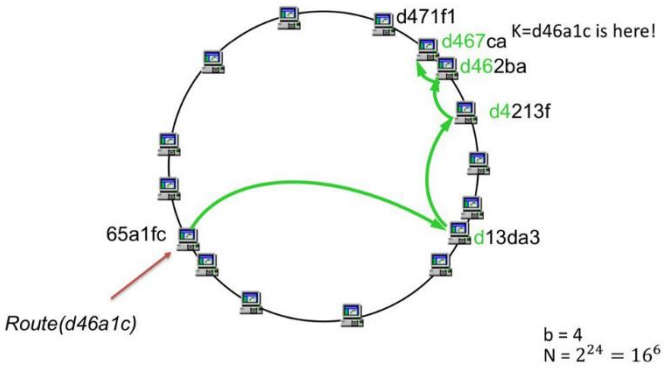
0x	1x	2x	3x	4x	5x		7x	8x	9x	ax	bx	cx	dx	ex	fx
60x	61x	62x	63x	64x		66x	67x	68x	69x	6ax	6bx	6cx	6dx	6ex	6fx
650x	651x	652x	653x	654x	655x	656x	657x	658x	659x		65bx	65cx	65dx	65ex	65fx
65x		65xa	65xa	65xa	65xa	65xa	65xa	65xa	65xa	65xa	65xa	65xa	65xa	65xa	65xa
x		x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf

- ***Leaf set***.
  Ogni nodo x mantiene indirizzi IP dei leaf sets, cioè l nodi col NodeID numericamente vicino a quello del nodo x.
  - $\frac{l}{2}$ sono quelli con il NodeID più grande e numericamente più vicino al nodeID del nodo x
  - $\frac{l}{2}$ sono quelli con il NodeID più piccolo e numericamente più vicino al nodeID del nodo x

- ***Neighborhood set***.
  Ogni nodo x mantiene gli indirizzi IP dei nodi a lui più vicini fisicamente parlando. Si utilizza quando possibile per ottimizzare e ha un ruolo fondamentale per gestire *node departure/failure*.

Introdotte le tabelle alla base del routing di Pastry possiamo definire i vari passaggi:

- Si trasmette un pacchetto costituito dalla coppia $\langle \text{msg,key} \rangle$.
- In ogni *currentNode* (uno dei nodi che costituisce il percorso svolto dal pacchetto) si visiona la tabella di Routing.
- Il pacchetto viene inoltrato al nodo il cui *nodeID* condivide con la chiave key un prefisso che è almeno una cifra più lungo del prefisso condiviso tra *key* e *nodeID* del *currentNode*.
- Se non si individuano nodi del genere nella tabella di routing si visiona il *leaf set*, inoltrando il messaggio a un nodo il cui *nodeID* è quello numericamente più vicino alla chiave rispetto al *nodeID* del *currentNode*.
- Se non esiste un nodo del genere all'interno del *leaf set* ci fermiamo, la chiave viene memorizzata nel *currentNode*!



5.5.2.4 Proximitry metric and route convergence property

Vogliamo definire una matrice di prossimità, cioè poter definire la distanza tra un qualunque coppia di nodi.

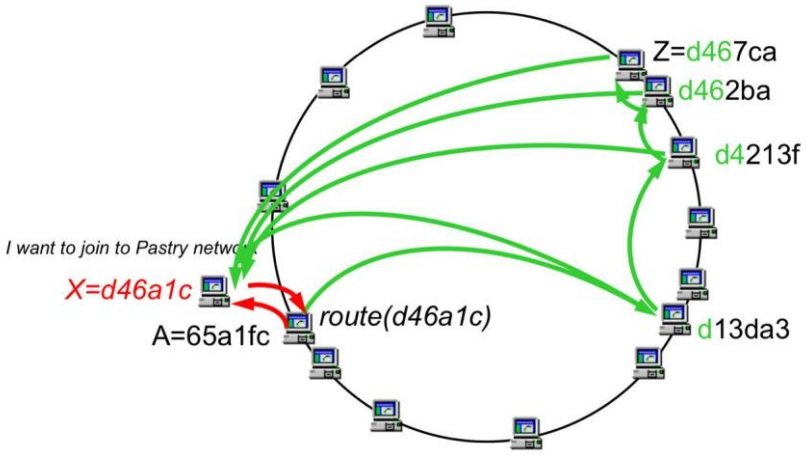
- Un esempio è l'uso del Round Trip Time.
- L'implementazione di Pastry prevede una funzione che permette a un nodo di conoscere la sua distanza rispetto a un altro nodo di cui è noto l'indirizzo IP.

Ogni entrata della tabella di routing è scelta in modo tale da fare riferimento al nodo più vicino, in accordo alla matrica di prossimità (nearest node with a longer prefix match).

Route convergenze property. Consideriamo la distanza percorsa da due messaggi aventi la stessa key. Data la topologia della rete, simulazioni dimostrano che la distanza media percorsa da ciascuno dei due messaggi prima di convergere a destinazione è approssimativamente uguale alla distanza tra i rispettivi *source node*.

5.5.2.5 Node Arrival

Cosa succede se un nodo con NodeID X arriva per la prima volta presso la rete Pastry?



- Contatta un nodo conosciuto A, appartenente alla rete Pastry, per chiedere di partecipare alla rete. Se conosce più nodi si utilizza una metrica di prossimità per scegliere il nodo più vicino (ad esempio il Round Trip Time).
- Il messaggio di richiesta viene trasmesso (per mezzo di routing) a un nodo Z che è quello col nodeID numericamente più vicino ad X.
- Z trasmette al nodo X il suo leaf set, e la i-esima riga della tabella di Routing dall'i-esimo nodo individuato nel percorso dal nodo A al nodo Z.
- A quel punto X notifica tutti i nodi coinvolti per segnalare la sua presenza.

5.5.2.6 Node Departure or Node Failure, then Node Recovery

Per gestire node departure e node failure si fa affidamento al neighborhood set introdotto precedentemente.

- I nodi vicini (quelli appartenenti al set) scambiano periodicamente col nodo keep-alive messages.
- Se il nodo non è attivo in questa corrispondenza per un tempo T allora si suppone che quel nodo non sia più presente all'interno della rete.
- I nodi appartenenti al leaf set del nodo fallito vengono avvertiti, questi di conseguenza aggiornano il loro leaf set rimuovendo il nodo fallito.
- La routing table è riparata con calma (*routing table entries that refer to failed nodes are repaired lazily*)

E se un nodo ritornasse? Chiamiamo tale nodo ***recovering node***.

- Esso contatta i nodi del suo ultimo leaf set per ottenere i loro leaf sets.
- Aggiorna il suo leaf set usando i dati ottenuti e notifica i nodi del suo nuovo leaf set segnalando la sua presenza.

5.5.2.7 Pastry API

Abbiamo visto nell'introduzione due funzioni sempre previste (eventualmente in forma diversa) in DHT. Cosa abbiamo con Pastry?

- **`nodeId = pastryInit(Credentials)`**
  Funzione che permette a un nodo di accedere a una rete Pastry. Inizializzazione di tutto il necessario e restituzione del nodeID. Credentials è un parametro che contiene informazioni necessarie per l'ingresso del nodo all'interno della rete Pastry.

- **`route(msg, key)`**
  Il messaggio viene posto nel nodo (tra quelli appartenenti alla rete Pastry) col nodeID numericamente più vicino a key.

- **`send(msg, IP-addr)`**
  Il messaggio viene inviato al nodo avente l'indirizzo IP indicato.

- **`deliver(msg, key)`**
  Funzione invocata quando un messaggio è ricevuto dal nodo. Valida sia per i messaggi inviati con route che per quelli inviati con send.

- **`forward(msg, key, nextId)`**
  Funzione invocata da Pastry per inoltrare la coppia ⟨msg,key⟩ al nodo con nodeID nextID.

- **`newLeafs(leafSet)`**
  Funzione invocata da Pastry quando emergono modifiche nel leaf set.

5.5.3 Proposta di sistema Publish-Subscribe: Scribe

Premessa: non è nel programma, ma il professore ogni tanto lo chiede per alzare il voto

5.5.3.1 Introduzione

Implementiamo sopra Pastry un sistema Publish-Subscribe: Scribe!

- Un nodo appartenente alla rete può creare un gruppo.
- Gli altri nodi possono: aderire al gruppo, inviare un messaggio multicast a tutti i membri del gruppo. Per quest'ultima cosa si richiede la creazione di un *multicast tree*.
- Basato sulla rete peer-to-peer di Pastry.
- Completamente decentralizzato in quanto ogni decisione è presa sfruttando informazioni locali. Ogni nodo, inoltre, ha le stesse capacità degli altri nodi.
- Soluzione perfettamente scalabile: i gruppi possono essere di dimensionare variabile, molto piccoli ma anche molto grandi.

5.5.3.2 Funzioni implementate

Scribe prevede le seguenti funzioni (*credentials* identifica il nodo che sta eseguendo queste funzioni):

- **`create(credentials, groupId)`**
  Creazione di un gruppo identificato da un groupId.
- **`join(credentials, groupId, messageHandler)`**
  Il nodo locale richiede di accedere al gruppo identificato da groupId. I messaggi multicast per quel gruppo, a partire da questo momento, saranno inviati anche a questo nodo e gestiti dal messageHandler indicato.
- **`leave(credentials, groupId)`**
  Il nodo locale abbandona il gruppo identificato da groupId.
- **`multicast(credentials, groupId, message)`**
  Invio di message a tutti i nodi appartenenti al gruppo identificato da groupId.

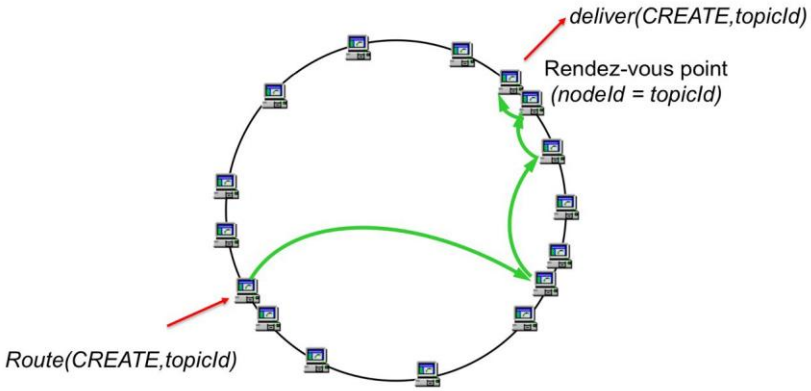
5.5.3.3 Implementazione

Come già detto si fa affidamento a Pastry, che si trova nel livello inferiore.

- Ogni group è pensabile come uno dei topic detti all'inizio: è identificato da un topicId univoco.
- Definiamo all'interno della rete Scribe il rendez-vous point:
  - Il nodo Scribe col il nodeId numericamente più vicino al topicId
  - Il nodo scribe che presenta la radice del multicast tree relativo al gruppo.
- Il topicId è generato per mezzo di una funzione hash SHA-1, dove poniamo in ingresso il nome testuale del gruppo concatenato al nome dell'autore del gruppo. La funzione hash adottata garantisce una distribuzione uniforme dei topicId, in linea con quanto visto in Pastry.

La presenza del multicast tree richiede l'implementazione di funzioni *forward* e *deliver* a livello Scribe.

- **Creazione del topic**.
  Cosa succede quando un particolare nodo lancia la `funzione create(credentials, topicId)`?
  Avviene l'invio di un pacchetto che segnala la creazione del gruppo identificato da topicId: a seguito del routing il pacchetto arriva al rendez-vous point, che come abbiamo detto prima è il nodo con nodeId numericamente più vicino al topicId. Si crea in questo nodo la radice del multicast tree.



- **Implementazione delle sottoscrizioni**.
  Le sottoscrizioni sono implementate con aggiornamenti del multicast tree.

[non lo ha finito di spiegare….]