Design and Analysis of Distributed Algorithms

Notes from the Nicola Santoro's book

Version: 0.0.3
Last update: 27/02/2018

Distributed Computing Environments

- **Definitions**: entities, messages, communication, actions, behavior
- **Entity**: cpu, memory, alarm
- **External events**: spontaneous impulse, alarm, message
- **Action**: finite and indivisible seq of operations (transaction).
  - **Entity behaviour**: Status $\times$ Event $\rightarrow$ Action; **homogeneous** behavior.
- **Axioms**: finite communication delays, local orientation (in/out neighbor distinguished)
- **Port numbers**: $\lambda_x(x,y)$, x port from x to y; a port is a function
- **Restrictions**: communication (BL), reliability (TR), topological (CN), timing
  - communication: queuing policy, reciprocal communication, bidirectional
  - reliability: edge/entity fault detection
  - topological: tree, ring, connected, complete
  - timing: unary communication delay, bounded delays, sync clocks
  - **Standard restrictions**: STDR = { BL, TR, CN }
- **Costs**: message (M) and time (T) (should use ideal exec delay restriction)
  - Entity workload (M/V) and transmission workload (M/E)
- Problem **complexity**: protocol agnostic message (*M*) and time (*T*) lower bounds.
- **Problem**: what entities must accomplish. P = < P_{INIT}, P_{FINAL}, R >
  - P_{INIT}/P_{FINAL}: initial/final conditions (predicates); R: set of restrictions
- **Solution protocol**. Rules set to solve a problem.
- Small systems execution view using **TED** (Time x Event diagrams).
- **State**: registers, alarms, pending messages
  - *Future(t)* = pending events in the system at time t. *Future(0)* = spontaneous impulses.
  - $State_X(t)$ = entity X registers and alarms at time t
  - *State(t)* = system state. $\forall$ X, Σ_X $State_X(t)$ + *Future(t)*
  - State values: S_{INIT}, S_{TERM}, S_{START}, S_{FINAL}
    - S_{TERM}: once reached cannot be changed by the protocol; actions are still possible
    - S_{FINAL}: the sole action is *nil*
- **Termination.** *Termination(t)*: ($\forall$ X, $State_X(t) \in S_{TERM}$) and (*Future(t)* = $\emptyset$)
  - ExplicitTermination(t) : $State_X(t) \in S_{FINAL}$; Termination detection
- **Correctness.** $\exists$t *Correct(t)* $\equiv$ $\forall$ t' $\geq$ t, $P_{FINAL}(t')$
- **Solution Protocol**: set of rules such that Termination and Correctness is reached.
  - *Plug-in*: a protocol where not all entities enters the terminal status (truncated protocol)
- **Knownledge levels**: local, implicit, explicit, common
- **Knownledge types**: metric, topological properties, topological map.
  - type-D: about input data, type-S: about the status of the system
- *Synchronous schedule*: all entities start in the same moment the protocol and have same internal clock values (thus also sync alarms).

Basic Problems and Protocols

G a graph; x,y arbitrary nodes:

- **distance** : d(x,y)
- **radius** x: r(x) = max{ d(x,y), $\forall$ y $\in$ G }
- **diameter** G : d(G) = max { r(x) , $\forall$ x $\in$ G }
- **radius** G : r(G) = min { r(x) }
- r(G) ≤ d(G) ≤ 2r(G)

Broadcast

Flood

- **R = STDR ∪ { UI+ }**
- **M = 2m - n + 1**
- *M* ≥ m , proof by contradiction
- **T ≤ d(G)**
- *T* = d(G)
- Special cases:
  - Trees: M = n-1; T = radius(init)
  - Complete Graphs
    - Flood: M = n(n-1)/2; T = n-1
    - Smart: M = n; T = 1
  - Hypercubes
    - in H_k, n = 2^k, m = n·k/2 = (n·$\log_2$n)/2 *(each node has k links...)*
    - M = n$\log_2$(n/2)+1 = O(n*logn) (with Flooding)
    - HyperFlood: constructs a SPT
      - Every x connected to every y with a (unique) decreasing path (Proof)
      - Termination. Trivial proof: $l_{i+1} < l_i$
      - Correctness: Every node recv the info (decreasing path)
      - M = n-1
      - T = k ; In H_k the bigger l value is k.

Wake-Up

WFlood

- Flood with 1+ initiators; awake nodes does not forward messages
- **2m - n + 1** (1 init) ≤ **M** ≤ **2m** (n init)
- *M*(WakeUp) ≥ *M*(BCast) ≥ m
- *T*(BCast) ≥ *T*(WakeUp) (in general)
- Special cases
  - Trees: n-1 ≤ M ≤ 2(n-1); M = n + k - 2, k are the initiators (proof by ind)
  - Complete graph: using WFlood M = O(n^2)
    - Using IDs we get O(0.5nlogn), they solve the *Election* problem
  - Hypercubes: TODO?

Traversal (sequential)

- Every traversal protocol solves the broadcast problem.
- The converse is not true (flooding violates sequentiality)

DFT (Depth First)

- As far as possible; possible **backtrack**

- **M = T = 2m** (on each link pass a req and an ack/nack.
- $M \geq m$, same proof as BCast
- $T \geq n-1$, sequential access... at least num of nodes
- **DFT Special cases**
  - Trees: M = T = 2(n-1) ; side effect: **VR ring** construction
  - Rings: M = T = 2n ; exploit topology: M = T = n
  - Complete Graph : M = n(n-1) ; exploit topology: M = T = 2(n-1)

DFT+

- Broadcast *Visited* msgs to neighbors (not to the one from where T has been received). Wait for *Acks*.
- **M = 2(n-1) + 2(2m-n+1) = 4m** 2(n-1) for T/ack ; 2(2m-n+1) for Visited/ack
- **T = 2(n-1) + 2n = 4n – 2** 2(n-1) for T/ack ; 2n for Visited/ack
- Reduced time complexity from O(m) to O(n)

DFT++

- From entity x concurrently send N(x) *Visited* msgs, there are no *Ack*s.
- There can be **mistakes**. Worst case: 2 mistakes on each backedge.
- Max backedges = m - (n-1); max-mistakes = 2(m-n+1)
- **M[NoMist] = 2(n-1) + (2m-n+1) = 2m+n-1**
- **M[MaxMist] = 2(n-1) + (2m-n+1) + 2(m-n+1) = 4m-n+1**
- **T = 2(n-1)** (assuming unitary delays there are no mistakes)

DFT+++

- T msg is used as an implicit '*Visited*'. No *Visited* msgs to T destination.
- Add *f* to the cost: tot num of nodes that doesn't require T because were already visited.
- **M ≤ 4m-2n+f+1** (used MaxMist cost , removed n *Visited*, but added *f Visited*)
- **T = 2(n-1)** ; same as DF++

Spanning Trees (SPT)

- The algorithms seen have cost O(m). In a tree m = n - 1.
- Every BC proto solves the SPT problem (trivial proof)
- Converse is also true. Thus the problems **computationally equivalents**
- **Global problem**: every entity must partecipate.
- Every UI proto that solves a global problem solves BC
- Given a protocol P, in general, $T[P|SPT(G)] \geq T[P|G]$ ($d[SPT(G)] \geq d[G]$)

Shout

- RI = { TR, BL, CN, **UI** }
- Flood (Q) + Ack (Y/N); x reply Y only to the first Q (parent)
- Termination follows from Flood termination
- Correctness: Q reach every node (because of flood correctness), that will have only one parent (induction).
- M = 2M[Flood] = 2(2m-n+1)
- T = T[Flood]+1 ; +1 to send the Ack from the farthest node
- Ack to parent and Q to children are sent together.
- $M \geq m$; proof by contradiction (same as Flood).
- $T \geq d(G)$; as Flood

Shout+

- Q message can act as an implicit N.
- **M[Shout+] = 2m**

DFT

- M = M[DFT]; T = T[DFT]

Notes

- Shout generates SPT with smaller diameters, but non deterministic
- **Breadth-First-SPT**: distance between a node x and root u same as in G
- **Broadcast-SPT** : d[BSPT(G)] ~ d[G] (not much greater)
  - Determine center c of G (will be the initiator)
  - Construct the BF-SPT
- With multiple initiators and under R, SPT problem is deterministically unsolvable (proof similar to elect)
- Multiple initiators required restrictions = IR = R ∪ **{ID}**

Multi-Shout

- k initiators uses the Shout protocol
- **M = k·M[Shout]**

Multi-Shout Select

- Selective construction: each entity is part of a single SPT, kills the other.
- Locally terminated nodes can be **resumed** by receiving of msg from initiator with smaller id.
- **Global termination** can be implemented adding broadcast from root.
- Multi-Shout constructs a SPT rooted in the initiator with min(id).

Computation in Trees

- T[x-y] subtree containing x, with removed link x-y
- d[x,y] = max {d(x,z) | z in T[y-x]}
- diameter d(T) : the longest distance between two nodes
  - if d(T) = d[x,y] then the path between x and y is diametral

Full-Saturation

- Three stages: **activation** (wake-up), **saturation** (converge), **resolution**
  - If used as a *plug-in* the resolution is skipped
- Only two nodes becomes saturated and are neighbours (proof)
- **M[FS] = M[A] + M[S] = 2n+k-2**
  - **M[A] = n+k-2** , k initiators
  - **M[S] = (n-1)+1 = n**
  - M[R] = (n-1)-1 = n-2 (just in case...)
- **T[FS] = max { t(l) + d(l,s) | l ∈ L}** : time to saturate s
  - t(l) = min{ d(l,y) + t(y) | y ∈ I } ; time to activate leaf *l* = min distance from a leaf to an initiator

Minimum-Find

- Generally use Full-Saturation + Resolution (to spread the min)
  - With rooted tree use Convergecast in place of Full-Saturation
- **M = M[FS] + M[R] = 3n+k-4**
- **T = T[FS] + max{ d(T[s2-s1]), d(T[s1-s2]) }**
- Other commutative semigroup operations (closure, associative, commutative) and cardinal statistics can be performed using FS.

Node Eccentricity

- With rooted tree, use Convergecast. M = T = 2(n-1). ????For every node: $n(2n - 2) = 2(n^2 – n)$???
- In general, **Saturation** is used:
  - given two saturated nodes x and y; x send to y the max dist from x to the nodes in T[x-y]+1;

    - a node z has all info, except max dist in T[p(z)-z], p(z) = parent;
    - distribution of parent distances starts from saturated nodes.
    - once an entity has computed its eccentricity will provide the missing info to its children
        - possibly different info in each subtree
- **M = M[MinFS]**
- **T = T[MinFS]**

Slow Center Finding

- **Eccentricity + MinFind** (only Saturation+Resolution)
- **M = M[EC] + M[Min]** = 3n+k-4 + [(n) + (n-2)] = 5n+k-6 ≤ 6(n-1)
- **T ≤ T[EC] + 2d ≤ 4d**

Fast Center Finding (Again)

- **Lemma 1**: One center or two (and are neighbours) and are on a diametral path.
- **Lemma 2**: d1[x] and d2[x] are the largest and second largest of all {d[x,y] | y in N(x)}, respectivelly.
    - A node x is a center iff **d1[x] - d2[x] ≤ 1**
- **Lemma 3**: Let y and z neighbours of x such that d1[x] = d[x,y] and d2[x] = d[x,z].
  If d[x,y] - d[x,z] > 1, then all centers are in T[y-x].
- The second statement allows to know if x is center in the Eccentricity phase.
- The third gives a tool to not perform the whole resolution of Eccentricity; proceed towards the deepest subtree.
- Since d(s,c) ≤ n/2, then **M[CenterPlugin] ≤ M[FS] + n/2** = (2n+k-2)+n/2 = 2.5n+k-2 ≤ 3.5n-2

Rooted Trees

- A distinct node: the root, the other links logically oriented towards the root.
- Tree to RootedTree transformation, under R, is **deterministially unsolvable** (proof). But it is under RI.
- ConvergeCast: saturation stage simplified, only the root is saturated. But just saves 1 message.
- Applications: ordered tree traversing (pre-order generates the same sequence), broadcast with termination detection (e.g. once that Shout SPT has been constructed we perform convergecast to notify termination, global termination).

Election

- Choose a leader because is required by the proto or to simplify the solution.
- Enforce the **UI restriction where it doesn't exist**
- Election problem , under R, is deterministically unsolvable (proof)
- Restrictions to "*Break the Symmetry*": **IR = R U {ID}** (UI is not a valid option, trivial).
  - *TRNG idea to assign dynamic IDS?*
- Solutions: MinValue (more perf when among the initiators only), RootedSPT (SPT to RSPT is unsolvable under R)

Election in Trees

MinFind

- Use Saturation protocol already described.
- **M = 3n+k-4** (M[FS] + M[Res]).

ElectRoot

- Leader is chosen between the saturated nodes
- **M = M[MinFind] + 2**
- At bit level is less expensive than MinFind because of empty saturation payloads (values are not carried).

Election in Rings

- Complete structural symmetry. Min finding approach is generally used.
- Single entity transmission direction is not influent.

All the Way

- Initiator entity x sends and waits return of its token (T_x).
- Entity x receiving T_y forwards it and, if not already done, sends T_x.
- Termination: when entity x received n different tokens; n is dynamically computed (the ring hops).
- **M = n^2**
- **T ≤ (n-1)+n = 2n-1**

All the Way+

- Only initiators sends their tokens (and increment hop counter). The others only forwards.
- At the end the relays are informed by the leader (+n msg)
- **M = nk+n** = n(k+1), k initiators (note: n(k+1) < n^2 => k < n-1)
- **T ≤ 2n-1 + n = 3n -1** (+n is to distribute the val)

As Far as it Can

- T_x a token generated by entity x with id #x.
- T_x blocked by entity y if #x > #y
- Only the msg of entity with smaller id will return to the sender (leader).
  - For termination detection a notification follows (+n msg)
- Cost depends on ids disposition in the ring.
- $T_{(i+1)}$ will be stopped by any with smaller id, there are *i* of them.
  - Thus, will travel at most *n-i* hops.
- **Worst case**:
  - If ids are sorted with increasing order respect to msgs direction and entities activates from the bigger.
  - **M = (1+2+...+n) + n = n(n+3)/2 = O(n^2)**
- **Average case** (oriented rings):
  - Consider all the arrangements equally likely
  - C(a,b) = combination of 'a' elements in groups of size 'b'

- **T_i travel *k* hops if the next *k-1* ids are greater ad the k^{th} is smaller or equal** (itself).
- There are ***n-i* ids larger** than *i* from which to choose those k-1 larger neighbors. There are **i-1 ids smaller** than *i* to choose the k^{th} id. Constraint: $1 \le k \le n$
- Cases where **T_i travels k steps**: Fav = **C(n-i,k-1) C(i-1,1)**
- Total ways to arrange the items: Tot = **C(n-1,k-1) C(n-k,1)**
- P(i,k) = Fav/Tot. **Exp[distance T_i] = Σ_k P(i,k)**
  - Note: if *i* = 1, then P(i,k)=1, (also via the formula)
- **Exp[M] = $n + \Sigma_i \Sigma_k P(i,k) = n \Sigma_k n/(k+1) = nH_n$** (Harmonic series)
  - $H_n \sim \ln(n) \sim 0.69 \log_2(n)$. Thus, **E[M] ~ $0.69n\log_2 n + O(n) = O(n\log n)$**
  - AsFarInit: E[M] ~ 0.69nlogk

Controlled Distance

- Election stage: limited distance, feedback msgs, check both sides. Let d_i distance at stage *i* and #x id of entity x.
- Possible, for delay variations, that different entities are in different stages. Smallest ID wins, regardless of the stage
- Termination: if not min, an entity will encounter another with smaller id. When $d_i \ge n$ the message from x completes the tour. Happens only if #x is the min.
- $n_{i+1} \le \lfloor n/(d_i + 1) \rfloor$, on stage *i* every candidate up to d_i is defeated, thus in a group of d_i+1 candidates only one survives to enter stage *i+1*.
- In stage *i*: the n_i entities sends 2 *Forth* messages in both directions. The n_{i+1} survivors receive 2 *Back* messages, the other $n_i - n_{i+1}$ receive one or zero *Back* messages. Each will travel at most d_i (*Forth* msg may be blocked earlier).
- **$M[Stage_i] \le d_i \cdot (2n_i + 2n_{i+1} + (n_i - n_{i+1}))$** $= d_i \cdot (3n_i + n_{i+1}) \le d_i \cdot (3 \cdot \lfloor n/(d_{i-1}+1) \rfloor + \lfloor n/(d_i+1) \rfloor) < 3 \cdot n \cdot d_i/d_{i-1} + n \cdot d_i/d_i = \ldots$
  $\ldots = n \cdot (3 \cdot d_i/d_{i-1} + 1)$, with $d_0 = 1$
  - The M total depends on the num of stages *k*.
  - The algo terminates as soon as $d_i \ge n$. Set *k* as the smallest *i* that verify the condition. We also set **$c = d_i/d_{i-1} \rightarrow d_i = c \cdot d_{i-1} \rightarrow d_i = c^i$** . Thus **$d_k = c^k \ge n \rightarrow k \ge \log_c n \rightarrow k = \lceil \log_c n \rceil$**
  - **$M[Control] \le n \cdot \Sigma_{i=1,k} (3c+1)$** $= nk(3c+1) = n(3c+1)\lceil \log_c n \rceil = n \log_2 n \, (3c+1)/\log_2 c$
  - Thus the messages are min when the constant $(3c+1)/\log c$ is minimized, the best choice is for c=3
  - **$M[Control, c=3] \le 6.309 n\log_2 n$** , (+n messages for termination broadcast) = **$O(n\log n)$**
- $Stage_i$ time is equal to $2d_i$ and there are max k stages.
  - **$T \le \Sigma_i \, 2d_i$, i=1:k**. Using **$d_i = c^i$** yields O(n) time : $\ldots = 2(c - c^{k+1})/(1-c) = 2c/(1-c) \cdot (1 - c^{\log cn}) = a(n-1) =$ **$O(n)$**

Stages

- Candidate *i* sends $Elect_i$ in both directions, without set a max distance.
- Msg travels until reach another candidate in the same stage (msg ordering restriction). No *Acks*.
- A candidate will recv a msg from each side, **survives** if its id is smaller than both.
  - $Elect_{i+1}$ acts as implicit *Nack* to defeated nodes.
- **Termination**: initiator with min id will never be defeated.
  Candidates decreases at each stage: one survivor → two defeated adjacent candidates (left, right)
- **Correctness**: leader is the only one receiving its msg back on both sides
- Given two neighbor candidates, in stage *i*, only two messages are sent between the two (one in each direction). Thus, *2n* msgs for each stage.
- **Max stages**: for each survivor there are, at least, 2 defeated candidates (left/right). Thus, $n_i \le (n_{i-1} / 2)$
  $2n_i \le n_{i-1} \rightarrow 2^{i-1} n_i \le n_1 = n \rightarrow$ if the stages are σ , $n_\sigma = 1 \rightarrow$ **$\sigma \le \log_2 n + 1$**
- **$M[Stages] \le 2n(\log_2 n + 1) = 2n\log_2 n + O(n) = O(n\log n)$**
  - M[Stages,MinInit] = $2n\log_2 k + O(n)$, k initiators
- We've assumed that messages in stage *i* arrives before messages in stage *j* > *i*.
  - Ordering can be simulated by eventually enqueuing higher stage messages.

Stages*

- Message ordering is not assumed.
- Give stage number to all nodes, for defeated nodes the stage where has been defeated
- Defeated nodes only forward messages from higher stages.
- If entity x in stage *i* receive M* with stage *j* > *i*:
  - if x is defeated in this round, it will forward M*
  - if x survives, then it implicitly know that it would defeat all nodes with stage less than *j*,
    - gains *j-i* message **credits on that port**. While credit > 0, it doesn't have to wait (and even send) messages from that port during current stage, just the other port is active.
  - Reception of M* with *j* > *i*, can prevent a node to send msg for stages *j* - *i*, because is defeated. Thus is more efficient than *Stages p*rotocol.
- Out of order messages disappear under "ideal time" restriction.

Stages with FeedBack

- If, in $Stage_i$, candidate x recv $Elect_i$ from candidates y=left(x) and z=right(x) then x sends *Feedback* to y if #y < min{#x,#z}, to z if #z < min{#x,#y}, and nothing otherwise.
- Candidate survives a $Stage_i$ only if receives a feedback from both sides.
- A node sending a feedback doesn't survive.
- A node that doesn't receive a feedback in $Stage_i$ (even from a single port), will receive a $Stage_{i+1}$ *Elect* message from the candidate that defeated it in the $Stage_i$. This acts as an implicit *Nak* (the node becomes defeated and forwards it).
- **Correctness/Termination**. Entity with smallest id will always receive a positive feedback from both sides. In $Stage_i$ its left/right candidates never survive. Thus n_i is decreasing.
- In each stage we send 2n *Election* messages, a candidate sends at most one *Feedback*. Thus max 3n msg per stage.
- **Max stages**: if x survives in $Stage_i$, then we know that right(x), right(right(x)), left(x), left(left(x)) were defeated.
  - $n_i \le (n_{i-1} / 3) \rightarrow 3n_i \le n_{i-1} \rightarrow 3^{i-1} n_i \le n_1 = n \rightarrow$ if the stages are σ , $n_\sigma = 1 \rightarrow$ **$\sigma \le \log_3 n + 1$**
- **M[Stages] ≤ $3n(\log_3 n + 1) = 3n\log_3 n + O(n) = 1.89n\log_2 n + O(n) = O(n\log n)$**
  - At bit-level gain is even greater, the feedback do not carry payload.

Alternate

TODO?

- **M[Alternate] ≤ 1.44 nlogn + O(n)**

Election in Mesh Networks

TODO (important)

Election in Torus Networks

TODO (important)

Election in Complete Networks

- **Densest** network (m=n(n-1)/2) and **smallest diameter** (d=1), contains any other net as a subgraph.
- Entity x sends *Capture*<#x,$stage_x$>; captured entity remembers its owner id and stage.
- Attack x to y is successful if y was not owned and #y < #x.
- If y is owned by z and $Stage_x > Stage_z$, then y sends *Warn* to z; if $Stage_z < Stage_x$ or ($Stage_z = Stage_x$ and #z < #x) the attack to y is successful and z becomes passive (not immediatelly acquired). Note that if y detects that $Stage_z > Stage_x$ *Warn* is not sent (Stages cannot decrement with time).
- Additional intelligence in captured nodes required: **warning queue** and **locally solve** warn when possible.
- In $Stage_i$ each candidate has a territory of size *i*, territories are disjoint. Thus in Stage *i*, $n_i \le n/i$. Candidate generates max 4 messages in each stage. Candidate that enters stage n/2 + 1 is the leader.
- **$M = \Sigma_{i=1,n/2}\ 4n_i \le 4n \Sigma_i\ (1/i) = 4n\ H_{n/2} = 4n\log_e(n/2) = 4n(\log_e n - \log_e 2) = 4n(0.69 \log_2 n - \log_e 2) =$** ...
  … = $2.77n(\log_2 n - 1)$ (+ n-1 messages for termination broadcast) = **O(nlogn)**
- **$T = 4(n/2 + 1) \le 2n + 4 = O(n)$** , there are max n/2+1 stages and each stage has a max duration of 4 messages.

Election in Chordal Rings

- Complete graph as a ring with chords. Added restriction: **Chordal labeling**, $\lambda_x(x,y) = k$ then $\lambda_y(x,y) = n-k$
- Labels 1 and n-1 form a ring. We can just use a ring election protocol. But we can do better.

KElect

- Candidate x in $Stage_i$ sends *Elect* on both sides until canditates y and z are reached. In the next stage all defeated items between them are bypassed (use hop counter to identify the port to use)
- Receiver knows that port n-i connects with the originator.
- If a shortcut is used the hop count must be incremented by the shortcut port val.
- R(i) ring used in $Stage_i$, $R(1)=K_n$, with stages proto: M = 2(n(1)+...+n(k)), n(i) size of R(i) and k ≤ logn

- R(2), … R(k) do not have common links and no overlap. Form G a planar graph, m(G) ≤ 3(n-2), Thus:
- M[KElect,Stages] ≤ 2(n(1)+m(G)) =9n-13
- Chordal Rings: TODO

MegaMerger

TODO

Message Routing

- If G is connected, communication possible via Broadcast. Θ(a,b) link cost from a to b (or length).
- Shortest path routing problem using **routing table** (sufficient info at each entity).
- Restrictions RI= {BL, CN, TR, **ID** }
- Generally the algorithms assumes that there is a SPT already constructed.

Gossip Map

- Maintain complete network map with the costs. Compute shortest path (e.g. using Djikstra).
- Initially x has only one row: Map[x,*]. Every entity BC its info on the tree.
- Each entity acquires its neighbor link cost info (tot cost 2m) and then **broadcasts N(x) items of info**.
  M = 2m + Σ_xN(x)(n-1) = 2m + 2m(n-1) = **2mn = O(nm)**.
  - If long messages allowed M = 2m+Σ_x(n-1) = 2m+n(n-1)

Iterative Construction

- Network map is not saved. We use a **distance vector** (V). Initially x only knows costs of N(x).
  - Storage cost is limited to the routing table size
- Every update is an iteration.
- $V_x^i[z]$: cost of shortest path from x to z at iteration *i*, in the x Vector.
- On each iteration i, x recv the vector of the neighbors and for each destination z computes
  $V_x^i[z] = \min_{y \in N(x)}\{ \Theta(x,y) + V_y^i[z] \}$
- Shortest path property: **$\gamma(x,z) = \min_{y \in N(x)}\{ \Theta(x,y) + \gamma(y,z) \}$**
- **Convergence** : at most, **n-1 iterations** (max tree diameter).
- In each iteration, each x sends **n items of info** to its N(x) neighbors. Thus, one **iter cost: 2mn**
  M ≤ 2mn(n-1) = O(n^2m).
  - If long msg allowed M ≤ 2m(n-1)
- τ to tx n items of info to one neighbor. Then, **T = τ(n-1)**

Shortest path SPT

- Given entity *s*, paths constructed with GossipMap form a shortest path SPT rooted at *s* (aka ***SinkTree***$_s$ *or* ***PT(s)***).
  - Generally different from the tree used by the GossipMap algorithm.
- Routing tab at *s* can be constructed by directly constructing PT(s).

Serial strategy (*Abstract*)

- T is a connected **fragment** of PT(s)
- To each link going outside of T give a value: **$v(x,y) = \gamma(s,x) + \Theta(x,y)$** , cost to reach y passing through x.
- Add to T the **outgoing** link with min v(x,y).
- **Termination** is guaranteed since T grows monotonically.
- Assumes that link x knows which links are outgoing and which are internals.

PT-All

- Implements the serial strategy
- Iteration started with root *s* broadcast in T of *IterStart* msg.
- Each entity x in current T computes, locally, v(x,y) for each outgoing link not in T and elects one that has min cost.
  - To prevent selection of internal links: when a new link is added to T, it notifies the neighbors about it.
- ConvergeCast to send min to the root.
- Overall min v(a,b) among all is computed by *s*. The link is added to T.
- The root notifies b about the selection, sending also γ(s,b) to it. The node b notifies all adjacent nodes (plus ack).
- Then *a* sends a notification to s to start the new iteration.
- The iteration composed by four ops: **Broadcast**, **ConvergeCast**, **NotifyTheSelected**, **NotifyTheRoot**.
  - Each op performed in current rooted tree.
- M[PT-SingleEntity] $\leq 2n^2 + 4m - 3n + 1$
  - Broadcast and ConvergeCast one msg per link $2(n_i-1)$. Notifications are sent directly to the dst, max (degen tree) n_i-1 msgs in each direction. Two additional msg are sent to the new node for expansion.
    - Each iteration adds one node, thus $n_i = i$
  - There will be **n-1 iterations** each adding a new node.
  - In each iteration *i*, $\mathbf{M_i \leq 4(n_i-1) + 2 = 2(2i - 1)}$.
  - $\mathbf{M[Expansion] = 2\,\Sigma_i\,(2i - 1) = 2n^2 - 4n + 2}$.
  - Add the cost to send msg from the selected node x to the N(x) nodes (and ack): $\mathbf{2\,\Sigma_x |N(x)| = 4m}$
  - Finally add **n-1** messages **for term broadcast**.
  - $\mathbf{M[PT\text{-}All] \leq n \cdot M[PT\text{-}SingleEntity] = O(n^3)}$

SparserGossip ??????

- Clever construction of the SPT of the network: *sparser*, then simulate execution of *GossipMap* on it.
- V'⊆ V. Eccentricity of x ∈ V': r(x,V'); r(V') = min { r(x,V') }
- Density od x ∈ V' : den(x,V')= | N(x) ∩ V'|
- Given a collection *A* of subsets of nodes, r(A) is the largest of radii of those subsets; den(A) the sum of densities of those subsets.
- (a,b)-sparser is a partitiion of V into subsets such that r(S)=a and den(S)=b, for each subset S.
  - sparser V' = <V'$_1$, …, V'$_k$>
  - elect a leader x_i in each V'$_i$
  - establish a path connecting the two leaders of each pair of neighboring subsets.
- The exec of protocol in G is simulated in the sparser
  - Each leader exec the algorithm for each node in its subset
  - When the algorithm requires to send a message from a node in V'$_i$ to V'$_j$ then the message is sent by x_i to x_j.

Min-Hop Routing

- All links have the same cost, assume unitary. **γ(a,b) = d(a,b)**
- Shortest path SPT here coincides with the **Breadth-First SPT**

BF

- Similar to PT-All but we choose several links simultaneously. In step *i*, s chooses all nodes at distance *i*.
- Iteration *i*, nodes at dist *i* sends an *Explore* msg to nodes y at dist *i+1*; y sends an *Ack* to the first *Explore* received, *Nak* to the others.
- The neighbors of y are nodes at dist *i+1* and *i-1*, and y know the nodes at dist *i-1* (sent Explore in prev iteration).
- **M[BF] ≤ 2m + 2(n-1)d(G)**
  - dist(x)=*i*. Thus *x* sent an Ack/Nak to each node at dist *i-1* (in stage *i-1*) and an *Explore* to each node at dist *i+i in stage i*. $M_x = |N(x)|$.
    M[Explore] = Σ_x|N(x)|=2m
  - In iter *i+1* BroadCast and ConvCast are sent in iter *i* tree. Thus n_i-1 msgs per iteration.
    M[BC/CC] = $2\,\Sigma_{1 \leq i < r(s)}\,(n_i-1) \leq 2(n-1)d(G)$ **(indaga)**
- **T[BF] = $2\,\Sigma_{1 \leq i \leq r(s)}\, i$** = 2 (r(s)(r(s)+1))/2 = **$r(s)(r(s)+1) \leq d(G)^2 + d(G)$**
  - in iteration *i* nodes up to dist *i*-1 recv BC and send CC, nodes at dist *i* recv *Explore* and send *Ack/Nack*, thus each iteration *i* requires 2*i* time units.

BF-Levels

- BF **Sync-phases**: Broadcast and Convergecast. Need to reduce number of sync phases.
- In each iteration expand by $l > 1$ levels (with $l = 1$ it is the normal BF).
- *Explore(j, k)* : with *j* the level to be assigned and *k* a counter decremented as *Explore* is forwarded.
- *Ack* is sent back if the destination is not part of the tree of if its level is greater than j, else *Nak* is sent.
- *Explore* is forwarded by y only if y is not part of the tree.
- The first *Explore(j,k)* received by x doesn't determine the *x* parent for the final tree. It is possible that x recv later an *Explore(j',k')* with j' < j. In this case x throws away the previous work and restart.
  - If k' > 0 sends *Explore(j'+1,k'-1)* to all its neighbors except its new parent.
- In one iteration at most $j - t - 1 = l - (k+1)$ "***throw aways***", with t the current leaves level and *j* the first "acquired" level.
- **Correctness**: by induction, prove that each node up to dist t + l is attached at proper level to the existing fragment.
  - Base case: nodes at level t+1 receive *Explore(t+1, l-1)*. Their level will not change.
  - Assume true for nodes up to level *t+k.*
  - Let π be the path of len *t+k+1* from *s* to *x* and let *u* be the neighbor of *x* in this path. Thus *u* is at level *t+k*, by hyp. its level is correctly set. When this happened *u* sent *Explore(t+k+1, l-k-1)* to all its neighbors (parent excluded), including *x*. Thus *x* set its level to *t+k+1*. To terminate this phase *u* should have received a reply from its neighbors, including *x*.
- **Sync** cost
  - At most 2(n-1) messages as both ops are performed on current available tree.
  - **Iterations** $\leq \lceil d(G) / l \rceil$, as there are radius(root) ≤ d(G).
  - **M[Sync]** $\leq 2(n-1)\lceil d(G) / l \rceil \leq 2(n-1)^2 / l$ (d(G) ≤ n-1)
- **Expansion** cost:
  - In iteration *i* the nodes involved are the nodes at levels L(i) = { (i-1)l + 1, … , (i-1)l + l } and the *sources*.
  - Messages sent in this phase are *Explore(t+1, l-1)* … *Explore(t+l, 0)* and corresponding replies.
  - Each node in L(i) recv from a neighbor at most one of each of the *l Explore* messages.
    - Thus there will be on each edge at most *2l Explore* messages (*l* in each direction) and *2l* replies..
  - In iteration *i*, there are m_i links between the involved nodes. Thus there are at most $4lm_i$ messages.
  - Given that links on each iteration are disjoint, we have: **M[Expansion]** $\leq \Sigma_i 4lm_i = 4lm$, $1 \leq i \leq \lceil d(G) / l \rceil$
- **M = M[Sync] + M[Exp]** $\leq 2(n-1)\lceil d(G) / l \rceil + 4lm$
  - If $l = O(n/\sqrt{m})$ then $M = O(n\sqrt{m})$. If G is sparse, m=O(n), then $M=O(n^{1.5})$. If dense, $m=O(n^2)$, then $m=O(n^2)$
  - In worst case performs like BF, else performs better.
- **Time** Cost: iteration *i* consists of reaching levels in L(i) and return to root (no throw aways).
  - Hence ideal time is *2il* for $1 \leq i < \lceil d(G) / l \rceil$, and 2d(G) in the last iteration.
  - $T = \Sigma_i 2il + 2d(G) = d(G)^2/l + d(G)$
  - *If* $l=O(n/\sqrt{m})$ we have $T = O(g(G)^2\sqrt{m/n})$

Hacking (More messages, less time)

- Choosing $l = d(G)$ we obtain $M = O(m\, d(G))$ and $T = O(d(G))$, thus we reduce time at cost of more messages.
- Group the "sources" (i.e. leaf of current fragment) in clusters and coordinate actions of sources within the same cluster. Sources in same clusters are connected.
- TODO!?!

Suboptimal Solutions

- Guarantee of delivery is the only requirement
- Construct only one routing tree T of G (not n as with optimal solution). No more than diam(T) messages to traverse.
- **Center-based:**
  - root at center *c* of G. diam(G) ≤ diam(PT(c)) ≤ 2·diam(G) (Why?) (Ex. 4.6.27)
  - Two protocols: CenterFind and PT construction.
  - Best worst case
- **Median based:**
  - Traffic(T) = $\Sigma_{(x,y) \in T}$ | T[x-y] |·| T[y-x] |, assuming unitary link costs for each (x,y).
  - Min Traffic SPT: T* that minimize Traffic(T). Find T* is a **NP-hard** problem.
  - Let *z* be a *median* of the network: SumDist(z)= $\Sigma_{v \in V}$ d(z,v) is minimized.
  - Property: Traffic(PT(z)) ≤ 2·Traffic(T*) (Why?) (Ex. 4.6.29)
  - Two protocols: MedianFind and PT construction
  - Best average case.

- **Minimum-Cost SPT:**
  - Construction can be done with protocol MegaMerger

Routing Tree Evaluation

- **Stretch factor** : $\sigma_G(T) = \max \{ d_T(x,y)/d_G(x,y) \}$, with $x,y \in V$ (always ≥ 1)
  - Not equal to d(T)/d(G). $\sigma_G(T)$ refer to the same couple of nodes, d(T) and d(G) can include different couples.
  - Thus, max $d_T(x,y) \leq \sigma_G(T) \cdot d_G(x,y)$
- **Dilation** : $dil_G(T) = \max \{d_T(x,y) \}$, with $(x,y) \in E$
  - Is stretch factor where we only consider x,y such that $d_G(x,y) = 1$.
- **Edge-stretch factor**: $\varepsilon_G(T) = \max \{d_T(x,y)/\theta(x,y)\}$ (avrei fatto l'inversa)
  - Here we also consider the cost of the link, if the cost is not unitary
- Both measures the worst ratio between the distance in T and in G for the same pair of nodes and the same edge, respectively.
- For both the measures Stretch-factor and Edge-stretch factor in place of max we can consider the average.
  - Construction of SPT with low avg edge-stretch can be done effectively (Ex. 4.6.35, 4.6.36)
- The choice of the suboptimal tree to use can be biased by the above measures.

Coping with Changes

- If a link cost change, both the nodes can detect it.

Adaptive routing

- **Triggered** update: If a change is detected x updates its map and propagates the information
- **Periodic** update: the update is period driven
- **Iterative construction** or the **map update** (GossipMap) protocols can be started by the two nodes.
- In iterative construction we can **throw away** current tab and restart or **update current** table
  - The first is costly, thus practically not used.
  - The second can manifest the ***count-to-infinity*** problem (if periodic update is also used).
    - Partial solutions: split horizons and sh with poison reverse (vedi libro reti).
- **Oscillation**: when the cost is proportional to the amount of traffic this problem can manifest.
- Shortest path routing cannot be guaranteed until the the algorithm convergence.

Fault-Tolerant tables

- Work well with most one link failure (SLF)
- No maintenance or update required.
- In every node there are 2+ edge-disjoint paths for each destination: the shortest and the secondary.
  - Secondary paths does not necessary form a tree
- When link $e_s[x](p(x),x)$ fails the tree PT(s) disconnects in T[x-p(x)] and T[p(x)-x]
- A link $(u,v) \in G \backslash PT(s)$ that can reconnect the two is called a **swap-edge** for e_s.
  - The new path cost becomes $d_{PT(s)}(s,u) + \theta(u,v) + d_{PT(s)}(x,v)$
- Find optimal swap edge for each edge $e_s[x]$ in PT(s).
- This process must be repeated n times, one for each destination s (i.e. for each PT(s))
- Note : PT(x) can be different from PT(s), but the shortest path entry from x to s in PT(x) is equal to the one from s to x in PT(s).