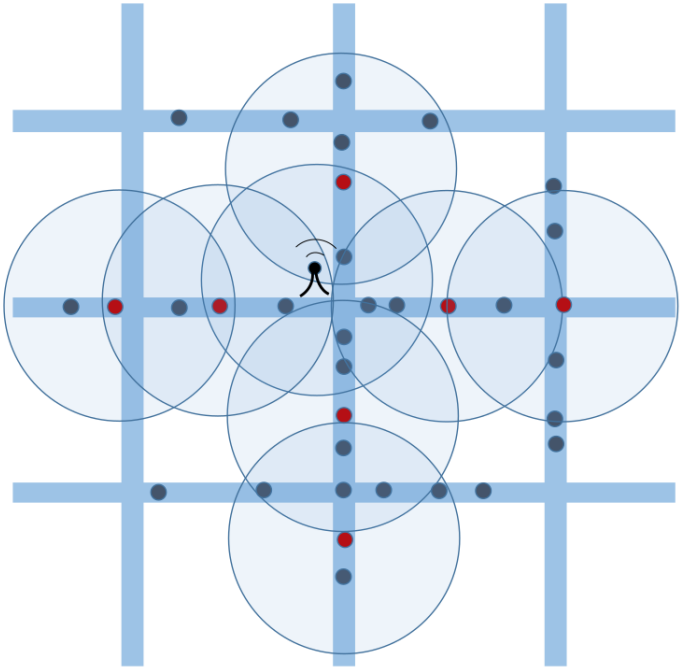
Dobbiamo selezionare un **sottoinsieme** di veicoli che fungono da **Relay Nodes** (RN).
All'inizio abbiamo un **RSU** come radice del grafo che semplicemente broadcasta il messaggio.
Settiamo il **numero di hop** in base all'area che vogliamo ricoprire.
Comincia una **reverse wave** dove i nodi generano e rimandano indietro dati alla sorgente.
2 tipi di pacchetti:

1. *Request*: creato dalla RSU; con essi si richiama la funzione che elegge i Relay Nodes.
   Questo pacchetto ha la seguente sruttura → **<ID, POS, HL, HLC>**
   ID: identificatore del pacchetto
   POS: posizione del mittente
   HL: hop limit
   HLC: variabile inizialmente uguale a HL e decrementata ad ogni hop

2. Reply: in direzione della RSU; essi contengono dati generati dai Relay Nodes.
   Questo pacchetto ha la seguente sruttura → **<ID, B>**
   ID: identificatore del pacchetto
   B: struttura di dati contenente più di una **Local Dynamic Map** (LDM)

Come funziona

L'RSU broadcasta. Ogni veicolo possiede una versione aggiornata della LDM.
L'idea chiave è di dividere l'area in cerchi di raggio D, con sotto-regioni parzialmente sovrapposte.
Eleggiamo come RN, il veicolo che sta più vicino al centro di queste sotto regioni. Questa posizione speciale, la chiamiamo Nominal Relaying Position (NRP).



V_{TX}: veicolo mittente.
V_{RX}: veicolo ricevente.
V_{RX} vede se è il più vicino al NRP rispetto a V_{TX} tra tutti i suoi vicini.

→ $dist(\ V_{RX}\ ,\ NRP\ ) < dist(\ v\ ,\ NRP\ )$ per ogni $v \in LDM_{VRX}$

Se è questo il caso, V_{RX} elegge se stesso come RV, decrementa HLC, controlla se HLC>0, aggiorna POS con la propria posizione e rimanda il pacchetto Request con un ritardo T_d (penso come il nostro).
Se non è questo il caso, invece, si mette in lista per un meccanismo di backup, nel caso in cui il più vicino non si elegga RV: il meccanismo sceglie il secondo più vicino e così via scorrendo la lista, mentre tutti aspettano che il processo termini correttamente.
A questo punto ogni V_{RX} crea un vettore delle distanze dei suoi vicini (incluso se stesso) ordinati in base alla distanza dal NRP e setta un timer: *Timer = T_d * posizione nel vettore*.
Se durante il Timer V_{RX} non riceve nessun'altra copia della Request, allora possiamo assumere che nessun altro vicino più prossimo al NRP si sia eletto RN.
Il pacchetto Reply è schedulato da un ritardo:

$$RepTimer_{V_{RN}} = T_{max}\frac{HLC}{HL}$$

Algorithm 1 RELAY NODE ELECTION

V_{TX}**:** the transmitting vehicle
V_{RX}**:** the receiving vehicle
NRP**:** the Nominal Relaying Position, it's the position at distance D from V_{TX}
$Vector$**:** a distance-based vector made of tuples $< v, dist(v) >$ where v is a vehicle and $dist(v)$ is the distance of v from NRP
T_d**:** a parameter denoting the maximum delay needed by a vehicle to broadcast a message

dist(V_{RX}) = computeDistance(V_{RX}.*coord*(), NRP);
Vector.add(V_{RX}, dist(V_{RX}));
for all $v \in LDM_{V_{RX}}$ **do**
    dist(v) = computeDistance(v.*coord*(), NRP);
    Vector.add(v, dist(v));
end for
sort Vector in ascending order according to dist
if $Vector.getFirst() == V_{RX}$ **then**
    return TRUE;
else
    setBackupTimer($T_d * posInVector(V_{RX})$);
    return FALSE;
end if