TAC Planning

Tracking area planning is done to balance signaling overhead that results from tracking area updates (TAU) with paging load required to reach UE camped on the network. As user traffic patterns change throughout the day and the over time, tracking areas and the associated lists sent to UE can be optimized, but such changes must be balanced with the cost of reconfiguration (for example, a problematic change could immediately trigger all UE in coverage to initiate a network connection to perform tracking area updates).

TAC design involves:

- Assigning a virtual tracking area code to each ground cell (semi-statically as part of TAC planning and optimization)
  - 65534 TACs are valid and if direct to cell uses dedicated PLMNIDs, the full range of TACs should be available. Therefore each cell could potentially have a unique TAC, but there's no real need or benefit to it, and in some cases a larger TA is preferred (due to TAL size limitation)
  - 22k cells of radius 11km are required to tile the continental US
- Assigning a physical TAC to each beam (for example) by choosing the virtual TAC with maximum overlap
  - Can explore changing TAC during the life of a beam if needed, but in terrestrial networks a TAC change typically requires an eNB restart (TAC is sent in SIB1)
- Determine the tracking area list for each beam, which the MME should assign to UE when updating a UEs TAL
  - NOTE: The TAL can contain a maximum of 16 different tracking area identities (see [24.301, 9.9.3.33])

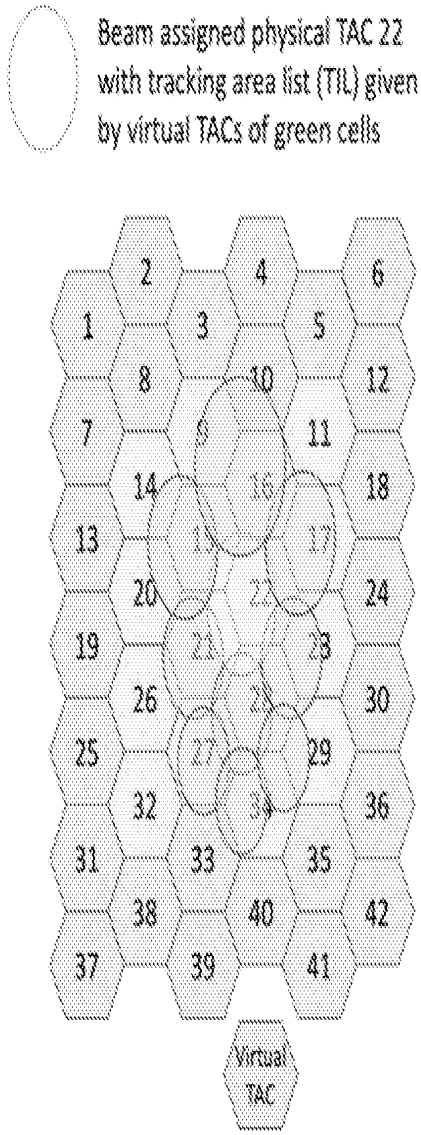
The high-level requirements and objectives for TAC planning are:

- A semi-stationary UE registered to the network should never need to do a TAU due to selecting a sector with TA not already in UEs TAL

- Estimating a UEs physical location within a sector is not required to meet the first objective
- Minimize the number of sectors that need to be paged to reach a UE
    - In addition to tracking the boundary of the TAL configured for each UE, paging can use a hierarchical approach where the page is first sent to the sector where UE as last seen, then surrounding sectors, etc.
- Minimize the rate of tracking area updates

- Semi-static allocation of "virtual TAC" to ground cells
- DTC GW anchors TAC from EPC point of view and maps virtual TAC to physical TAC for each beam
- A beam's TAC may need to change over time depending on how far it moves during its lifetime

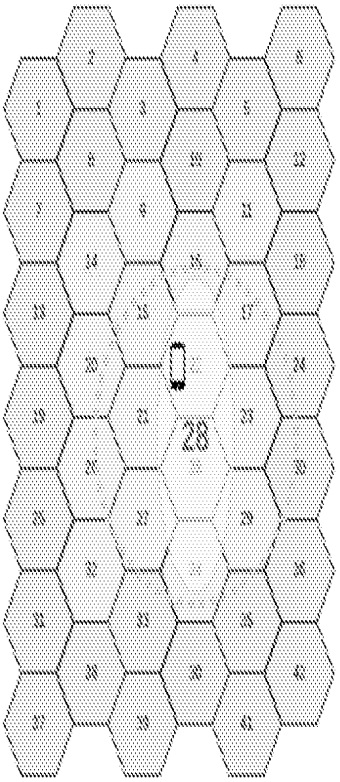
For example, at T = 1:



Tracking Area Updates

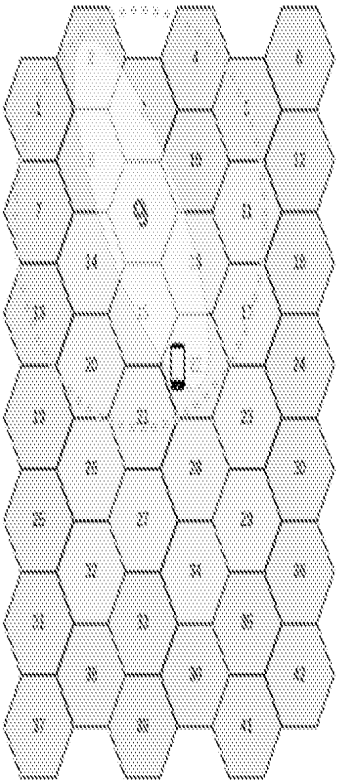
A goal of this TAC planning strategy is to ensure that a stationary UE does not have to make tracking area updates, while still ensuring tracking area updates if the UE moves so it can be accurately paged.

Let's examine the tracking area update behavior for both a stationary and moving UE:



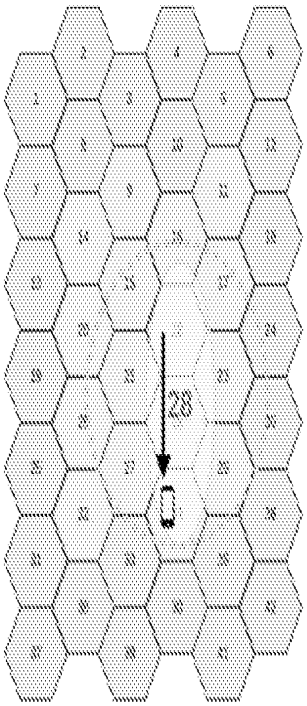
Stationary, T=2:
Stationary UE initially covered by beam centered at 22 becomes covered by a stretched beam centered at 28. Beam 28 is within is TAL so no update is required.

Maximum possible extent in all directions for a beam centered at virtual TAC 28 is shown by the dotted orange line.



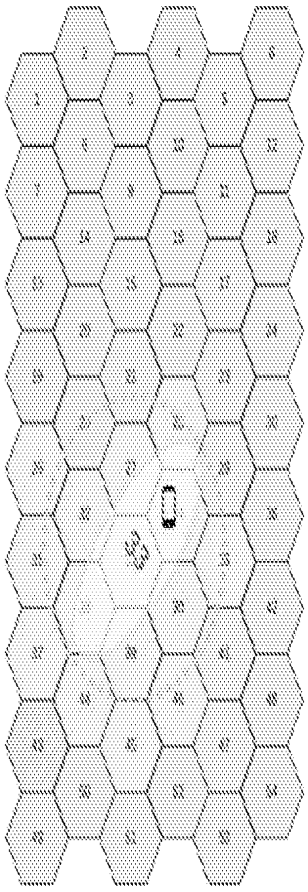
Stationary, T=3:
Coverage is now provided by a beam centered at virtual TAC 9.

This is still within the TAC list so no update is needed



Nonstationary, T=2:
The UE has moved from its original virtual TAC 22 to virtual TAC 34.

By chance, the same stretched beam centered at virtual TAC 28 covers it. Despite being outside physically its original tracking area list, it still observes an allowed TAC so it does not need to perform a tracking area update



Nonstationary, T=3:
TRACKING AREA UPDATE
The UE remains in place in virtual TAC 34 while the beams update around it. It is valid for the beam scheduler to select any beam whose maximum extent overlaps virtual TAC 34 to cover this area.

The UE is served by a beam centered at virtual TAC 33. This is not in the current TAL so it requests a tracking area update.

The core instructs the UE to replace its TAL with the TAL associated with TAC 33, the TAC that it received in the broadcast from the beam.
The new TAL contains all virtual TACs whose maximum extents overlap with virtual TAC 33.

Key properties of the tracking area update:
* The new TAL always contains some overlap with the old TAL.
* The new TAL is not necessarily centered at where the UE is, but is based on where the UE reports itself as being based on the broadcast TAC.
* It may take time for the UE's TAL to converge on one that contains the maximum extent of all beams that may overlap its physical location over time. Convergence is impacted by changes in both UE location and beam footprint. Convergence is statistically guaranteed for a UE that remains stationary given sufficient time.

A UE that has converged on a correct tracking area list may need to update with a move of as small as one virtual TAC. It may also move out to the far maximum extent of a beam footprint centered at the edge of its current tracking area list without a guaranteed UE-motion triggered update.

The probability of a tracking area update triggered by beam footprint changes increases with distance from the center of the tracking area list.

TAC Index Calculation from H3

The direct to cell beam footprint at nadir is 48km in diameter. This is very close to Uber H3 Resolution 4 (R4) cells, which are 52 km in diameter. The larger our TACs are, the less likely it is a user needs to do a tac update by being on the fringes of a beam outside their tracking area list. However paging a TAC consumes airtime on all the beams that overlap with it, so TACs that are too large make paging expensive. A TAC size close to a beam footprint is ideal.

The native width of an H3 cell ID is 64 bits. The minimum number of unique bits required to represent an r4 cell if flags and resolution metadata are omitted is 19 bits. Tracking Area codes are 16 bits long. We can map from R4 cells to tracking area codes that do not repeat within a tracking area list.

If we define the TA List as an r4 cell and all its first and second neighbors, we can generate tracking area codes that do not repeat within any TA List by using the 16 least significant bits of the zlib crc32 hash computed on the H3 cell ID.

The code to calculate this tracking area code in python looks like this:

```
from binascii import crc32

def tac_hash(h3_cell):
    return int.to_bytes(crc32(h3_cell), 4, 'big')[2:4]

# Run it on an example r4 cell ID 08438c05ffffffff
h3_cell = bytes.fromhex('08438c05ffffffff')
tac = tac_hash(h3_cell)
print(hex(int.from_bytes(tac, byteorder='big')))
# prints 0xd008

No collisions are found within search radius of 2 r4 cells
Closest hash collision is cells 848bb3dfffffffff and 84a8e97fffffffff. They are 6 cells apart
```