ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2010 Prof. Dr. Roger Wattenhofer, Johannes Schneider, Philipp Sommer

Ad Hoc And Sensor Networks
Sample Solution to Exercise 13

Assigned: December 20, 2010
Due: -

1 Mobile IP

a)
- The redirection is not anymore transparent to the user, it requires an explicit lookup. As a result, existing applications need to be updated / rewritten.
- When MA moves, the IP changes. The MA can announce its new IP to its HA *and* CN. But this requires all connections held by CN to be updated (which is not supported as of today). (Why not?)
- In the classic mobile IP approach, the HA may buffer messages sent to the MA while the MA is disconnected and send them when the MA announces its new location. With the direct routing approach, this becomes much more challenging, especially, if the MA changes its network address frequently.

b) *Publish*(MA,current network address of MA): The MA sends its network address to its HA whenever its address changes.

Lookup($MA - name$): The CN sends a lookup request to the HA and obtains the current network address of the MA. The CN executes a lookup before it sends the first packet to the MA. Because the MA may change its position at any time and obtain a new network address, the CN may need to call this method more often, especially if the packets are not acknowledged any more.

2 Mobility in MANETs

a) We may have a *linear* network, where the home agent is at the left end and the destination node t at the right end. Even if the sender node s is close to the destination t, the sender first sends its message to the home agent, which introduces an arbitrarily large overhead (we can make the node-chain arbitrarily long). Example:
HA x x x x x x x x x x **s** x x **t**

b) MLS has several *location pointers* (aka location servers) for each node. The location pointers of a node t are arranged such that if the sender is close to t, it does not have to walk too far to reach a location pointer. Technically speaking, MLS ensures the following property: If the optimal distance between s and t is d_{opt}, the path to find the first location pointer of t is $d_{lookup} \leq c \cdot d_{opt}$, with c being a constant.

We have seen in the lecture that the search for a first location pointer follows a spiral-like path. First, s assumes that t is very close. If this is really the case, t has stored a location pointer on its lowest level also very close to s. Therefore, s first tries to find one of these location pointer. If this search is not successful, s assumes that t may be a little bit further away, and searches on the next higher level, and this search continues recursively. The trick of this approach is that the lookup path on every new level is about twice as long as the lookup path on the previous level. As a result, when s finally finds a location pointer on level k, the cumulated lookup cost on the levels $1 \ldots k-1$ is about the same as the cost for searching on level k.

Finally, observe that if s does not find a location server of t on the surrounding level-$(k-1)$ cells, it knows that t is *not* located in any of the visited cells. But this means that t is at least 2^{k-1} away from s, giving a lower bound on d_{opt}. Searching for a location pointer on level k costs about $9 \cdot 2^k$ (neglecting some small factors), and the cumulative cost for searching on the levels $1 \dots k$ is roughly $18 \cdot 2^k$, which is only a (not very small) constant larger than d_{opt}.

c) The location pointers do not store the exact position of the node, but only in which of the four sub-cells the node is located. This avoids many updates if the node only moves small distances. Furthermore, a node needs to leave a given cell c by quite some amount until it updates the location pointer which points to the cell c.

Of course, there are sometimes situations where the node needs to send many costly update messages to location pointers which are far away, even though it only moved a very short distance. But once this is done, we know that the node needs to move quite some distance until the next big update is outstanding. To analyze such behavior, we use an *amortized analysis*, which examines the cumulated publish cost over a long time. It can be shown that this amortized publish cost is bounded by the total distance a given node moved.