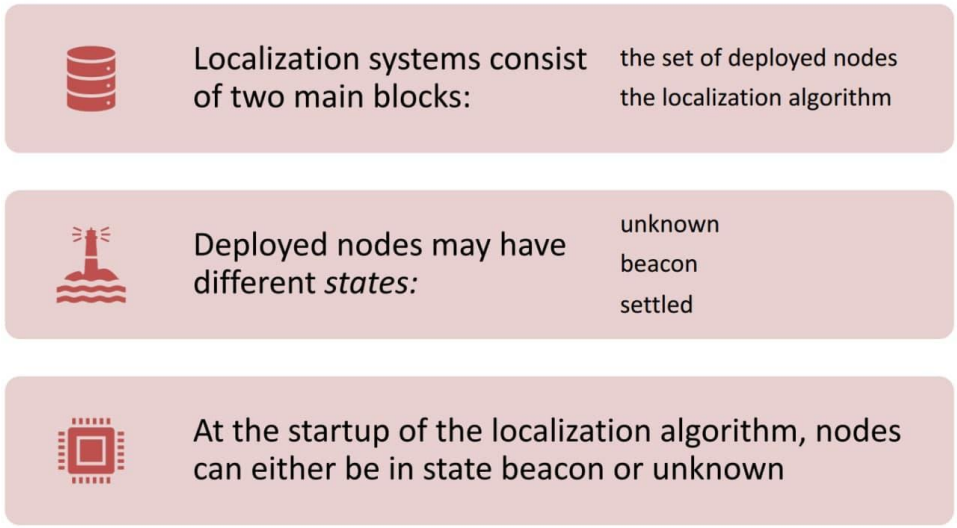
Wireless Localization Techniques

A **Cyber-Physical-System(CPS)** is the new generation of networks and embedded systems.
A CPS is made up of a computer system in which a mechanism is controlled or monitored by computer-based algorithms. Moreover, physical and software components are deeply intertwined and, because of that, they are able to operate on different spatial and temporal scales.
CPS can interact with each other in ways that change with context.

A **Wireless Sensor Networks(WNS)** is a networks of spatially dispersed and dedicated sensors that monitor and record the physical conditions of the environment and forward the collected data to a central location. WNSs are built of **"nodes"**, from a few to hundreds or thousands, where each node is connected to other sensors. Each such node typically has several parts: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit for interfacing with the sensors and an energy source, usually a battery or an embedded form of energy harvesting. Moreover, WSNs can measure environmental conditions such as temperature, sound, pollution levels, humidity and wind.

Localization is CPSs or in WNSs is the ability to determine the position of Components, Sensors and Events. Localization enables *location-aware applications* or support *network services* such as track objects or evaluate network coverage.

Taxonomy

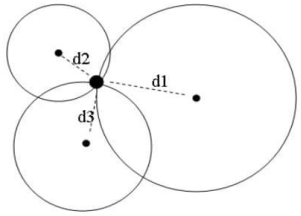


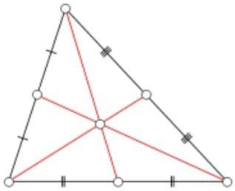
Nodes

The first two state are ***Beacon*** and ***Unknown***. Beacon nodes are also referred as landmarks or *anchors*. Their position is already known through a manual placements or through a GPS reading. On the other hand, unknown nodes, also referred as *target* nodes, are those that do not have any information about their geographical location.

Over the time nodes can change their state from *unknown* to *settled* if they succeed to determine or to estimate their location. Both beacon and settled nodes are useful for unknown nodes in order to estimate their location, they can be used as reference nodes. Because of that we usually evaluate the distance between targets and anchors using two approaches:

- **Trilateration**: it is a range-based method that estimates the position of a node using the distance from the anchors. In a two dimensional space we need the distance from three anchors in order to estimate the position of a target.
- **Centroid**: it is a range-free method, this uses the median of a triangle in order to estimate the position.





Algorithms

Localization algorithms aim at finding the position of unknown targets, there are different ways that can be determined based on the cyber-physical application. *Global Positioning System (GPS)* is the most intuitive solution to determine location accurately, however this is not the optimal solution because for CPSs due to cost and energy constraints, moreover GPS do not work well indoor or when satellites are shielded.

Alternative solutions can be used, these exploit the **sensing** and the **wireless communication capabilities** of CPSs components. For example we can use the *Received Signal Strength (RSS)* or propagation delay to infer distance or angles to some reference nodes, with these information we can calculate the position using simple geometric computations. We can also estimate distances by exploiting difference between heterogeneous waves' propagation properties. We can also use range-free localization using proximity and connectivity information.

Topology

Topology states **where** and **how** the location of a given node is calculated, there are four different system topologies:

- **Remote positioning**: the location of a node is computed at a central base station, where anchor nodes collect transmitted radio signals from the mobile node and forward them to the base station. The latter computes the estimated position of the mobile node based on the measured signals forwarded by the anchors.
- **Self-positioning**: the location of a mobile node is computed by itself based on the measured signals that it receives from anchor nodes.
- **Indirect remote positioning**: like in the self-positioning case, the mobile node computes its locations and sends it to a base station through a wireless back channel.
- **Indirect self-positioning**: like in the remote positioning, the location of the mobile node is computed at the base station, then the base station forwards the estimated position to the mobile node, through a wireless back channel.

Basically, we can distinguish between two different approaches, **Centralized** or **Localized**.

- **Centralized**: in this approach a central device estimates the position of an unknown device using signal measurements forwarded by the anchors. This results in a better accuracy, on the other hand it is not scalable and it is based on a multi hop strategy.
- **Localized**: in this approach each unknown node estimates its position using the collected information of the anchor in its neighborhood. This method is much simpler but it uses more iterations and computations, resulting in a higher energy cost.

Coordinate system

Coordinates can be expressed as ***Absolute*** or ***Relative*** then they can be represented in a ***Physical*** or ***Symbolic*** way.

- *Absolute*: locations are expressed as unique coordinate values referring to special nodes that know their positions, which are mainly anchors.
- *Relative*: locations are determined relatively to other nodes with no absolute anchors.
- *Physical*: location are represented as point in a 2D or 3D coordinate system, such as the Universe traverse Mercator(UTM) or the Degree/Minutes/Seconds (DMS) used for GPS.
- *Symbolic*: locations are expressed as logical positions' information, such as cell number or building number.

Performance Metrics

There are some metrics through which we can measure the performance of a localization method.

- Accuracy: mean distance error, usually the average Euclidean distance between the estimated location and the true location.
- Precision: reveals the variation in algorithm's performance over many trials. Usually, the cumulative probability functions (CDF) of the distance error.
- Complexity: can be attributed to hardware, software, and operation factors. Usually, location rate is an indirect indicator for complexity.
- Robustness: the system's capability of working when some signals are not available (e.g., when some of the RSS value or angle character are never seen before).
- Scalability: the positioning performance degrades when the distance between the transmitter and receiver increases. A location system may need to scale on two axes: geography and density.
- Cost infrastructure, maintenance, additional nodes...

Communication Paradigm

This section describes how nodes can exchange messages between each other.

- **Non-cooperative**: communication is restricted between unknown nodes and anchors, after the unknown node have received a certain number of information, no more communication is needed. There is the need of a high density of anchors or long range anchor transmissions to ensure that each unknown node is reached at least from three anchors.
- **Cooperative**: in this approach communication between unknown nodes is permitted, with this addition the anchor density can be lowered, on the other hand we need more processing.
- **Opportunistic**: with this approach we exploit interactions between existing nodes and other nodes that occasionally pass their proximity, this method requires an efficient node discovery and data exchange.

Range-Based vs Range free:

Range-based methods are distance-estimation- and angle-estimation-based techniques.

Range-free methods requires no additional hardware and instead uses the properties of the wireless sensor network and the appropriate algorithms to obtain location information. They rely on the availability of point-to-point distance or angle information.

Range-Based Methods

The location discovery consists of two phases: the **Ranging phase** and the **Estimation phase**. During the ranging phase each node estimates its distance or angle from the neighbor anchor nodes. During the estimation phase nodes can combine the information gathered in the previous phase and the position of the anchors to estimate their location.

During the **ranging phase** we can measure the distance using different methods:

- *Strength of wireless signal*: since the strength of the signal decreases with the increment of the distance, we can easily compute the distance in this way.
- *Time of Arrival(ToA, TDoA)*: we can compute the distance using the signal's arrival timestamp using the propagation speed of the signal itself. There are some drawbacks that can be solved using the time difference of arrival, we measure the difference of two different waves.
- *Angle of Arrival*: in this case we need a special direction antenna that can provide both the angle and the power of the signal.

Whereas, during the **estimation phase** there are three main techniques that we can use to estimate the position of a node.

- *Trilateration*: we can use the anchors to draw circles and we can estimate the position of a node into the intersection of all the circles that we are considering. We measure the distance from reference nodes (anchor)
- *Triangulation*: if we know the angle of the signals we can easily draw a triangle and estimate the position of the node inside that triangle. The smaller is the triangle, the more accurate is the estimation. Because of that is we have high anchor density the triangle is smaller. Even in this case we can talk about a self-position schema with non-cooperation approach. We measure the angle from reference nodes (anchor) and estimate the vertexes using the cosine formulas.
- *Multi-lateration*: we may receive information from an higher number of anchors, in this way we obtain a polygon. We can estimate the position of a node using the centroid of the polygon, the smaller it is the better is the accuracy.

Note: Those methods can be applied in different configuration of topology and communication methods. For example, we can apply self-positioning and non-cooperative with trilateration or triangulation, in this case the unknown nodes use the received signals to measure the angles/distance from the anchor nodes and the compute the position by themselves (without sending nothing to the bs). We can also use other approaches.

Received Signal Strength (RSS)

Target nodes estimate their distance from anchors using the attenuation of emitted signal strength. In free space the Power of the received signal follows the following formula.

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2}$$

Where P_t is the power of the transmitted signal and G is the gain of the antennas. That equation only works in free space and ideal condition. In real world there are some problems due to reflection, refraction, scattering etc. and also due to different obstacles. Because of that the relation above does not work anymore, in particular we cannot use the previous value of the gain of the two antennas, also the relationship with the distance can change.

To solve this problem, we can consider the CPS to operate in free space, for example we are in an open space outdoor with few obstacles, in this case we can use the canonical formula. The other option is to compute a **site-specific** new equation, this has to be done a priori, this is more expensive. For a site-specific relationship we can use $P_{RSSI} = \frac{X}{r^n}$ where X is a generic constant, we can observe that the distance can have also powers greater than 2 (n in this case). If we want to deploy the application in another site, we must recompute the relationship.

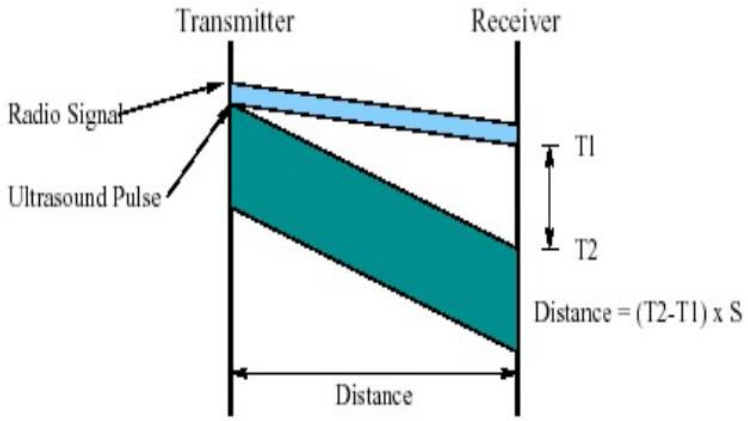
Time of Arrival

This method is based on the propagation time of the signal. Once we know the propagation speed, if we can measure the time spent (the difference between timestamp_arrival-timestamp_departure), we can easily calculate the distance between two nodes. In 2D positioning ToA measurements must be made with respect to at least three anchors. The problem with this approach is that if we use Radio Frequency, it has roughly the speed of light as propagation speed, so nodes have to be precisely synchronized because even 1ms error can be translated in 300km error in positioning. This type of synchronization can be acquired with an indoor CPS, but we cannot relay that synchronization on wireless sensor network or, even worse, with mobile network wireless sensor system.

$$d = c * (t_2 - t_1)$$

Time Difference of Arrival (TDoA)

This is a better technique since we do not need a synchronization among clocks. We will use a time difference at the receiving node, we will use two types of signals: *radio frequency* (RF) and *Ultrasound.* We have to compute the difference between the time of arrival of the RF signal (light blue) and the time of arrival of the ultrasound signal (green).



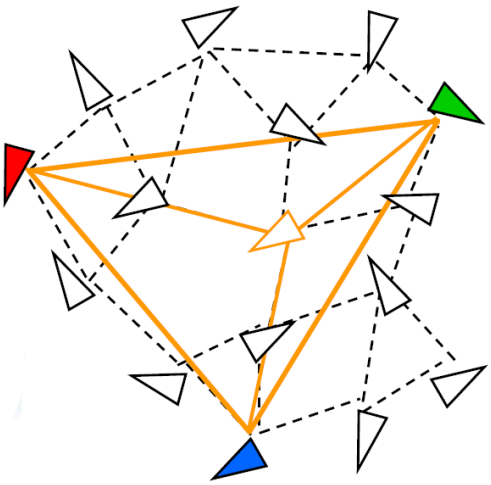
With this two information we can estimate the distance between the anchor and the target node using the speed of sound and the normal speed formula. The main advantage of this technique is that using the

speed of sound we do not have to relay on precise clock synchronization since speed of sound is much smaller than speed of light, because of that small error on the time difference results in small error on distances. There are some drawbacks, first of all we need extra hardware because we have to transmit and receive RF and Ultrasound signals, because of that we may also have higher energy consumptions.

The communication protocol is simple because only anchors have to send beacon messages and we do not need target node to exchange messages among them.

Angle of arrival (AoA)

This approach is based on the angle of arrival rather than on the time of arrival. In this case, we need not only to receive the message, but also to know the angle from which the message has been sent. To do that we need special hardware, in particular we can use a directional antenna or an array of antennae mounted on each node.



In order to know the angle of arrival we can simply give and index to each antenna that is directed along the main direction. At the receiving node we can use that index to compute the receiving angle, that can be used based on one of the following configurations.

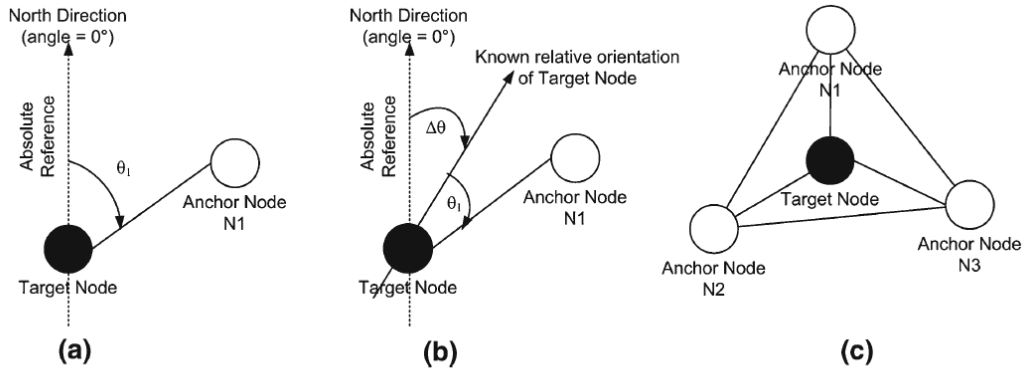


Fig. 7 AOA orientation concept. **a** Absolute orientation. **b** Relative orientation. **c** AOA with unknown orientation

For example, in case **C** we can draw a triangle using trigonometry and we can estimate the position of the target node inside it. Of course, we still have an error on accuracy that depends on the area of the triangle.

This type of technique is not used frequently since it requires additional hardware and it is expensive to deploy on large sensor network.

Range-Based Limitations

In this paragraph we will analyze the main limitations of the range-based techniques. Many of them are **based on assumption that do not always hold** or are impractical. Among them:

- Circular radio range: with this assumption we have an **isotropic** system, that means that we can use the same formula between the receive signal strength and the distance independently of the direction from which we see the messages and so on. In a real cases we never have a circular radio range because the environment introduces many inhomogeneity.
- Symmetric radio connectivity: this means that if I am able to receive a beacon message from an anchor, then also the anchor should be able to receive my messages. In the real world this is not always true, target nodes may not have RF connectivity.
- Additional hardware: as discussed before range-based approaches may need additional hardware to work properly.
- Lack of obstructions
- Clear line-of-sight: for example in TDoA we use two different type of signals, because of that environmental obstacles must be absent to ensure good communication.
- No multipath (multipath=reflection of signals) and flat terrain: otherwise noise and inhomogeneous propagation may occurs.

Range-Free Localization

Range-Free methods do not rely on distance or angle estimation during the localization phase, they use proximity or connectivity information to devise the location of the target. In that way nodes never try to estimate their absolute distance from other anchors.

Range-free localization methods are divided into two main parts:

- Area-based approaches: that are the **centroid localization** and the **APIT (approximate point-in-triangle test)**.
- Multihop localization approaches: one of them is the **DV-HOP**.

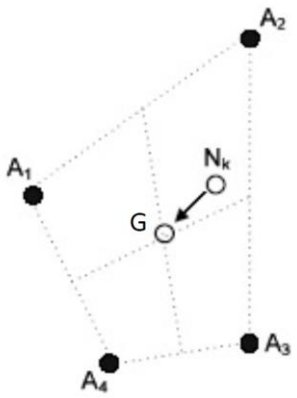
The main difference between these two parts are physical in the sense that with area-based approaches we need to assume that all the target nodes are in the radio range of anchors so we need high density of anchors or a large radio range with less anchors, this because these methods rely on a **single hop approach**. On the other hand, when we cannot guarantee a high density of anchors we need to use the multihop approach. We ask target nodes to spend energy to compute their location and also to contribute in the communication protocol, in other words target nodes have to forward beacon messages.

About this localization techniques we can have some advantages like **cheap hardware** and **low computational power**, on the other hand we have less accuracy compared to the *range-based* methods.

Centroid Method

This is an *Area-Based* method, this means that the target node, at some point, must be able to draw a polygon whose vertices are the anchor from which the target node receives beacon messages. The higher is the number of anchors, the more precise is the estimation of the target node. Once we have drawn the

polygon we can estimate the position as the **geometric center (barycenter)** of that polygon that can be computed locally pretty easily.



$$(X_G, Y_G) = \left(\frac{\sum_{i=1}^{n}(X_i)}{n}, \frac{\sum_{i=1}^{n}(Y_i)}{n} \right)$$

The payload of the beacon messages contains the coordinated of the anchor nodes, we have to wait a certain time so that we are sure that the majority, or possibly all, the target nodes have received beacons from at least three anchors. After this amount of time that can be configured each target node can draw its polygon and compute the barycenter. (the minimum number of beacons that have to be received depends on the cardinality of the space. 2D -> 3, 3D -> 4).

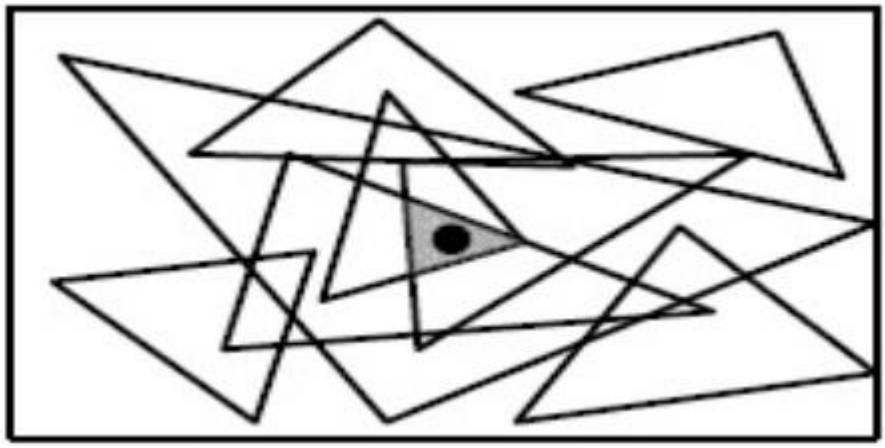
There is a tradeoff between the anchor density and the radio range amplitude. With an high anchor density we can have a small radio range since we can cover up the area correctly, with a low anchor density we must have a huge radio range in order to have a good coverage.

Advantages and disadvantages

Talking about advantages of this method, it is simple and easy to implement and it has a small overhead since we use few beacons and the computations are easy to execute. On the other hand we have a low accuracy and we need overlapped anchors to have a correct estimation. An overlapped radio range means that the anchor's range must intersect each other.

APIT

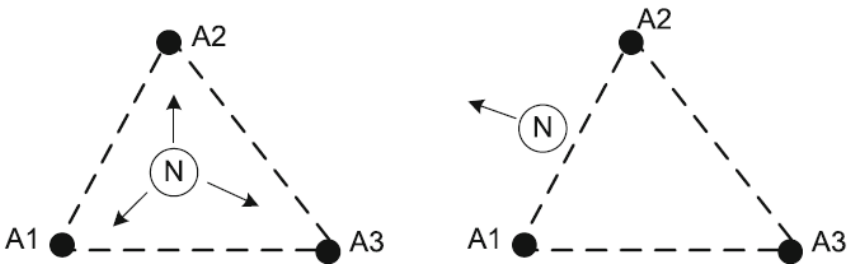
The idea of this method is similar to the one seen before. In this case we can exploit the high number of beacon that a target node can receive from anchors to build a large number of triangles. After that we can retrieve the list of triangles that contains the target node, we have to take the intersection and to estimate the position of a target node we have to compute the center of gravity of the intersection as shown in the image below. Obviously the area with this method is much smaller than the area found with the centroid method.



In this method we assume that each target node can recognize if it is inside or outside a given triangle. In the previous approach we could not know if we were inside or outside the polygon so we make an approximation error due to the center of gravity, moreover we make an error that comes from the fact that we do not know the position of the target node.

Point-In-Triangulation Test (PIT)

This test determines whether a target node with unknown location **N** is inside a given triangle or not. First of all, we have to be in radio range of the anchors in order to receive their beacon that contains the coordinates, if we receive more than 3 beacons, let's suppose n, we can draw $\binom{n}{3}$ triangles.



The **perfect PIT test** follows two main propositions:

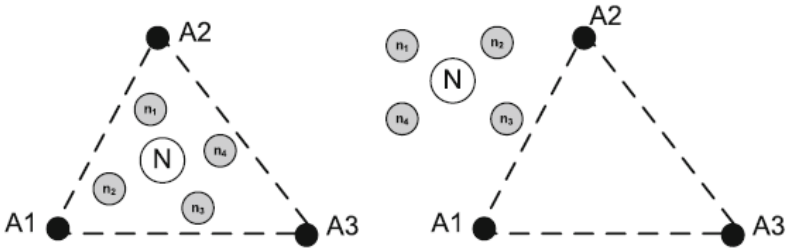
- Proposition 1: if **N** is INSIDE the triangle, when N moves in any direction, the new position will be nearer to (further from) at least one anchor. (This is shown by the left image.)
- Proposition 2: if **N** is OUTSIDE the triangle, there exists a direction so that, when N moves along it, the position of N gets far (or close) to all the three anchors at the same time. (This is shown by the right image.)

The perfect PIT test is unfeasible in practice for two main reasons:

1. Nodes typically do not move, and they do not have the ability to recognize the direction without moving.
2. It is not possible to perform an exhaustive test covering all the possible direction the target may move to.

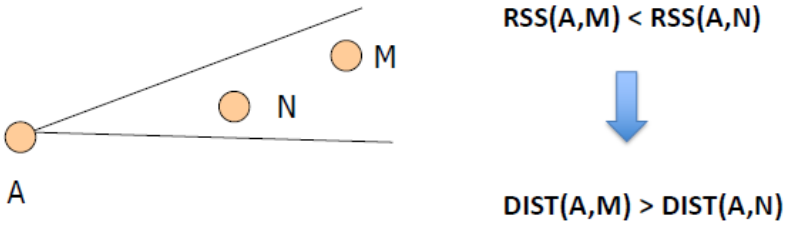
There is a solution to these problems, an approximation has been proposed. The idea of this new solution is to use neighborhood information that is exchanged via beaconing. This is done in order to simulate the movement of the perfect PIT test. This new solution is called **Approximate PIT (APIT) test**.

Since we have a high density of nodes we can exploit the neighbors in order to simulate the movement of the target node. Let's suppose that we have a group of neighbors that share the same triangle, we can obtain the distance between the neighbors and the anchors. With this information we can substitute the movement of the target node and estimate whether it is inside or outside the triangle.



If no neighbor is further from (or closer to) all three anchors simultaneously, then the target assumes itself as being inside the triangle. Otherwise, the target assumes it is outside the triangle. The problem is that we would like to exchange the distances, but this is not possible since we are using a **range-free approach**.

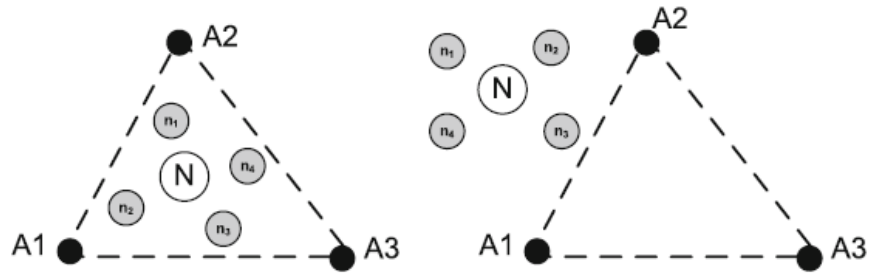
We do not need an absolute distance, we can use a relative distance in order to estimate if the distance from the target node to an anchor is higher than the distance from a neighboring node to the same anchor. To compute this estimation we can use the *Received Signal Strength (RSS)* as shown in the image below. The further away a node is from the anchor, the weaker the received signal strength is.

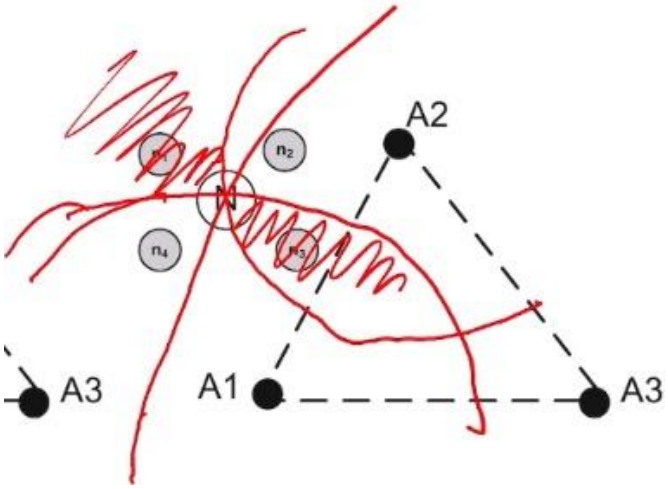


Note: Using RSS does not make this method range-based, this is because nodes do not compute a distance from the signals. They just share their RSS values.

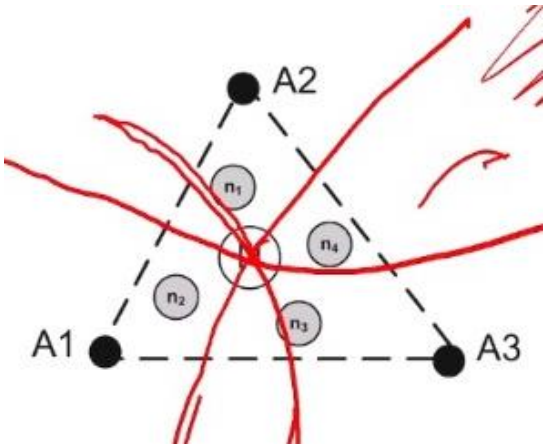
Let's analyze the following image, we can see that n_1 is further from all the other anchor, whereas n_3 is closer to all the anchor simultaneously. The other two neighbors are misleading, if we draw the three red circles centered in the three anchors we can state that n_2 is closer to A2 and A3 and further from A1, a similar situation occurs with n_4. This is not a problem since Proposition 2 of the perfect PIT test states that there must be at least one direction that the target node can follow in order to get further (or closer) to all three anchors simultaneously, in this case we have two directions, one along n_1 and one along n_3. In other

words it's not necessary that the condition is respected for all the neighbours.

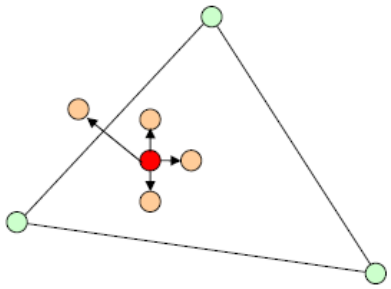




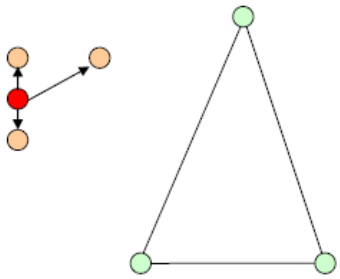
Now we will analyze the case in which the target node is inside the triangle. We have to draw again the three circles, in this case we can state that, for example, n_1 is close to A2 but further from A1 and A3, similar situation for n_2 and n_3, whereas n_4 is closer to A2 and A3 but further from A1. This information is consistent and allows us to state that the target node is inside the triangle.



We may commit some errors using this approximate test as we can see in the image below. In the left side, the target node set itself as outside the triangle because one of its neighbors is further from all the anchors simultaneously, on the right side target node estimate itself as inside the triangle because using its neighbors, no one is closer to (or further from) all the anchors simultaneously, this is enough to state that the target is inside.

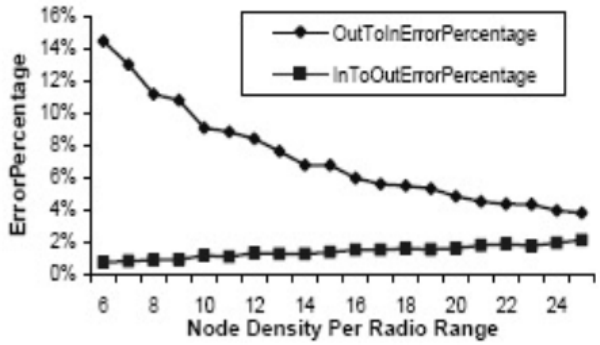


In-to-out error



Out-to-in error

We can solve this problem if we have a high density of node per radio range as we can see from the chart below. After a certain threshold, for example 25 nodes per Radio Range, we can assume the error rate of the two types equal and about 3%.

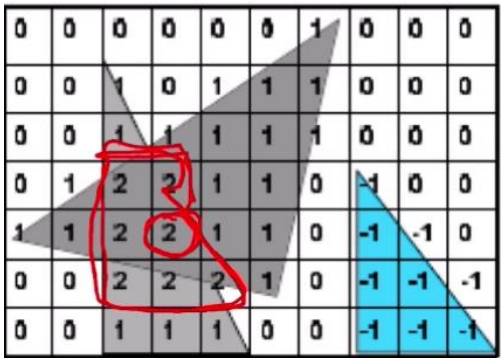


The algorithm works as follows:

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($\bigcap T_i \in InsideSet$);

We have to create a large number of triangles and then we have to apply the approximate PIT test to create a list of triangles. Once we have the list, we have to intersect all the triangles in the list and compute the center of gravity of the intersection to obtain the approximation of the target node.

In order to compute the intersection, we can rely on a grid surface, as shown below, if the target node is inside a triangle we can increment the counter, if the target node is outside we decrement the counter. This results in an area which the counter is maximum that is the intersection of which we have to compute the center of gravity.



Note: In this example the location of the node is in that red circle.

Advantages and drawbacks

Talking about advantages, this method has a small overhead and it is more accurate than the centroid one, this is due to the smaller area in which we compute the center of gravity. There are also disadvantages, the accuracy is strongly related to the density of nodes and anchors, moreover there is a problem in determining the location of a node that is outside all anchor triangles.

DV-HOP

This method is based on a *multi-hop communication* since it not always possible that a target node is in communication range with at least three anchors, this is due to transmission limitation. To overcome this limitation, we can adopt multi-hop communication in order to expand the communication area.

DV-HOP is a range-free method so we will not compute absolute distances, we will obtain a distance estimation starting from a logical distance that consists of the number of hops that the beacon travels from the anchor to the node.

The network is initialized such that each anchor broadcasts a beacon that contains the coordinates of the anchor and a hop counter, this counter is increased each time the beacon is forwarded. In this situation a target node receives a beacon from each anchor and not only to the ones that are in radio range, with the hop counter we are able to obtain the physical distance. Before translating we have to know the average length of a single hop in meter. This length depends strictly on the network that we are working with, moreover the length can also vary among different regions of the same network since we are in a real situation. If we compute the 1 hop length locally we reduce the error with a consequent increase in accuracy.

This method is made of two phases:

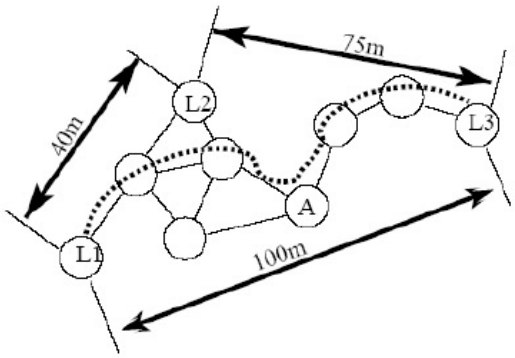
- **Node update phase**: in this phase a target node receives beacon messages originated from anchors, also the anchors receives beacons of other anchors. For each anchors a target node stores the anchor's coordinates and the hop counter, after that the target node increments the hop counter and broadcasts the beacon to the nodes in its radio range. At the end of this step each target node knows the location of each anchor and the hop necessary to reach them. Also anchors have this type of tables and, since anchors knows their location, they can compute the Euclidean distance among anchors.
- **1-hop distance estimation phase**: when an anchor receives the location and the hop counter of another anchor, it can compute the estimated 1-hop distance, referred as *correction factor* c_i, with

this formula: $c_i = \frac{\Sigma(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2})}{\Sigma h_i}$. Where i is the anchor that computes the correction, j are the anchors known to i, x and y are the coordinates and h is the hop count (the sum is the sum of all hop counts from other anchors to this one). The correction is a measure in meters. At the end of this phase each anchor has computed the correction factor. Anchors will broadcast the correction factors to the target nodes, in this way target nodes are able to obtain the estimate distance in meters.

A node that receives the correction, stores it and forwards it and then stop forwarding additional correction that may receive. In this way each node only stores the correction factor of the nearest anchor. This is done because, as we said before, it is a good idea to compute the physical length locally, because of that we only save the nearest correction factor.

A target node can localize itself by multiplying the correction factor and the hop count of at least three anchors, with this minimum number trilateration can be used to estimate the location.

In the following image is shown a simple example of a DV-HOP method. *A* is a target node, all three anchor nodes can compute the correction and start forwarding it. Since L2 is the nearest anchor to A, A will save that correction factor. A can compute the distance from all the anchors with a simple multiplication.



L_1 computes the correction $\frac{100+40}{6+2} = 17.5$.
L_2 computes a correction of $\frac{40+75}{2+5} = 16.42$
L_3 a correction of $\frac{75+100}{6+5} = 15.90$.
A gets its correction from L_2

Distance to L1 = 3 x 16.42, to L2 = 2 x 16.42, to L3 = 3 x 16.42

Advantages and Drawbacks

The main advantages is that this method is very simple to implement. On the other hand, there are some drawbacks for example the is a large overhead due to the broadcast phases, the estimation error depends on the number of anchors that a node can hear, of course, the higher is the number of anchors the more precise are the estimation (in pratica, se ho tante anchor che mandano beacon posso trovare un correction factor più preciso che mi porta a ottenere una stima della distanza fisica migliore). The last drawback is that this method only works in isotropic networks otherwise the correction factor will be affected by errors.

Distributed Hash Table (DHT)

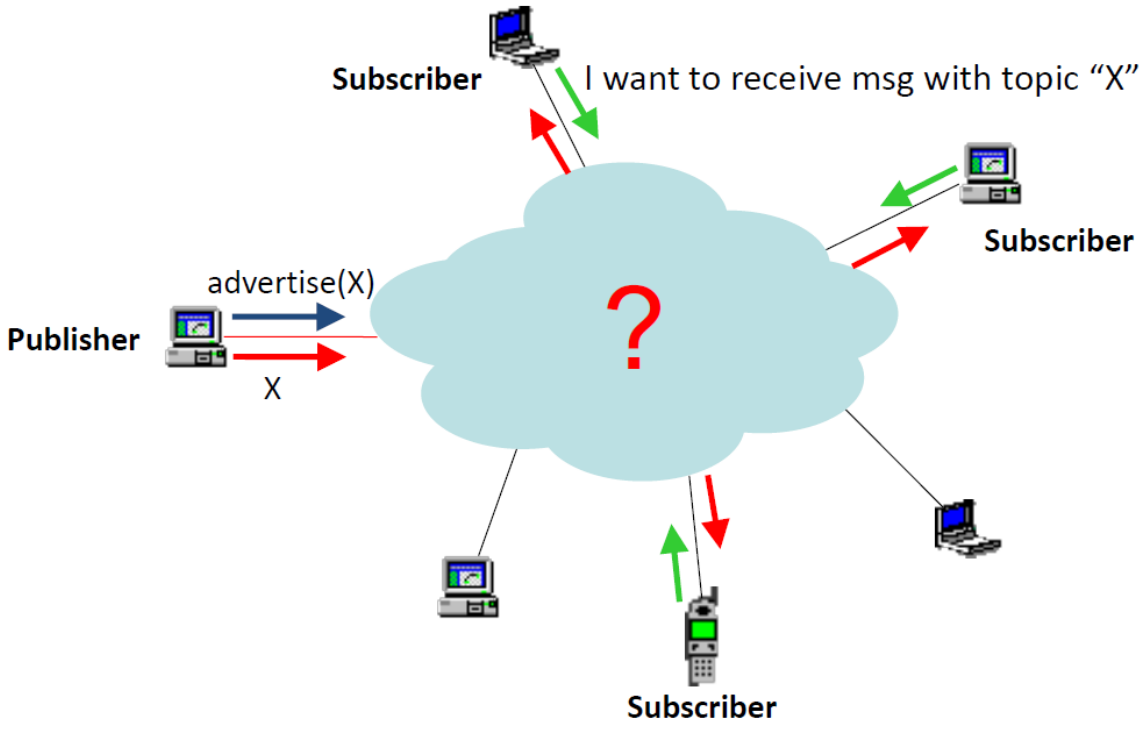
The Publish-Subscribe pattern

This pattern is a messaging system that can be implemented in any way. There are nodes that act as **publisher** and nodes that act as **subscriber**. The role of the publisher is to create content and messages about a given topic, the interesting part is that a publisher does not send a message to a specific receiver, instead the message is advertised and it is received only by the nodes which are interested in receiving messages of a specific topic.

First of all, when we implement a publish-subscribe pattern we must define how the messages are advertised in the network. We need to implement a **filtering** step, in this way the publisher can advertise its message in two different manners:

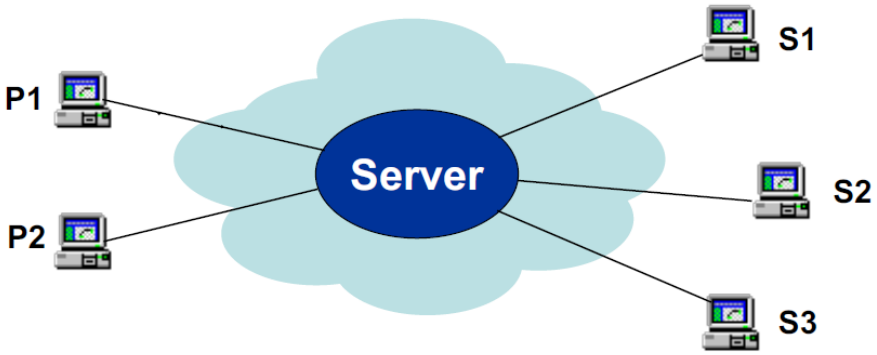
- **Topic-based**: with this approach messages are published to "topics" or named channels. The publisher is responsible for defining the topics to which subscribers can subscribe. A subscriber receives all the messages in a specific topic channel.
- **Content-based**: with this approach messages are delivered to a subscriber if the content or the attributes of the message match the constraints defined by the subscriber. This approach requires more computational resources since we have to analyze the content of the message and the subscriber is responsible for the classification.

In the following image is shown the classical view of a publish-subscribe system.



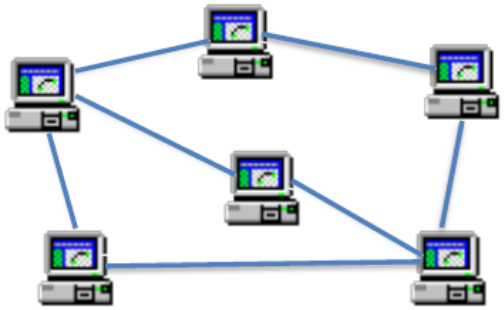
We have to guarantee that a publisher can advertise its messages, that the messages and the topics are stored somewhere and we also have to guarantee that a subscriber can receive the messages.

There are many ways in which we can implement this system, the first one is based on a **client-server** solution as shown below.

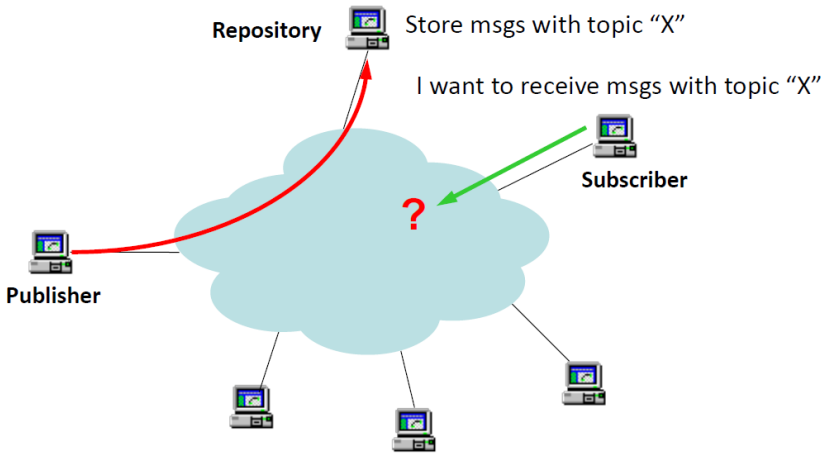


This solution is quite easy, on the other hand there are some disadvantages. This approach is not scalable, there is a lot of traffic and computation on the central server and there is the presence of the **single point of failure**.

Another way that we can use to implement this paradigm is the **Peer-to-Peer Architecture** as shown in the image below.

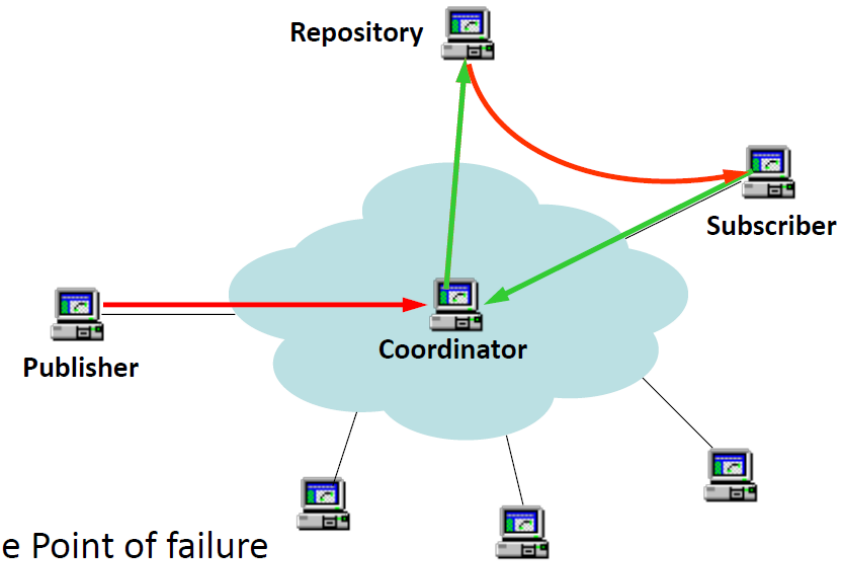


In this architecture each node is symmetrical in the sense that the role of each node can change over time from publisher to subscriber. Also, the number of nodes in the system can change dynamically, moreover in this implementation a node can also be a **repository node**. The architecture is made of a large number of heterogeneous and unreliable node, this ensures the fault tolerance. Using a distributed architecture brings to the so called **Lookup problem**, this occurs when a node has to retrieve the information because, in order to do that, it has to locate and contact the repository node. Also the publisher has to reach a repository node in order to advertise and store its messages.



P-S Centralized Approach

To solve this problem we add a new role, the **coordinator** node. This is still a distributed system since there can be more than one coordinator. By the way this is still a sort of centralized approach. The coordinator decides in which repository node a message must be stored. In this situation, a subscriber can send a message to the coordinator to tell him the topic of interest and then the coordinator trigger the correct repository to send the stream of topic's messages.



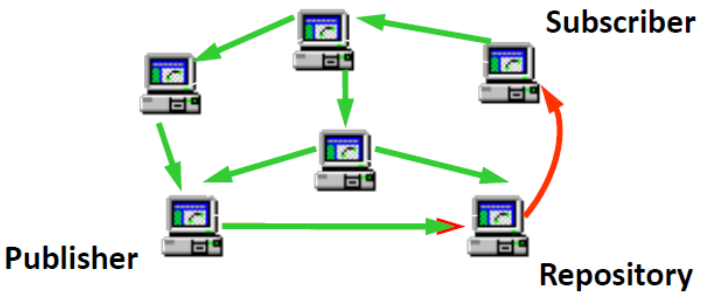
Single Point of failure

O(N) status in the coordinator node

In this hybrid approach we still have the single point of failure as main drawback. The coordinator node have to maintain O(N) status, these are the nodes that want to advertise and the node that are subscribed to a certain topic.

P-S Decentralized Approach

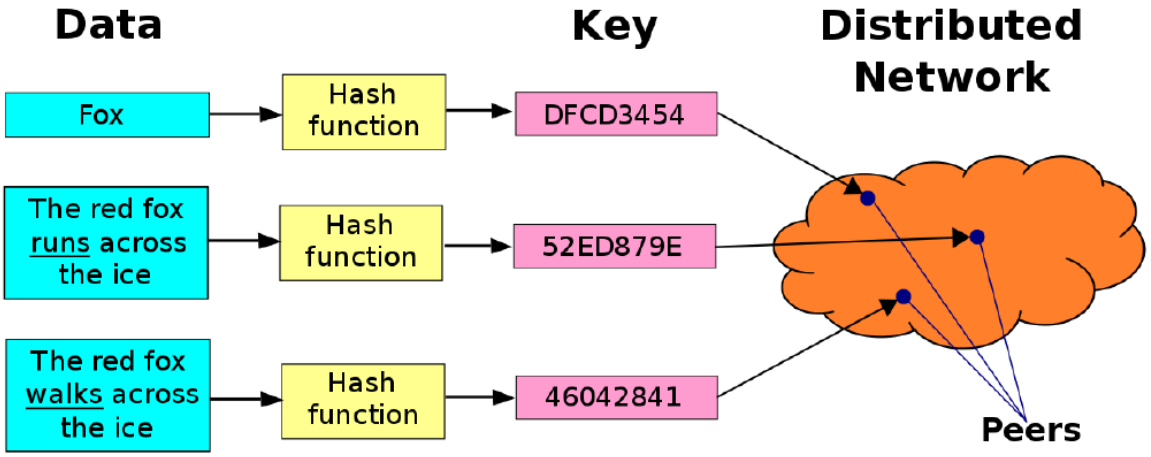
If we want to maintain a pure peer-to-peer architecture, we can simply flood the network with messages. If a publisher starts to produce a message of a certain topic it will inform all the nodes, the same happens when a subscriber node wants to start receiving the message. This brings to a high overhead in the entire network. In this case we have O(N) messages for the lookup.



DHT

We can introduce a new way to store data when they are associated with a **key**, we can use the key to find where the data are stored and by doing that we can retrieve them.

This distributed system provides a **lookup** service, this means that when we want to retrieve data we have to provide the associated key. In order to do that the system must store (key, value) pairs into the DHT. Since we have to provide the key, keys must be unique in the entire DHT, moreover any type of data have to be mapped with a key (addresses, document etc.). The mapping from key to value, that is where a certain value is stored, is distributed among all the nodes, we must select a rule that decides how to distribute the pairs. In this way we can redistribute the pairs in case of network changes. Because of that, nodes can be added or removed with minimum work of redistribution.

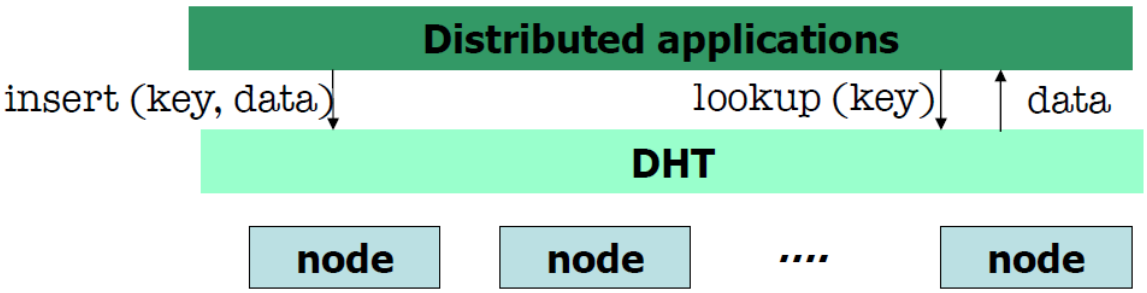


In the image above we can see how this approach works, we start from any kind of data in light blue, we apply an hash function to ensure the uniqueness of the key. Hash functions produce keys of fixed length, moreover they can also distribute the keys among the keyspace. Once we have produced the keys we have to decide a rule that can decides how to distribute the pairs among peers.

DHT Structure

We start from an **abstract keyspace**, for example the set of 160-bit strings. Then we have to decide a rule by which the **keyspace is partitioned** in order to split the responsibility of storing the pair among all the nodes. We also need an **overlay network** that connects the nodes, allowing them to find the correct owner of any given key in the keyspace, this depends on how the keyspace is partitioned.

DHT API



The insert function is used from the publishers when they want to advertise new messages. The implementation of this function guarantees that the (key, data) pair is store in one of the participants nodes. Nodes behaves as hash buckets so we can simply, for example, see the product of the hash as an integer number and store intervals of integers in each node. So adjacent nodes in the overlay network will

store adjacent interval of integer numbers. Overall we must guarantees that the keys are evenly distributed among the node.

The lookup function is used to locate the node that contains the value that we are looking for, finally there are some services that allow the data to be delivered from the repository to the asking node.

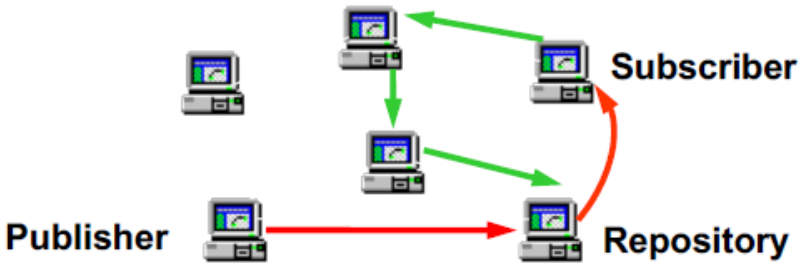
In the following image some properties are shown.

- *Autonomy and decentralization*: the nodes collectively form the system without any central coordination
- *Scalability*: the system should function efficiently even with thousands or millions of nodes
- *Fault tolerance*: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing
- *Load Balancing*: the hash function should evenly distribute keys to nodes
- *Anonimity*: can be achieved by special routing overlay networks that hide the physical location of each node from other participants

DHT application

- A DHT scales to extremely large numbers of nodes and can handle continual node arrivals, departures, and failures
- This allow DHTs form an infrastructure that can be used to build more complex services, such as instant messaging, multicast, peer-to-peer file sharing, content distribution systems
- In the Internet of Things, DHT-based peer-to-peer architectures can be used for Discovery Service and to manage the flow of data from sensors to applications

With the DHT the P-S approach is simpler since we do not have to flood the entire network as shown in the following image.



The complexity for a lookup message is O(log(N)), in this way a subscriber reaches the repository node in a more efficient way.

Pastry

Pastry is a peer-to-peer overlay network for DHT. In this method the key-value pairs are stored in a peer-to-peer network of internet host in a redundant way, redundance guarantees fault tolerance. With this method we need to design a routing schema that avoid the use of flooding. At the bootstrap pastry starts with the translation of the IP addresses of the participating nodes. The translation is done using a secure hash function that produces a fixed length integer that is called **NodeID**. At bootstrap the system has also to build the routing table for each participant in order to implement the routing schema on the overlay network. Because of its redundant and decentralized nature, pastry has no single point of failure and any single node can leave the network without warning with a little or no data loss.

Pastry Design

The most important thing is the rule that allows us to partition the key space, more precisely key are stored in the node with the NodeID **numerically closest** to the key among all live pastry nodes. In general the topology of the overlay network is **circular**. NodeIDs and Keys are **128-bit unsigned integer** represented in

base 2^b, the integer represents the position in the circular space. NodeIDs have to be chosen *randomly and uniformly*, in this way peers which are adjacent in NodeID are geographically diverse (maybe distant or not*). To produce a NodeID we can use a **secure hash function** of the node IP address.

***:** The overlay network is a logic network, nodes that are adjacent in this network may not be close physically in the real one.

Pastry Routing

Given a message and a key, Pastry reliably routes the message to the node that has the NodeID that is numerically closest to the key. In order to find the target node we do not want to travel the entire network using adjacent nodes, in this way the complexity will be O(N) with N number of nodes. We must be able to jump from a node to another in order to go as far as possible, with this method we can route to any node in less than $\lceil \log_{2^b} N \rceil$ steps on average. For example, with 128 bit for the NodeIDs we can have up to 2^{128} different nodeIDs and using $b = 4$ we will have a routing complexity of 32 steps to retrieve the target node. **Pastry can still deliver messages even with concurrent node failures unless $l/2$ or more nodes with adjacent NodeIDs fail simultaneously**, l is an integer parameter with typical value of 16.

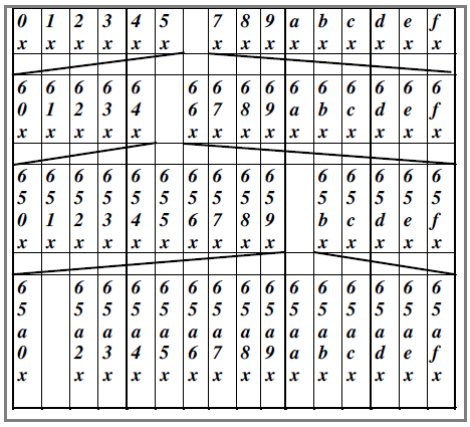
Note: In the example we have b=4, $N = 2^{128}$

Prefix Routing

The routing in Pastry is based on the so-called **Prefix Routing Schema**. As we said before NodeIDs and Keys are a sequence of digits with base 2^b. The **routing table,** that is initialized and stored in each node, is organized into $\lceil \log_{2^b} N \rceil$ rows and 2^b columns, this means that if *b* is equal to 1 we are representing the NodeIDs in a binary way so the binary table will have two columns. The entries in row ***n*** refers to a node whose NodeID matches the present node's NodeID (the NodeID of the host of the table) in the first ***n*** digits, but whose (n+1)th digit has one the $2^b - 1$ possible values other than the (n+1)th digit in the present's NodeID (and on the subsequent digits whatever). (In pratica controllo il NodeID del nodo attuale e per ogni riga inserisco i NodeID che hanno i primi n numeri, dove n è il numero della riga, uguali al NodeID del nodo attuale e il numero presente nella posizione n+1 vale uno degli altri possibili valori dati dalla base di rappresentazione tranne quello presente nel NodeID attuale. Ogni colonna corrisponde a uno possibile di questi valori.). Since each entry refers to one of potentially many nodes whose NodeID matches the actual prefix, we have to choose among them the one that is physically closest to the present node using a **proximity metric** like the round trip time.

In the image below is shown a routing table for a node whose NodeID is $65a1x$ and with $b = 4$, x represent an arbitrary suffix.

NodeId=65a1x



Note: the separations indicate a row, the columns inside that separation are the possible value of the row.

EG: in the second row (n=1 row) we can have 6-whatever_but_not_5-whatever etc. If in my network i have a lot of nodes that match the prefix how can i choose the one to put in that entry? Using a proximity metric, so the nearest one in the physical world (overlay network is not physical). At row 0 we must have the n(=0)+1 digit, so the first one, different from 6, so: whatever_but_not_6-whatever etc.

I cannot have 65x because at row 1 only 1 bit of the host prefix must match, in that case would be 2.

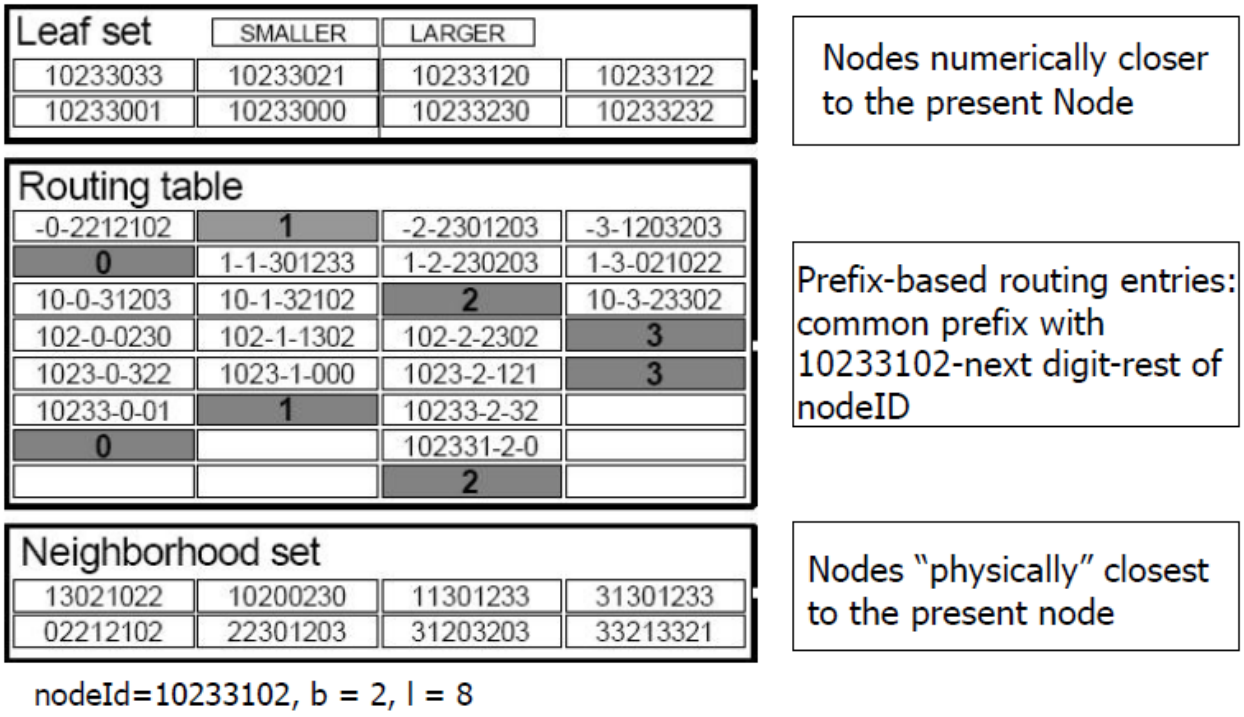
Since *b* is equal to 4 the NodeIDs are expressed as hexadecimal numbers. In this table the IP addresses are not shown, **we have to store them** since we have to route the messages. As we can see in row 0 there are the prefix that does not share any digit with the NodeID of the present node, as we increase the row number we have that an incremental number of digit must match the NodeID of the present node. The blank entries represent the fact that, for example in the row 0, we cannot have a NodeID that starts with 6 since it is the first digit of the present node.

Another property is that in the first row we store nodes that are **logically far (not physically)** from the present node, so when we choose one of these NodeID as destination we will make the longest jump in the overlay network. On the other hand, when we chose a NodeID from the last row we will make the shortest jump, this means that we are closer to the node that stores the key that we are looking for.

In addition to the routing table each Pastry node store a **leaf set**, in this set are stored the NodeIDs of the nodes that are numerically closer to the present node. In particular in the leaf set are stored the $l/2$ nodes with numerically closest larger NodeIDs and the $l/2$ nodes with numerically smaller NodeIDs. (se ci sono un set di nodi adiacenti si sceglie i più vicini seguendo la proximity metric).

Note: Each node of the network maintains the ip addresses of the nodes having nodeIds inside its leaf-set.

In the following image a complete routing table is shown.



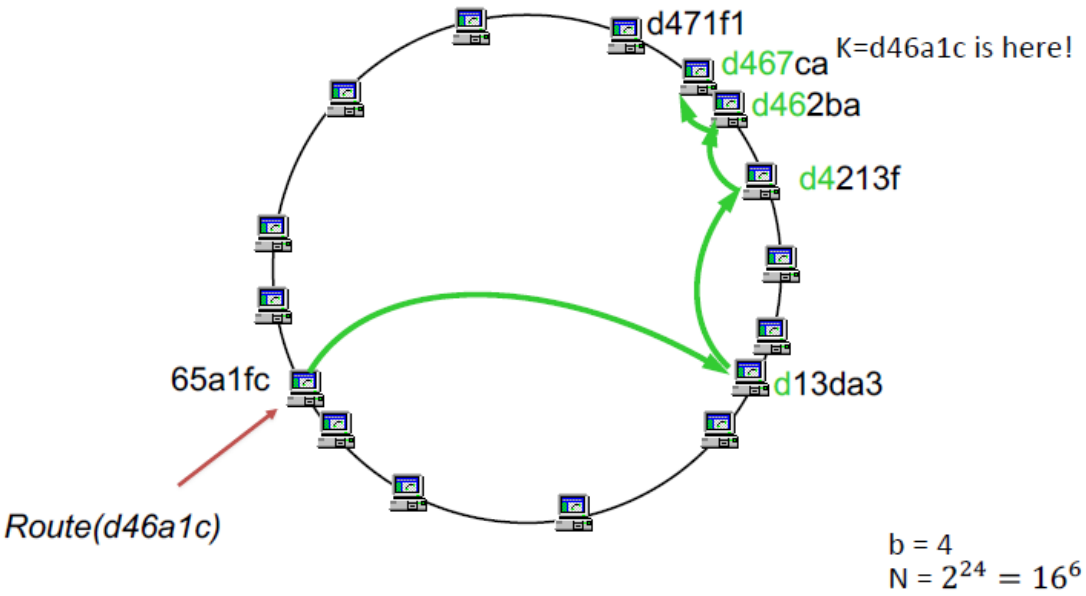
in this example l is the length of the NodeIDs, this results in a key space of 2^{16} bits since every digit requires 2 bits to be expressed since we are in base 4. (il numero fra "-" indica il numero che deve seguire il prefisso stabilito dal numero della riga). It may happen that some rows are empty or partially completed. Since we are using $l = 8$ in the leaf set are stored 4 nodes with lower NodeIDs and 4 nodes with larger NodesID, it is important to notice that because of the parameter every node in the leaf set shares 5 digit with the present node, from 0 to 4. In the **neighborhood set** are stored the NodeIDs of nodes that are physically closer to the present node. This set is useful for dealing with redundancy and with failing nodes, for example when the node is going to fail the node itself tries to upload its keys on the near nodes.

Routing Steps

In this paragraph we will describe how Pastry handle the full routing procedure. Let us suppose that a node receives a *lookup message*, this message has the key of the target node as parameter.

1. Seek the routing table: we need this step because the current node has to forward the message to a node whose NodeID shares with the key a prefix that is **at least one digit longer** than the prefix that the key shares with the current NodeID. We need to do that because each key is stored in a node whose NodeID is numerically closest to the key. The longer is the prefix shared between the key and the destination node, the longer will be the jump.
2. If no such node is found in the routing table, check the leaf set: this step is done because if we do not find a node in the routing table it means that we are numerically close to the target node, so we can search in the leaf set. The current node forwards the message to a node whose NodeID shares with the key **as long as** the current node, but the destination NodeID has to be numerically closer to the key than the current NodeID. With this step short jumps are performed.
3. If such node does not exist in the leaf set, the key is stored in the current node. (unless the $l/2$ adjacent nodes in the leaf set have failed concurrently).

In the following image is show an example of the step that we have just analyzed.



We are using a hexadecimal base since $b = 4$, because of that we can have up to 2^{24} nodes (that is 6 number of digit per NodeID multiplied by 4 that is b). This method guarantees a complexity of O(log(N)) in the number of jumps.

Locality

As we already said when we have more choices for a NodeID in a specific entry of the routing table, Pastry always selects the one that is physically closer to the present node. In order to do that we use a **proximity metric** that is a scalar value that reflect the distance between any pair of nodes, an example could be the **round trip time**. The locality is ensured by choosing a function that allows each Pastry node to determine the distance between itself and a node with a given IP address.

Short Routes Property

Given the locality assumption we can assume that in Pastry route mechanism each node in the routing table is chosen to refer to the nearest node, according to the proximity metric, with the appropriate NodeID prefix. As a result, in each step a message is routed to the nearest node with a longer prefix match. Simulations show that the average distance traveled by a message is between 1.59 and 2.2 time the distance between the source and the destination in the underlying Internet.

Route Convergence Property

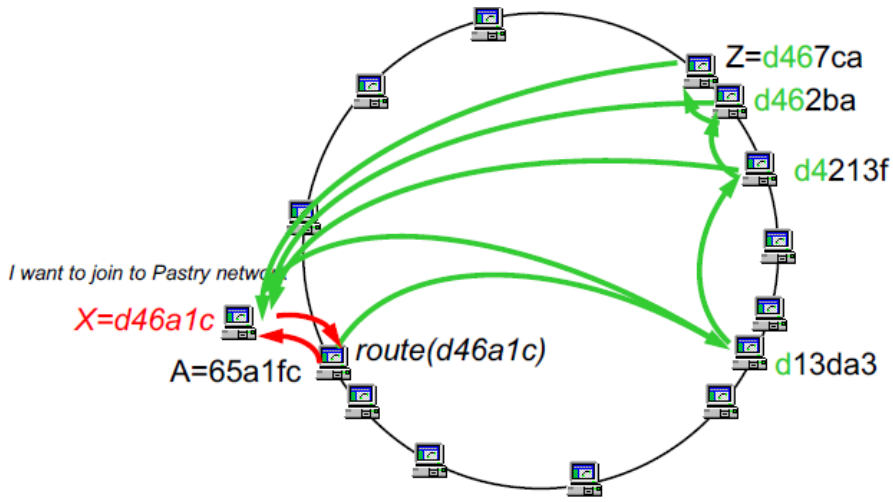
This property compares the distance that two messages (lookup messages), that refers to the same key, travel before their routes converge, we also assume that the two messages have different injection point in the network. Simulations show that, given our network topology, the average distance traveled by each of the two messages is approximately equal to the distance between their respective source nodes.

Node Arrival

Now we have to deal with a new node that wants to enter the Pastry network. Let us suppose that a new node arrives and, since the underlying network is the Internet, its IP address is known, because of that we can use the hash function to retrieve the NodeID of the new node. We have to put the node in the portion of the ring where other nodes exist and have NodeID that are numerical closer the new one. Since the problem is equal to the lookup for a key in the network, we can use the same routing schema. The only

difference is that the routing table of the new node must be initialized and we have to inform the adjacent nodes that the node is inserted, in particular we follow the next steps:

1. We contact a **known** nearby Pastry node *A* according to the proximity metric. We then ask to route a "join" message using X as a key, where X is the NodeID of the new node that wants to join the network.
2. Since we are using the same routing schema, the message is routed to the existing node Z with the NodeID numerically closest to X.
3. The new node X obtains the leaf set from Z, we initialize the i-th row of the routing table of X is taken from the i-th node encountered along the route from A to Z. (in pratica copio la i-esima riga della routing table dell'i-esimo nodo incontrato, nell'i-esima riga della routing table del nuovo nodo)
4. X notifies the nodes to update their states.



Node Departure or Fail

Neighboring nodes in the NodeID space (=leaf-set) periodically exchange keep-alive messages, they can do that because they are in each other's leaf set. We assume that a node is failed if it is unresponsive for a period T. as a consequence, all members of the failed node's leaf set are notified and they update their leaf sets. Since the leaf sets of nodes with adjacent NodeIDs overlap, this update is trivial. A recovering node (node rejoin) contacts the nodes in its last known leaf set, obtains their current leaf set and then notifies the members of its new leaf set of its presence. Routing table entries that refers to failed nodes are repaired lazily.

Pastry API

nodeId = pastryInit(Credentials)

a. causes the local node to join an existing Pastry network (or start a new one) and initialize all relevant state

b. returns the local node's nodeID

The credentials are provided by the application and contain information needed to authenticate the local node and to securely join the Pastry network

route(msg,key)

- causes Pastry to route the given message to the node with nodeID numerically closest to key, among all live Pastry nodes

send(msg,IP-addr)

- causes Pastry to send the given message to the node with the specified IP address, if that node is alive. The message is received by that node through the deliver method

Applications layered on top of Pastry must export the following operations:

deliver(msg,key)

- called by Pastry when a message is received and the local node's nodeID is numerically closest to key among all live nodes, or when a message is received that was transmitted via *send*, using the IP address of the local node

forward(msg,key,nextId)

- called by Pastry just before a message is forwarded to the node with nodeId = nextId. The application may change the contents of the message or the value of nextId. Setting the nextId to NULL will terminate the message at the local node

newLeafs(leafSet)

- called by Pastry whenever there is a change in the leaf set. This provides the application with an opportunity to adjust application-specific invariants based on the leaf set

Scribe

Scribe is a publish-subscribe architecture built on top of Pastry. Any Scribe node can create a *group*, other nodes can then join the group or multicast messages to all members of the group. The dissemination of messages is **best-effort** (will always be attempted without the guarantee of success) and there is no particular delivery order.

Scribe uses a peer-to-peer network of Pastry nodes to manage group creation, group joining and to build a per-group multicast tree used to disseminate the messages multicast in the group. Scribe is fully decentralized since all decisions are based on local information and each node has identical capabilities. Scribe can support simultaneously a large number of groups with a wide range of group sizes.

Scribe API

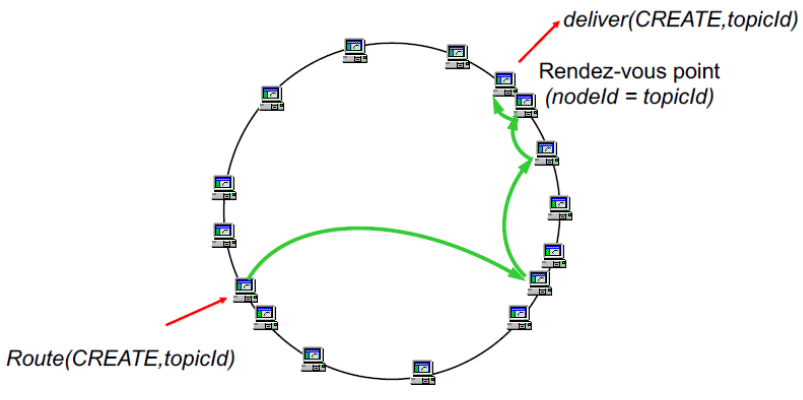
- **create(credentials, groupId)**
  creates a group with groupId. Throughout, the credentials are used for access control
- **join(credentials, groupId, messageHandler)**
  causes the local node to join the group with groupId. All subsequently received multicast messages for that group are passed to the specified message handler
- **leave(credentials, groupId)**
  causes the local node to leave the group with groupId
- **multicast(credentials, groupId, message)**
  causes the message to be multicast within the group with groupId

Scribe P-S Implementation

Each topic, or group, has a unique **topicID** generated from a string using the Pastry hash function. So the topicID belongs to the Pastry keyspace. Now we can define the **Rendezvous** point, it is the Scribe node with the NodeID that is numerically closest to the topicID, this node **is also the root** of the multicast tree created from the group. It is important to mention that the Scribe software on each node provides the *forward* and *deliver* methods that are invoked by Pastry whenever a Scribe message arrives.

As we said the TopicID is obtained by hashing the topic contextual name concatenated with its creator's name. We can use the same hash used in Pastry, this ensures the even distribution of groups across Pastry nodes.

In the image below is shown how a topic is created in the network.

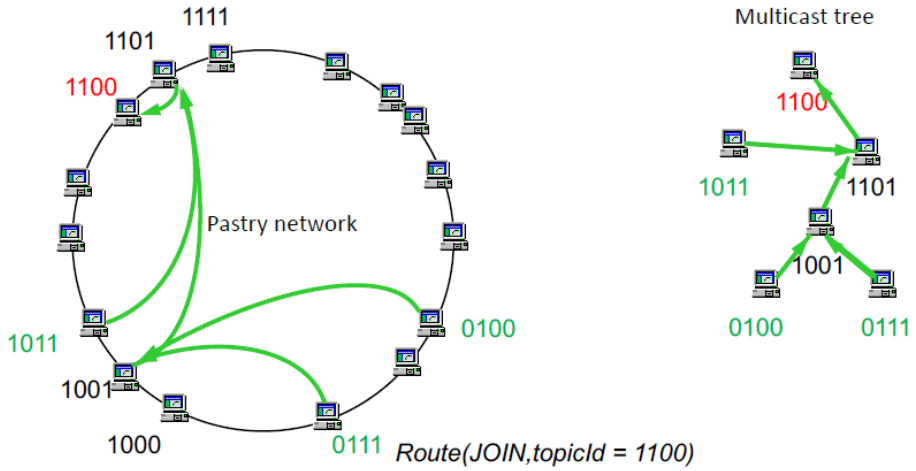


create(credentials,topicId)

The *Route* and *Deliver* methods are part of the Pastry API, whereas the *Create* method is part of the Scribe API. **Deliver is called to receive the message** when the node-id of the current node is the closest one among all live nodes.

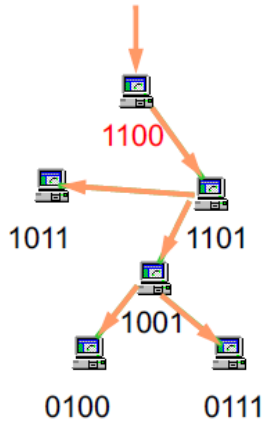
Scribe creates a multicast tree, rooted at the rendezvous point, that is used to disseminate messages. This tree is created using a scheme similar to reverse path forwarding (avoid loops), we can join the Pastry routes from each group member to the rendezvous point in order to create it. The nodes that want to join calls the join function of the scribe api that sends a JOIN message to the topicId node, the message will follow the route according to the pastry route and the node will now be part of the tree.

In the following image is shown how the tree is actually created, on the left side we can see the Pastry layer whereas on the right side we can see the Scribe layer with the actual tree. The leaf nodes of the multicast tree are the actual member of the group.



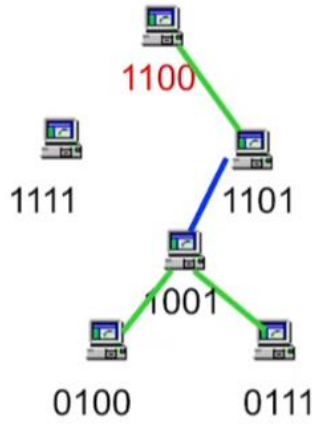
Message Dissemination

When a publisher wants to disseminate a message, it simply routes the message to the rendezvous point through the Pastry network using the TopicID as destination. Once the root has received the message, it knows the destinations and the first hop in the multicast tree. It simply forwards a copy of the message to the first hop and then the messages propagate according to the tree's structure.

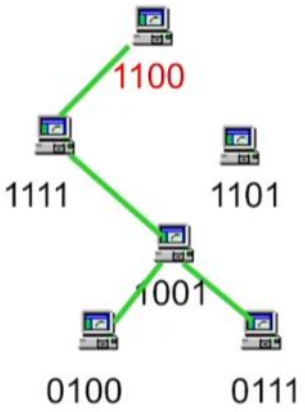


Repairing the Multicast Tree

It may happen that a link between nodes breaks, fortunately the multicast tree is fault tolerant. For example, the broken link is the one in blue in the image.



To solve the problem, the two leaf nodes 0100 and 0111 can send again the *Join* request in order to find an alternative way to the root node. The new tree is shown in the next image.



Using Human as Sensors

The idea of **Social Sensing** is to use human as sensors, this means that using any social platform humans are willing to share information, and this information come from what human can see in the surrounding environment. The idea is to use the social media as a sensor network, but sensors are not devices but are humans. There are some problems in using the information coming from social media, the first that we will analyze is that we have to determine if the information is true or false. We will start from a simple setting in which in the physical world some emergency is going to happen, in this way we are sure that the information are true, although there still is some noise.

Mobile Crowdsensing

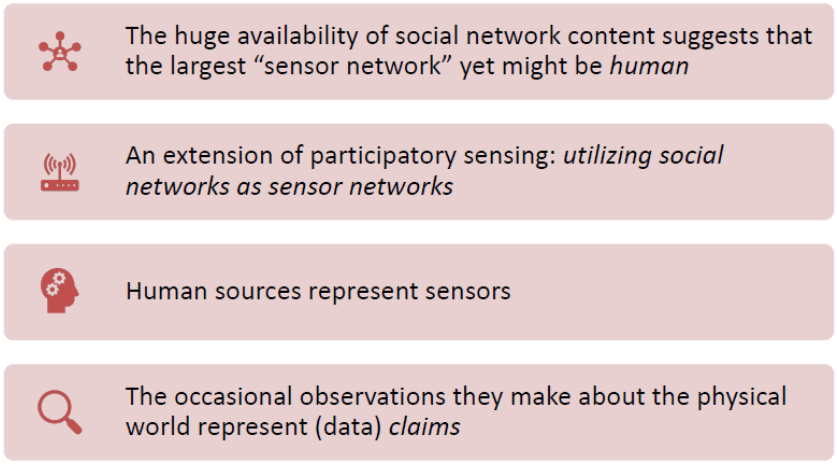
We have **mobile crowdsensing** when a large group of individuals, having mobile devices that are capable of sensing and computing, share and extract information to measure, map, analyze or infer any processes of common interest. In this scenario we do not consider exclusively human sensor, but we consider humans that carry physical sensors. At the beginning, information coming from humans themselves were not considered, nowadays we also want to involve that type of information.

Based on the type of involvement from the users, mobile crowdsensing can be classified into different types:

- **Participatory**: in this case we have users voluntarily participate in sharing information. Sometimes there are some types of incentives.
- **Opportunistic:** In this case we still collect data from the environment, but users are not aware of it.
- **Hybrid:** there are cases in which we can start with an *opportunistic* approach and then we move to a *participatory* one. This may happen when people, because of some type of emergency, start posting for example tweet about the emergency. People that posted the tweet are then contacted in order to gather more information.

Human as Sensors Paradigm (HaS)

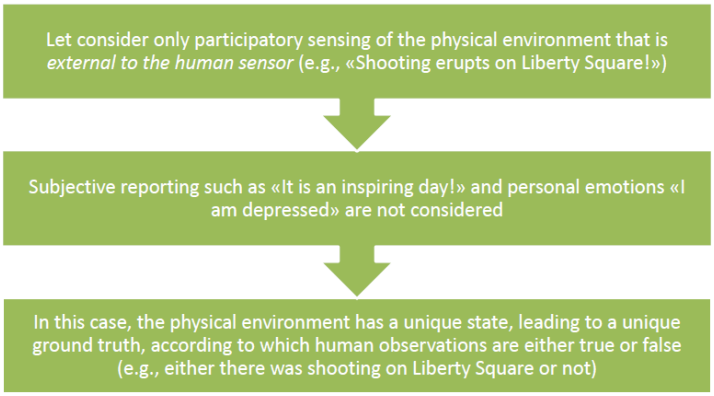
In this case we want to consider information that comes from human and not only from the sensors that humans carry. Human sensors can be considered an extension of participatory sensing, this is because we need to use *social networks* as sensor network, because of that this approach is participatory since users should know that the information that they post can be public or not. In this kind of network humans are sources of sense data, we will only consider situation in which humans talk about representation of physical world around them (eg. Fire or shooting). The posts that they make about the physical world represents **claims** (data).



In the following image are shown some examples in which twitter has been used in the aftermath of some emergency. It is useful since the time that passes form the occurrence of the emergency and the post of the information on the social media is typically small. In case of earthquakes, using typical sensors, we would require 12 to 34 hours in order to classify exactly where the earthquake occurred. We can also use the post on social media to estimate the level of damage of the emergency.

HaS: the Reliable Sensing Problem

The main problem is that when we use the humans as sources the reliability is not guaranteed. If we consider what is happening in the real world as ground truth, when we look at the post from a human we do not know if the state is true or false. The first problem is to distinguish a tweet based on the probability that the human is telling the truth or not. To simplify the problem, we will only consider information from human that can be classified either true or false, so we will ignore all other subjective observations.



As we said, observations of the physical world may be true or false and, because of that, they can be viewed as *binary claims*. The sensing problem is now to determine which claims are correct given that:

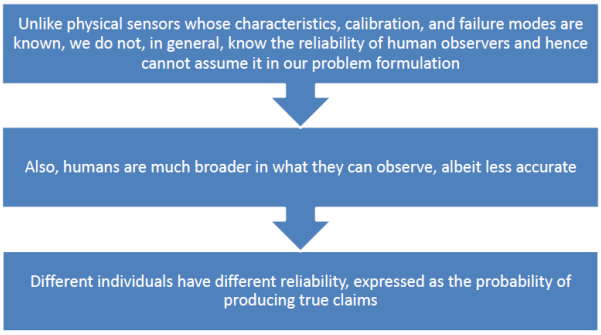
- The reliability of human sensors is generally unknown a priori.
- Uncertain provenance: The provenance of reported observations is uncertain because human can report observation of other humans. This cannot happen with physical sensors. This situation must be considered because we do not have a total independency on observations. We also have to distinguish when an observation is firsthand or not.

A Binary Model of Human Sensing

In this paragraph we will define a model that can be used to treat information from humans. First of all we have to define a binary model of human sensing, when we use observation form humans we know that the sources have unknown reliability and that binary measurement that we collect have uncertain provenance. we can try to formulate the problem of understanding to which extent the post reports what is happening in the external environment (the ground truth) using an estimation-theoretic approach. In other words, we make an observation that is not necessarily true, we can use a distribution of probability in order to assume the probability of the observation being true (failure model).

1: Unknown Reliability

We have already discussed about this point, we can say that with a physical sensor we know exactly its characteristics like the range, the calibration, we also know the failure modes of the sensor. With a human we do not have all this information. Another point is that, for example when we use an accelerometer, we know for sure that it measures the acceleration, with human sensors we do not know if they are willing to provide an exact measure and always the same type of measure. Also, humans are much broader in what they can observe, even though less accurate. Moreover, different individuals have a different level of reliability. **So, the reliability is the probability of producing true claims, and a true claim is capable of reproducing the physical world's state.**



2: Binary Observation

The physical world is a collection of facts that people want to report, in our example facts are restricted to some kind of emergency (e.g. “Main street is flooded”, “The BP gas station on University Ave is out of gas”). Human sensors report some of the facts they observe, because the observation can either be true or false, claims can be thought of as *measurements of different binary variables*, as we know we can associate a probability distribution to a binary variable when we are not sure about the reliability of the source. So, reliable sensing means to infer which among the reported human observations match ground truth in the physical world.

3: Uncertain Provenance

When we analyze an observation, for example a tweet of Bob that states “Main street is flooded”, even if we authenticate Bob as the actual source of the tweet, we still do not know if Bob truly observed that first-hand or heard if from another person. We can state that because it not unusual for a person to report observations they received from others as if they were his/her own. It is important to note that there is not an analogy about this problem when we use a physical sensor like an accelerometer because all the measure can be considered first-hand. From a sensing perspective, this means that error in measurement, that are false claims, across sensors may be *non-independent*. Because of that, when you post a non-fist-hand false observation you mystify the physical world in a non-independent manner. If we want to know if a set of claims are true or not we can for example use the vote approach, in the sense that we can consider a claim true if a large number of users have made that claim. We can easily understand that this approach is not correct since many users can repost a false claim, this would result in a high number of votes even if the claim if false.

Note: The perturbation added by more users through reposting is called “noise”

HaS: a Solution Architecture

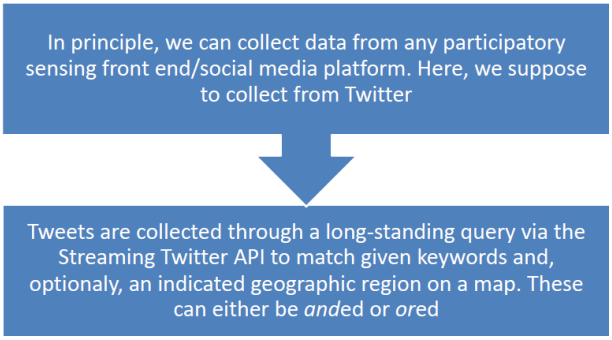
We have to understand which are the parameters that characterize the real phenomenon starting from unreliable information. We can use an architecture made of different software modules that we can implement in order to solve this problem:

- **Collect data** from the “sensor network” so any social media like twitter or facebook.
- **Structure the data for analysis**: the data we can extract from a social media are unstructured and do no follow a standard.
- **Understand how sources are related**: we have to understand how sources of information are related together, we have to understand for example if there is a network that describes relationships between human when they talk about the same real phenomenon, this is because we want to know if they are related or not. This point deal with the problem of *uncertain provenance*.

- Use this unstructured information and make them structured, for example translate them into vector in a given space. So at the end of this phase we have clusters of observations that talk about the same phenomenon
- Finally we can use the likelihood obtained from the probability distribution in order to classify an observation as true or false

1: Data Collection

We can use any platform, we will collect information from twitter using some API and some keywords.



2: Data Structuring

At the beginning of this phase we have a collection of non-structured text that we extracted from twitter. First of all we can use some NLP tools (also process the semantic) to process each tweet, we can use the **tokenize** function that can extract a word or a group of words from a text, by doing that we discard useless word and we obtain a single vector for each tweet in out collection. After this procedure, we can apply a *distance function* to each pair of structured tweet (vectors containing tokens), in this way we can obtain a similarity measure between two tweets, the more dissimilar the larger the distance. We can use the cosine similarity function to measure the similarity.

In the case of data collection from Twitter:

- individual tweets = individual observations
- *A*, *B* vectors of token occurrence in tweets
- let use a *cosine similarity function* to return a measure of similarity based on the number of matching tokens in the two inputs

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

dot product

vector magnitude

At this point we have computed the similarity measures for every couple in our collection, we can use this information to build a graph in which vertices are individual information and the edges represents the similarity among them. The higher is the similarity the bigger will be the weight of the link between two vertices (actually the distance between vertexes is lower with increasing similarity). After that we can cluster the graph so that similar observations will be clustered together, each cluster is called **claim** and it is made of different observations.

After this step we can build another graph since we have claim with sources that contribute to create such claims. This graph will be called **Source-Claim graph**. The graph has the following characteristics:

- Each source S_i, for example a human, is connected to all the claim he made. That are the clusters they contribute to.

- Each claim C_j, that is a cluster of information, is connected to all sources who espoused it. The connections represent all sources of tweets in the corresponding cluster.

We can say that $S_i C_j = 1$ if source S_i makes claim C_j. Each claim is true if it is consistent with ground truth in the physical world, otherwise it is false.

3: Sources Relationship

We have to deal with the problem of uncertain provenance. Looking at the **SC** graph we cannot state whether sources are independent or not, we simply see that a given source contributed to a given claim. We need another graph that is called **social information dissemination graph SD**, this graph can estimate how information might propagate from one person to another.

The way in which we build that graph strongly depends on the modality in which users interact in the social media, for example if we consider twitter we can post first-hand observation or we can retweet a statement made by another user.

In the following image are shown some methods that we can use to create as SD graph.

1. **Epidemic Cascade network (*EC*)**: each distinct observation is modeled as a cascade and the time of contagion of a source describes when the source mentioned this observation
2. **Follower-Followee network (*FF*)**: a directed link (*Si*,*Sk*) exists in the social graph from source *Si* to source *Sk,* if *Sk* is a follower of *Si*
3. **Re-Tweeting network (*RT*)**: a directed link (*Si*,*Sk*) exists in the social graph if source *Sk* retweets some tweets from source *Si*
4. **Combined network (*RT+FF*)**: a directed link (*Si*,*Sk*) exists when either *Sk* follows *Si* or *Sk* retweets what *Si* said

4: The Estimation Problem

At this point we have to estimate the credibility of claims because we do not know a priori whether claims are correct or not. Even though, we do not have a ground truth, we can build one, for example if observations are talking about a hurricane it is likely that the newspapers will talk about it, so we can use those news as a ground truth. Another important aspect is that a claim can be either true or false, this reduces the problem to a problem with Boolean variables and we have to deal with the probability that a variable is true or false.

With inputs computed, the next stage is to perform the analysis that estimates correctness of claims

For each claim, *Cj,* we need to determine if it is true or false

Tweets are analysed using a sliding window: Let the total number of claims computed from tweets received in the last window be *N*

Voting

The first way that we can use to analyze a claim is **voting** an this is the simplest approach. For each claim we have to count all the sources that have contributed to that claim. We can label a claim as *true* if it has an high counter.

This approach is **suboptimal** since it does not consider that:

- Sources have different degrees of reliability, consequently their votes should not have the same weight.
- Sources may not be independent. In this scenario if a source repeats what it heard from another source, its vote does not add anything to the credibility of the claim.

Likelihood Estimation

We can try to use classical probabilistic techniques to estimate if a claim is true or not. We observe an unknown phenomenon and the way the phenomenon manifest itself is given by a number of different observations that we have to put together in order to have some truth to emerge. In other words, we have to compute the probability of correctness of claims taking into account *unknown source reliability* and *uncertain provenance*. We have to compute the likelihood of a give claim C_j is true given graphs SC and SD, formally we have to compute $P(C_j|SC, SD)$ for each claim.

Maximum Likelihood Estimation

This is an optimization approach, we have to add some definitions. Let m be the total number of sources (for example sensors), let us describe each source $S_i, i \leq i \leq m$ by two parameters, bot unknown in advance:

- The odds of **true positive**: $a_i = P(S_i C_j = 1 | C_j = 1)$ this express the reliability of a given source.
- The odds of **false positive**: $b_i = P(S_i C_j = 1 | C_j = 0)$

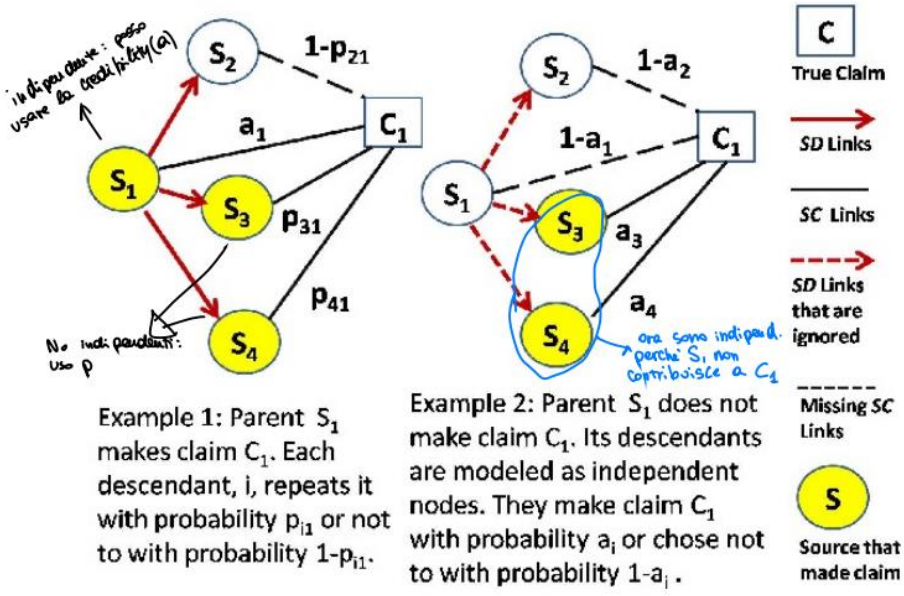
Let d be the **unknown expected ration of correct claim** in the system $d = P(C_j = 1)$. Let $\Theta = [a_1 \dots a_m b_1 \dots b_m d]$ be the vector of the unknown variable that we have just defined. Since we do not know the value of all these variable we want to build a Theta vector that better allows us to construct the SC graph that is the only evidence that we have on the phenomenon. Generally we use iterative algorithms that start from a certain value for Theta and give us a converged Theta that is the solution to our problem.

Formally we can describe the **ML Estimator** as follows: we have to find the value of the unknow vector Theta that maximize the probability of observation of SC given the social network SD. $P(SC|SD, \Theta)$.

We can assume that the claims are independent, so we can rewrite the probability. $P(SC|SD,\Theta) = \sum_Z P(SC,z|SD,\Theta)$ where Z is a vector where $z_i = 1\ if\ C_i$ is true. Z is a Boolean vector that gives us information about the truthiness of each claim, obviously this vector is **not known** in advance. If we are able to obtain Z and Θ we reach our objective because we have the reliability of the sources given by Θ and we also know which claims are true and with claims are false.

Now we have to analyze how we can put the relationship information among sources into the MLE problem. The intuition is to use the **Social Dissemination graph**, in this graph each node represents a source and each edge represents a relationship. We can consider subsection of this graph that represents a family, so our section is a two-level tree with parents and children. Having these relations means that the information shared from the child are secondhand information, it is also important to notice that it is not always true that a child reposts an information heard from a parent. Now we have to consider the probability that a child retweet an information heard from a parent, we call this probability p_{ik} where i is the child and k is the parent. The probability is computed observing the SC graph and calculated as the division of the number of times S_i and S_k make the same claim and the number of claims S_k makes.

The general intuition is that in the SD graph if a parent does not make the claim, then the children act as if they are independent sources, otherwise each child repeats the claim with a given p_{ik} probability.



Example 1: Parent S_1 makes claim C_1. Each descendant, i, repeats it with probability p_{i1} or not to with probability $1-p_{i1}$.

Example 2: Parent S_1 does not make claim C_1. Its descendants are modeled as independent nodes. They make claim C_1 with probability a_i or chose not to with probability $1-a_i$.

In the previous image we have isolated the SC graph for Claim 1 and then we have overlapped the SD graph to obtain the relationship between sources that have contributed to C_1. We assume that C_1 is a true claim. On the left side we can see that S_1 contributed to claim 1 so S_3 and S_4 have to use the *repeat probability* since they are not independent. On the right side, since S_1 did not contributed to Claim 1, S_3 and S_4 are independent and we can use a to compute the probability.

In the next image is shown the iterative algorithm used to compute the vector Theta.

Algorithm 1 Expectation Maximization Algorithm

1: Initialize θ with random values between 0 and 1
2: Estimate the dependent ratio (i.e., p_{ig}) from source dissemination graph SD based on Equation (7)
3: **while** $\theta^{(n)}$ does not converge **do**
4: **for** $j = 1 : N$ **do**
5: compute $Z(n, j)$ based on Equation (11)
6: **end for**
7: $\theta^{(n+1)} = \theta^{(n)}$
8: **for** $i = 1 : M$ **do**
9: compute $a_1^{(n+1)}, ..., a_m^{(n+1)}, b_1^{(n+1)}, ..., b_m^{(n+1)}, d^{(n+1)}$ based on Equation (12)
10: update $a_1^{(n)}, ..., a_m^{(n)}, b_1^{(n)}, ..., b_m^{(n)}, d^{(n)}$ with $a_1^{(n+1)}, ..., a_m^{(n+1)}, b_1^{(n+1)}, ..., b_m^{(n+1)}, d^{(n+1)}$ in $\theta^{(n+1)}$
11: **end for**
12: $n = n + 1$
13: **end while**
14: Let $Z_j^c =$ converged value of $Z(n, j)$
15: Let $a_i^c =$ converged value of a_i^n; $b_i^c =$ converged value of b_i^n; $d^c =$ converged value of $d^{(n)}$
16: **for** $j = 1 : N$ **do**
17: **if** $Z_j^c \geq 0.5$ **then**
18: C_j^* is true
19: **else**
20: C_j^* is false
21: **end if**
22: **end for**
23: Return the claim classification results.

- The input is the source claim graph *SC* from social sensing data and the source dissemination graph *SD* estimated from social network
- The output is the maximum likelihood estimation of source reliability and claim correctness

At line 17 we have to set a threshold to determine if a claim is true or not. This approach is very good because this allows us to infer the credibility of an observation using a huge number of observation.

Evaluation

We have to use the *Baseline* approach. We start with the **voting** as baseline for our evaluation so we have a claim we count the number of vote attributed to that claim and we decide with a threshold if that claim is true or not. Data credibility is estimated by the number of time the same tweet is collected from the human network, the larger the repetition, the more credibility is added to the content. As we said before, the main weakness of voting is that it counts both tweets and retweets, so it cannot distinguish between original information and dependent information. To improve the voting we can exclude the retweets in order to have a better estimation of the credibility.

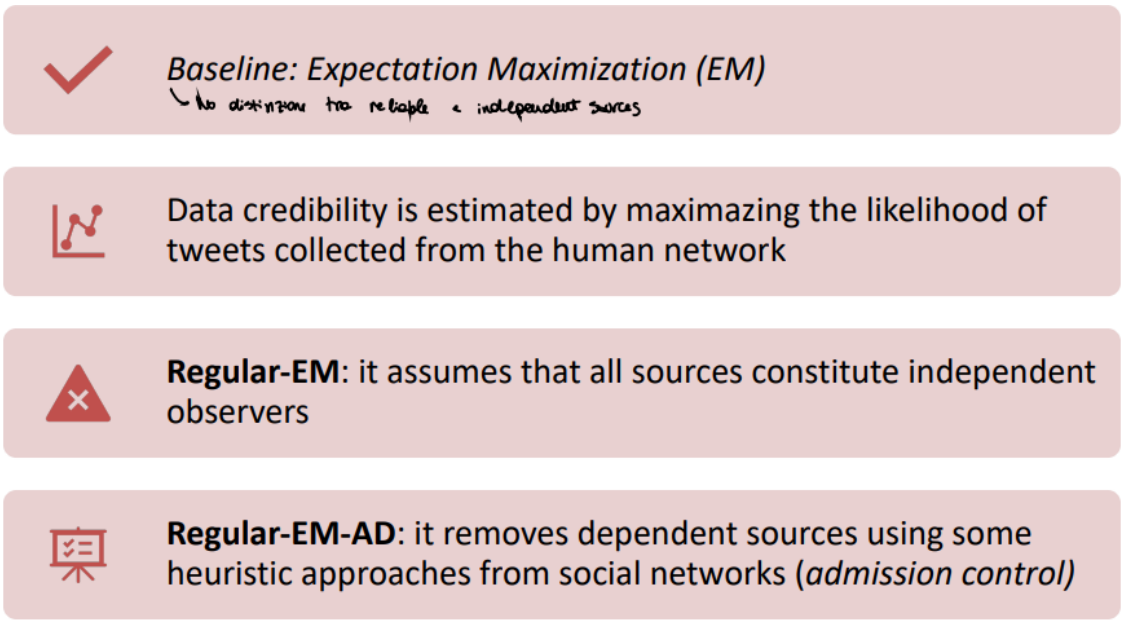
Baseline: Voting → conta tweet e retweet, non distingue gli originali

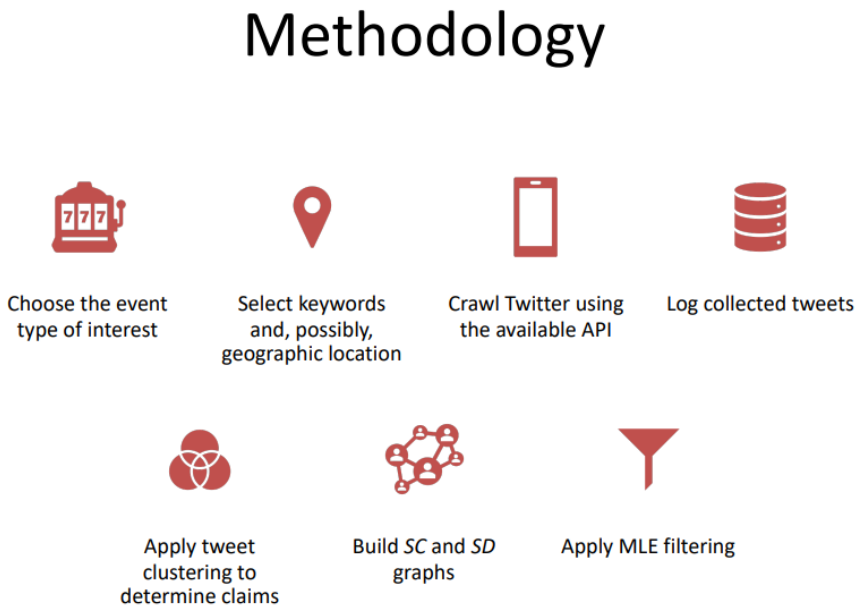
Data credibility is estimated by the number of times the same tweet is collected from the human network. The larger the repetition, the more credibility is attributed to the content

Voting: it counts both retweets and original tweets as full votes

Voting-NoRT: it only counts the original tweets
↳ Migliorameto

Another baseline is the standard **Expectation Maximization (EM)**. This baseline is the same algorithm saw before without considering the Social-Dissemination graph, so we are not able to distinguish between reliable sources and independent sources.





At the end we apply a MLE filter to eliminate the false claim, so in the output of the process we only have claims that are likely to be true.

Validation Method

To validate a method, we need a ground truth, we have to consider the collected dataset and *manually* label each claim as true or false. Since this process cannot be automated, we have to limit the number of claims that we need to label. We can decide to label a claim as true for example looking at the news in the media. There can be some *Unconfirmed claims*, we can take the responsibility of assigning a label or we can simply drop those claims. If we decide to drop unconfirmed claims, we have a reduced of true claim in the ground truth dataset; in other words we are following a **pessimistic approach**. If the method behaves good

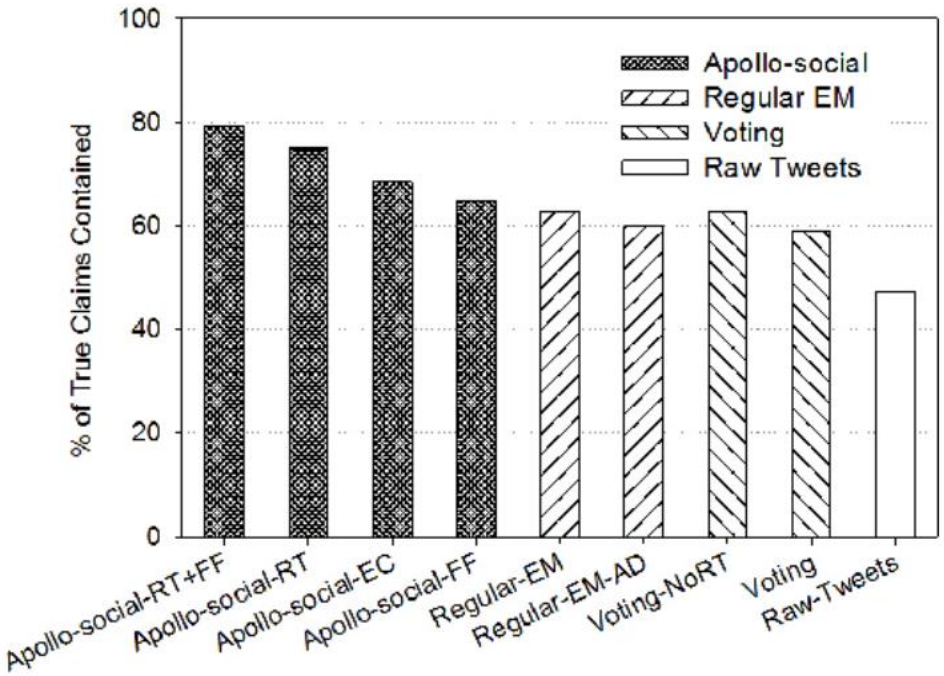
in this pessimistic approach it likely to work better in real word scenarios where some unconfirmed claim are true.

The output of filtering was manually graded to determine match with ground truth. Due to man-power limitations, manually grading only the 150 top ranked claims using the following rubric:

True claims: Claims that describe a physical or social event that is generally observable by multiple independent observers and corroborated by sources external to the experiment (e.g., mainstream news media)

Unconfirmed claims: Claims that do not meet the definition of true claims. May include true claims that cannot be verified

In the next image we can see the precision of the methods we have discussed so far. We can compute the percentages of true claims, to do that we use the ground truth computed as already said.



As we can see Apollo with the retweeting network and the follower-followee network reaches the 80% of true claim in a pessimistic approach, this means that in a real world scenario the method should work even better. In the next images are shown some limitation of Apollo.

The scheme tends to reduce the number of introspective (e.g., emotional and opinion) tweets, compared to tweets that presented descriptions of an external world

The reason may be that emotions and slogans tend to be retweeted and hence, tend to be suppressed by the scheme

In contrast, external facts are often observed independently by multiple sources, and hence not suppressed

Distortions and biases of human sensors are quite persistent in terms of direction and amplitude, unlike white noise distortions. In a sense, *humans exaggerate information in predictable ways*

The FF, RT and RT+FF schemes can be improved by building information propagation models that account for:

topic (e.g., sensational, emotional, and other news might propagate along different models)	expertise	relationship reciprocity	mutual trust	personal bias (e.g., the claim that «police is beating up innocent demonstrators», versus «demonstrators are attacking the police»)