Stabilizing Voronoi Diagrams for Sensor Networks with Hidden Links

Jorge A. Cobb

The University of Texas at Dallas

cobb@utdallas.edu

I am currently an Associate Professor in the Department of Computer Science at the University of Texas at Dallas. I received the B.S. in Computer Science (with highest honors) from The University of Texas at El Paso. I also received an M.A. in Computer Science and a Ph.D. in Computer Science from The University of Texas at Austin. While in the Ph.D. program at The University of Texas at Austin, I was awarded the AT&T Ph.D. Scholarship. I have been a member of many technical program committees of international conferences in computer networking.

My current research interests include:

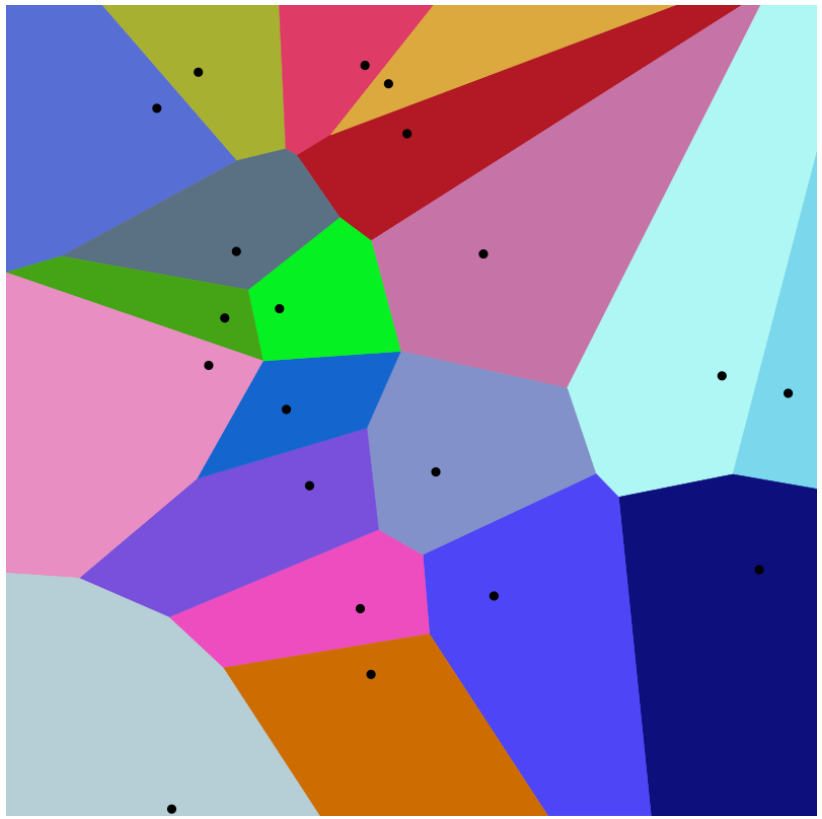
- *Sensor Networks*
- *SDN in Interdomain Routing (eBGP and iBGP)*
- *Stabilizing Systems*
- *QoS Scheduling of real-time traffic*

Dr. Jorge A. Cobb
Department of Computer Science
The University of Texas at Dallas
cobb@utdallas.edu

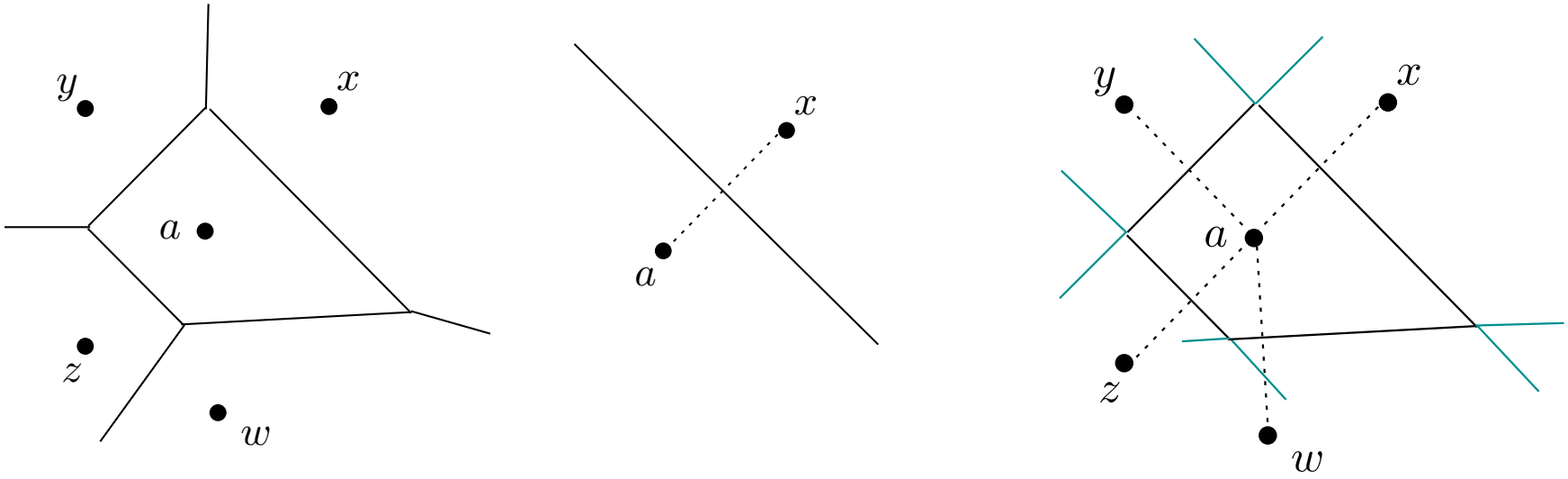
Background

Voronoi Diagrams

- A *Voronoi diagram* (VD) in a two-dimensional plane consists of:
  - a set of *generator points*
    $P = p_1, p_2, \ldots, p_n$
  - a set of regions
    $R = R_1, R_2, \ldots, R_n$.
  - where R_i consists of all points on the plane closer to p_i than to any other generator point.

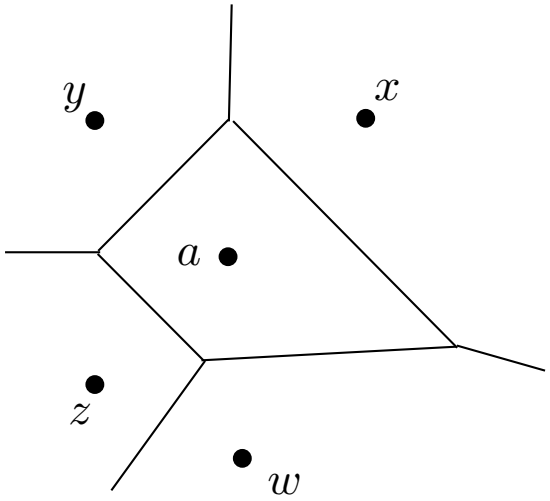


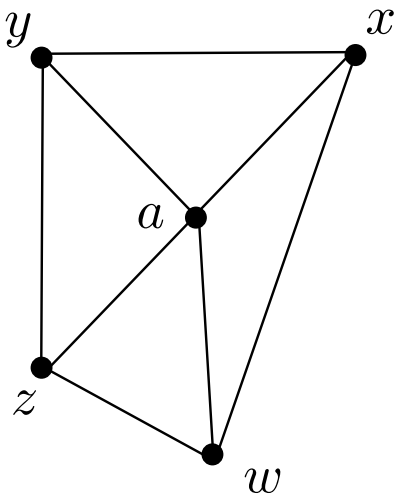
Region Boundaries

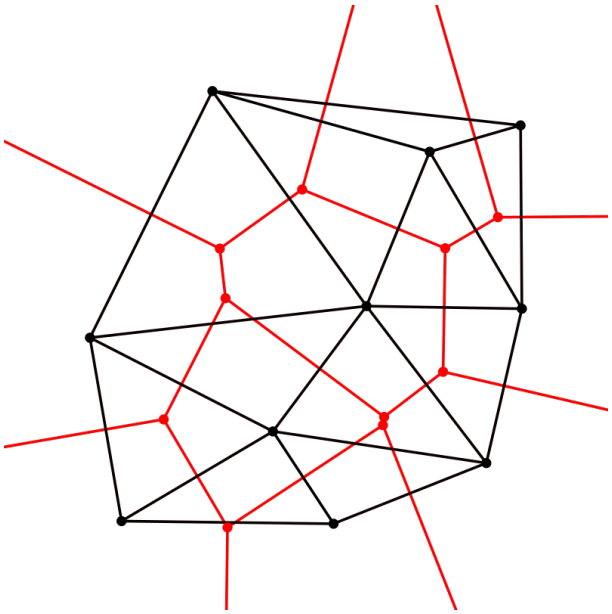


- R_a = convex hull obtained from the intersection of all the bisectors with all other generator points.

Delanuay Triangulation







- A Delanuay Triangulation (DT) is the dual of the VD.
  - It has the same information in different form.
- There is an edge between a pair of generator points p_i and p_j iff R_i and R_j share a face in the VD.
- Note that the DT forms a convex-hull of all the points.

Greedy Routing on DT

- Greedy routing (a.k.a. geographic routing):
  - For a destination *d*, a node *u* selects as the next hop a neighbor that minimizes the physical distance from *u* to *d*.
  - The routing state needed per node is independent of network size.
  - This is attractive for the resource-starved sensor networks, especially for large networks.
- Problem: greedy routing on an arbitrary graph may become trapped at a local minimum.
  - However, on a DT, greedy routing is guaranteed to reach the destination.
  - DT's are well suited for greedy routing.

Objective

- Obtain a distributed and self-stabilizing algorithm for each node to determine its neighbors in the DT.
- We want the system to be *stabilizing*
  - i.e., it restores itself to a normal operating state from *any arbitrary initial state* of its variables.
  - Stabilizing systems recover automatically from a wide variety of transient faults
- No flooding, and thus low overhead
  - Use a “distance-vector” type of approach rather than a “link-state” type of approach.
  - Communicate “only” with neighbors in the DT.

Related Work

- Distributed protocols for DT's exist in the literature,
  - however, they are not fault tolerant, and
  - they assume an underlying routing protocol.
  - Works designed for wireless greedy routing have limited fault-tolerance (not stabilizing).
  - A distributed and stabilizing solution for DT's exists, but it assumes an underlying routing protocol and is thus not suitable for greedy routing.

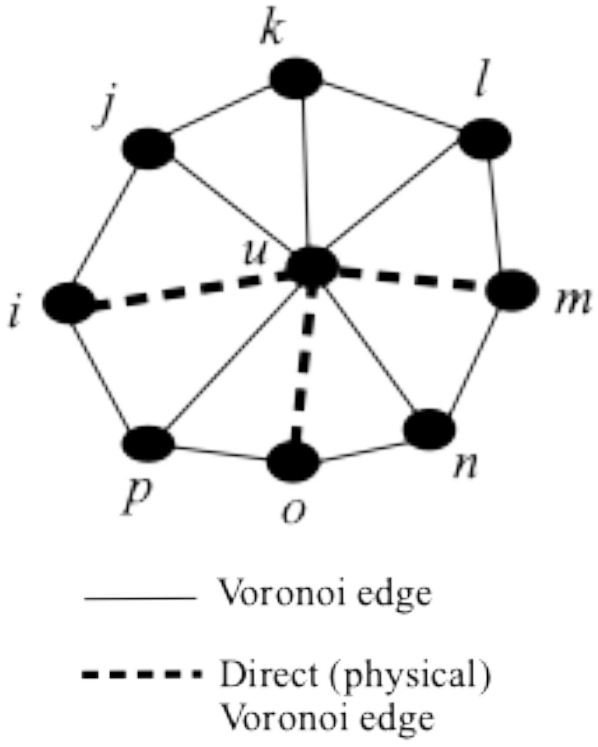
Our Earlier Work

- In ICSNC 2019, we presented a protocol for computing the DT that is distributed and stabilizing
  - However, it assumes the unit-disc transmission model: if two nodes are within a distance *r* from each other, they can communicate directly
    - I.e., there are no obstacles
- In this paper, we relax this assumption
  - We go beyond obstacles, we simply assume an arbitrary connected graph
  - If two nodes are directly connected is independent of their distance.

Network Model and Connectivity

Network Model

- Nodes u and v have a *physical link* <*u,v*> if they can directly exchange messages.
  - L_{phys}(u) = physical neighbors of u
  - Physical links form a connected network.
- $V(u)$ = set of neighbors of u in the DT
  - I.e., $V(u)$ = Voronoi neighbors of u
  - Edge (u, v) is a Voronoi edge if v in $V(u)$
- $L_{phys}(u) \cap V(u)$ are the physical Voronoi neighbors of u.
  - Also referred to as *core*(u)



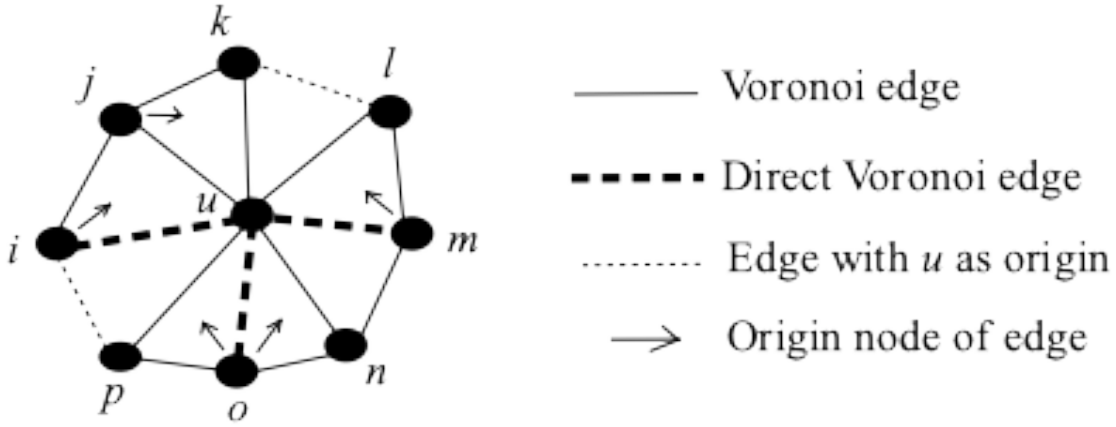
- $V(x) = \{i, j, k, l, m, n, o, p\}$
- $L_{phys}(u) \cap V(u) = \{l, m, o\}$
- u may have many other physical links other than $\{l, m, o\}$.

Arbitrary Physical Links

- With the unit-disc model used earlier, there is always a path of Voronoi edges between every pair of nodes [ICSNC 2019]
  - Thus, a node u only needed to exchange messages with its neighbors in $core(u) = L_{phys}(u) \cap V(u)$
- This is no longer the case with obstacles, i.e., with arbitrary physical links
  - A node potentially can have *none* of its Voronoi neighbors in $core(u)$

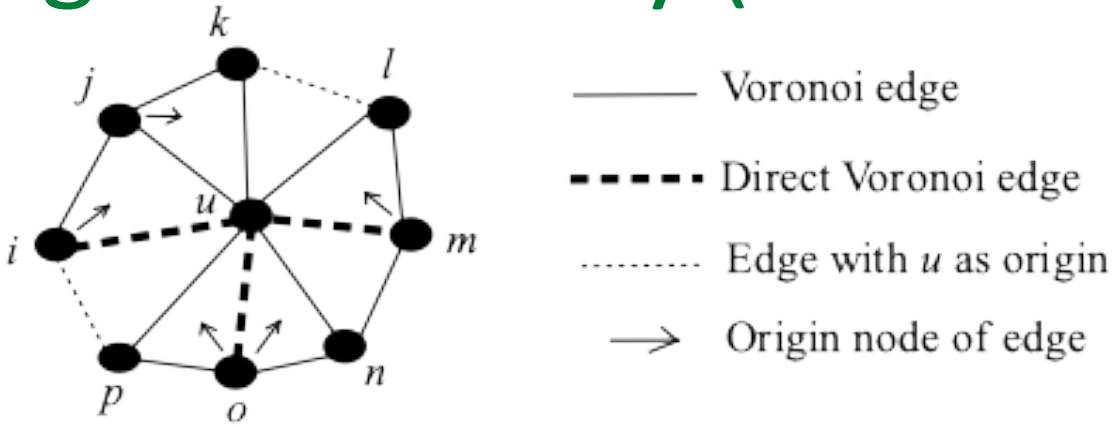
Region Construction

Region Anatomy



- *u* is aware of nodes in *core*(*u*) since it has a physical link to them.
- Consider *n:*
  - *o*, *m*, or both, can communicate with *n* (recall the network is connected).
  - It (or both) informs *u* about *n*.
  - *u* remembers who informed it of edge (*u,n*).
  - We refer to this node as the edge's *origin*.
- Each node keeps track of the number of transmission hops to cross a Voronoi edge, known as the *label*.
  - E.g., if the origin of (*u,n*) is *m*, then *label*(*u,n*) = *label*(*m,n*)+1.
  - Both *o* and *m* report to *u* the expected number of physical hops to cross edge (*u,n*) through them.
  - *u* chooses as origin the one with the least number of hops (i.e., label).

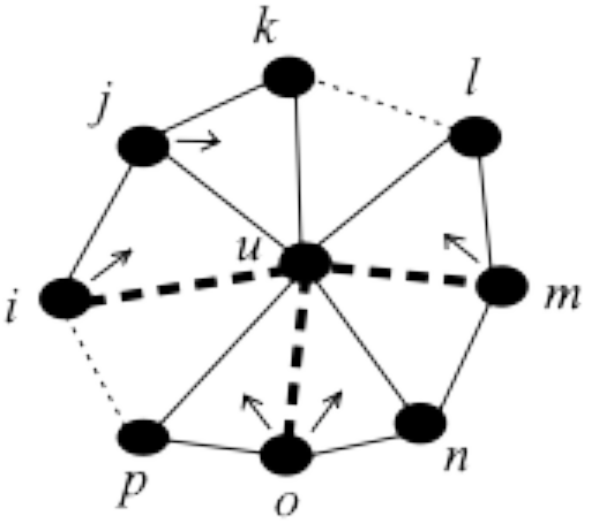
Region Anatomy (continued)



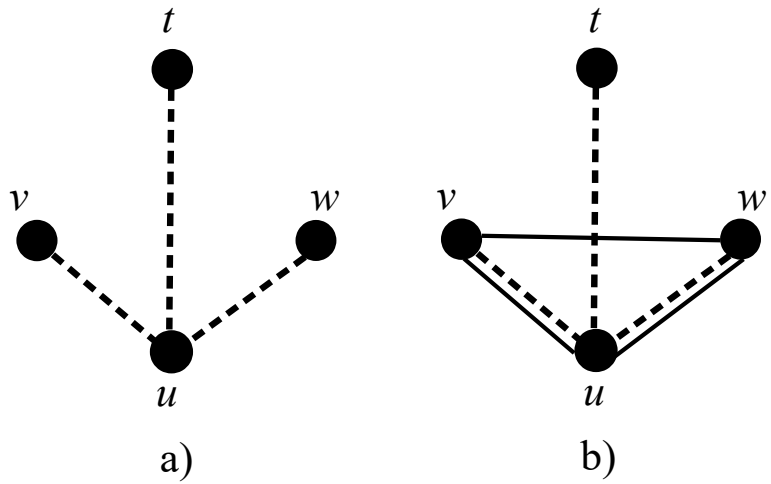
- Consider nodes *i*, *j*, *k* in between core nodes *i* and *m*.
  - *i* is the origin of (*u,j*) and *j* is the origin of (*u,k*).
- A *segment* is the sequence of nodes starting at a core node where each node is the origin of the previous edge.
  - The clockwise segment starting at *i* is (*i,j,k*).
  - The counter-clockwise segment starting at *m* is (*m,l*).
  - The counter-clockwise segment starting at *i* is simply *i* itself.
- Consider nodes *k* and *l,* they are not aware of each other
  - In this case, *u* must introduce them to each other, and
  - *u* becomes the origin of edge (*k,l*).
  - Similarly, it introduces *i* and *p* to each other, and becomes the origin of edge (*i,p*).

Forwarding of Control Messages

- In [ICSNC2019], we used the following strategy.
  - A single message type, *edge*, informs a node that it has a Voronoi neighbor. E.g., *u* must inform *k* and *l* of each other via an *edge* message.
  - If the destination is *k*, the *direction* field is set to *clockwise*, and the message is given to core node *i*.
  - The message then traverses *the entire segment: i*, *j*, and *k*.
- The whole segment is traversed due to how nodes forward a message not addressed to them: it is forwarded to the adjacent core node.
  - E.g., if *u* receives an *edge* message from *m,* and the destination is not *u,* and the direction is *clockwise*, *u* forwards the message to *i*, and if the direction is *counterclockwise*, it forwards the message to *o*.



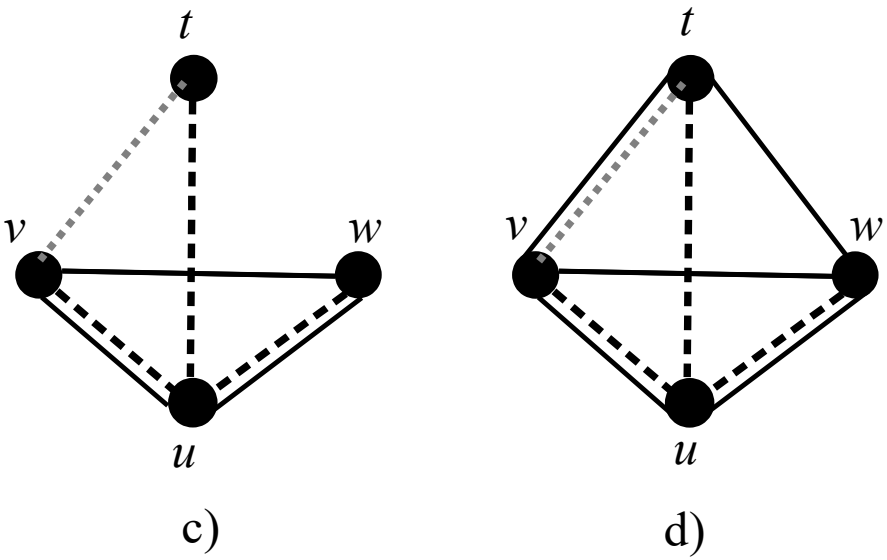
Obstacle Pitfalls



- Physical links are shown as dashed lines.
- Without obstacles, all nodes have a physical link with each other.
  - Due to obstacles, physical links <t,v>, <t,w>, and <v,w> are not present.
- In (b), the (incorrect) computed Voronoi edges are shown as solid lines.
- *u* is aware of *v* and *w*, and thus considers them part of its region
  - *u* sends an *edge* message to each of them, making them aware of each other.
  - However, since *t* is not in the region of *u*, *u* ignores *t*.
  - Thus, node *t* is not aware of *v* and w, and it cannot compute its own region.
- We address a mechanism to correct this next.

Extended Links

- *t* must become aware that, if not for the obstacles, it would have a physical link with *v* and with *w*.
- Thus, node *u extends* the link <*u, t*> to *v* and *w* (see (c)). I.e., it makes *v* and *w* aware that they should have a link with *t*.
- Node *u* sees that its physical link <*u*, *t*> intersects its Voronoi edge (*v, w*).
  - *u* notifies *v* and *t* that they have an *extended link* between them.
  - This link is shown as a gray dotted line
  - *v* considers as an ordinary physical link, and adds it to its set *L* of links.
  - The extended link <*t,v*> is treated in the algorithm like any other link, e.g., it could be part of *core(t)* and *core(v)*.
- Being aware of *t, v* can complete the triangle *t, v, w* (see (d)).



c) d)

- *u* does not need to extend the link <*u*,t> to *w* (although possible, but not necessary). *u* extends to the closest of the two.
- Extended links themselves can be extended.
  - See the paper for details.

Message Types and Routing

Edges and Links

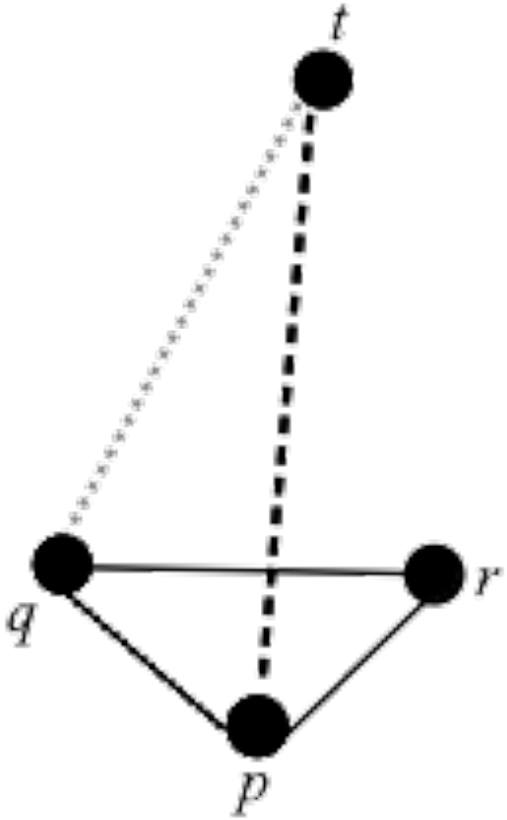
- A node can be notified of two types of neighbors:
  - a Voronoi neighbor
  - a neighbor across an extended physical link
- We thus have two messages two distinguish between these two cases:
  - *edge* message
  - *link* message

Routing across extended links

- Our objective is to modify our method from [ICSNC2019] the least possible.
  - There should be no ``difference'' in the way a physical link and an extended link is handled for routing *edge* and *link* messages.
- Extended links however are not real links
    - Messages must be *tunneled across them*
- We introduce to messages for tunneling
  - *throw*
  - *catch*
- These messages encapsulate other messages that must be sent across the extended link.

continued . . .

- Assume extended link <*q,t*> was created by *p* due to its physical link <*p,t*>
- If *t* wants to send a message to *q* along the extended link, *t* encapsulates it into a *throw* message that is sent to *p*.
  - Note that *t* (and *q*) must remember that the source of this extended link is *p.*
- *p* decapsulates the original message, and encapsulates it into a *catch* message, which is sent to *q*.
- *q* could be many hops away, so the catch message is routed like a normal message using the method in [ICSNC2019]
- Note that *edge* (*q,p*) might be an extended edge itself, and thus, *catch* and *throw* must be able to encapsulate *any* of the other message types.
  - Thus, sending messages requires a recursive procedure
  - See the paper for details.



Protocol notation and pseudocode are in the paper

Stabilization

- A *predicate* P of a network is a boolean expression over the variables in all nodes of the network.
- A network is called P-*stabilizing* iff every computation has a suffix where P is true at every state of the suffix.
- Stabilization is a strong form of fault-tolerance.
  - Normal behavior of the system is defined by predicate P.
  - If a fault causes the system to a reach an abnormal state, i.e., a state where P is false, then the system will converge to a normal state where P is true,
  - It will remain in the set of normal states as long as the execution remains fault-free.

Stabilization Tools

- The region of a node u is obtained from the convex-hull of the nodes known by u
  - Knowledge of a superset is not harmful
  - Nodes not in the region are discarded from the data structures.
- Sanity checks are performed on
  - the local data structures of each node
  - the messages received from each neighbor.

Stabilization Tools (continued ...)

- Problem: messages and data structures may contain nodes that do not exist in the network.
  - This information can propagate and prevent stabilization.
  - We assume there is a limit to the labels of edges (network diameter)
  - We add a hop count to messages when they are generated
    - This hop-count is decreased each time the message is forwarded.
- Once non-existent nodes are removed, the region of each node is slowly built, and is shown by induction on the labels of each edge.

The proof of the stabilization of the protocol is found in the paper

Concluding Remarks

- There is no assumption of having an underlying routing protocol, only knowledge of the physical links is assumed.
- Thus, the protocol can be used as the foundation for a geographical routing protocol.
- If nodes are distributed in the plane according to a Poisson process with constant intensity, then each node in the DT has on average six neighbors.
- Thus, in general, the overhead should be O(N), unless the topology is unusual.
- We will perform simulations in future work to determine the average overhead of the protocol.