Appendix A
Notation Description

The main notations and definitions in this paper are as follows:

- N: universal set of mobile nodes.
- m: total number of all mobile nodes.
- n_i: mobile node i.
- D_{ij}: the maximum communication distance between n_i and n_j.
- d_{ij}: the realistic distance between n_i and n_j.
- r_{ij}: link reliability between n_i and n_j.
- $r_{ij}(t)$: link reliability between n_i and n_j during the period $(0, t)$.
- $P_m(i, j, t)$: the probability of link failure caused by nodes' movement.
- $P_e(i, t)$: the probability of node error for n_i.
- λ_i: the average number of error in per unit period for node n_i.
- λ_{ij}: coefficients for link reliability that related to d_{ij} and v_{ij}.
- v_{ij}: relative velocity between n_i and n_j.
- G: a graph for the mobile devices' communication network.
- E: the edge of G.
- R: adjacency matrix of G.
- Ω_i: the neighbors of n_i.
- Θ_i: the partners of n_i.
- α_i: the threshold for chosing partners of n_i.
- $x_i(t)$: the information of n_i.
- $fusion(\cdot)$: the fusion function of information.
- $w_i(t)$: model parameter vector in n_i.
- $d_i^{(j)}$: data sample j of data set D_i.
- $g_{ip}(t)$: if the information from n_p is received successfully by n_i.
- ρ_i: the number of n_i's partners.
- s_w^2: variance of models in nodes.
- C_i: computing resource budget of n_i.
- B_i: communication resource budget of n_i.
- c_{l_i}: computing resource comsumed in local updates in n_i.
- c_{n_i}: computing resource comsumed in network analyze in n_i.
- b_{n_i}: communication resource comsumed in network analyze in n_i.
- c_{α_i}: computing resource comsumed in α-gossip in n_i.
- b_{α_i}: communication resource comsumed in α-gossip in n_i.
- ∂_i: the number of n_i's partners neighbors.

Appendix B
Functions for α-GossipSGD Algorithm

The three steps in the GREAT are implemented in the form of function algorithms: *LocalUpdate*, *MNLRS*, *Alpha-Gossip*. In order to synchronize the learning process in each node, we use time t_a, t_b, t_c, t_o as global knowledge before learning to control the switch of these three steps. Where t_a, t_b, t_c is a deadline for *local update*, *network analyze* and *α-gossip*, t_o is a reserved time between each step to make the switching elegant. A learning iteration is equal to $t_a + t_b + t_c + t_o$ and the total learning time is $T(t_a + t_b + t_c + t_o)$. In general, t_a,t_o is pre-determined according to the computing resource status of the node, and t_b and t_c are pre-adjusted according to the network status.

B.1 Local Update

Function Local Update is presented in Algorithm 2. It is correspond to the *local update* step where each node n_i use the local data D_i to train its model through SGD. We take these variables as its input: the samples in local data set D_i, the number of samples that has not yet been used for training n_d, the predefined local update time t_a for each iteration, current model w_i which is stored in *local model release* in Figure 2 of this paper. and the learning rate η. And the return of it is the trained model w_i', estimate value of resource consumption $\hat{c}_l$ and a refreshed integer n_d.

Algorithm 2 LocalUpdate

Input: D_i, n_d, t_a, w_i, η
Output: w_i', n_d, $\hat{c}_l$

function LocalUpdate(D_i, n_d, t_a, $w_i(t)$, η)
  initialize $timer \leftarrow 0$, $w \leftarrow w_i(t)$;
  activate *timer*;
  repeat
    $p \leftarrow$ random integer with $0 \le p < n_d$;
    compute $\tilde{w}$ by taking $D_i[p]$, w and η into equation (11);
    $w \leftarrow \tilde{w}$
    exchange $D_i[p]$ and $D_i[n_d - 1]$;
    $n_d \leftarrow n_d - 1$;
    if n_d=0 **then**
      $n_d \leftarrow length[D_i]$;
    end if
  until $timer > t_a$;
  estimate resource consumption $\hat{c}_l$;
  $w_i(t + t_a) \leftarrow w$;
  return $w_i(t + t_a)$, n_d, $\hat{c}_l$;
end function

In this function, a *timer* is activated at the begining to control the process of local update. This *timer* will run alone and end the train cycle when $timer > t_a$. We take D_i as an array list of data samples where the trained samples is the elements with larger or equal index (begining from 0) than n_i. In the train cycle, it will randomly select a data sample form un-trained samples and then perform stochastic gradient descent according to equation (11). After computing, the selected sample will exchange with the last untrained sample in D_i.[4] Finally, once this train cycle is stoped, the value of variable w_i', n_d and $\hat{c}_l$ be returned.

B.2 MNLRS

Function moving nodes link reliability sort (MNLRS) is proposed for analyzing the link reliability between n_i and n_j, which is correspond to the *network analyze* step in the α-gossip learning. It is deployed in each node n_i, and its result Ω_i is a list of tuples, which is sorted in descending order of link reliability. We focus on mobile devices with link compliance Assumptions 1 and 2, and take mobility and node failure as the main criteria to measure reliability.

Therefore, function *network analyze* analyze the reliability in an alternative method from equation (5). In this algorithm, most of the input parameters are predefined include *id*, error rate λ_i, time t_b and t_c. And the observed velocity of n_i is V_i obtained from

4. Though exchange $D_i[p]$ and $D_i[n_d - 1]$ might be meaningless when $p = n_d - 1$, exchange them directly is more efficient than judge whether they are equal, since frequency of unequal frequencies is much greater than equal, especially when D_i is huge.

Algorithm 3 Moving Nodes Link Reliability Sort (MNLRS)

Input: id, λ_i, t_b, t_c, V_i,
Output: Ω_i, $\hat{c}_n$, $\hat{b}_n$
function MNLR(id, λ_i, t_b, t_c, V_i)
  initialize $timer \leftarrow 0$, $listener \leftarrow [null]$, $\Omega_i \leftarrow [null]$, $temp = 0$;
  activate $timer$,$listener$;
  broadcast greet information: (id,λ_i, V_i);
  repeat
    while $length[listener] > temp$ **do**
      $LQI, j, \lambda_j, V_j \leftarrow listener[temp]$;
      compute $\tilde{v}_i j$ according to equation (4);
      compute $\tilde{r}_{ij}$ according to equation (5);
      insert $(j, \tilde{r}_{ij})$ into Ω_i by descending orders of $\tilde{r}_{ij}$;
      $temp \leftarrow temp + 1$;
    end while
  until $timer > t_a$;
  estimate resource consumption $\hat{c}_n$, $\hat{b}_n$;
  return Ω_i, $\hat{c}_n$, $\hat{b}_n$;
end function

sensors. This algorithm first activates a *timer* similar to Algorithm 2 and a *listener* to buffer messages in the signal. Once there is a new message received by *listener* (i.e. $length[listener] > temp$), this message will be parsed and the source of it will be regarded as a neighbor. The message from n_j is a quadruples composed of and $RSSI$ greet information (id,λ_j, V_j). For each message, the reliably $\tilde{r}_{ij}$ can be calculated by equation (5) and (4) bassis on the parsed new message. After computing, the $(j, \tilde{r}_{ij})$ will be inserted into Ω_i by descending orders of $\tilde{r}_{ij}$ to facilitate link selection in α-gossip.

B.3 Alpha-Gossip

Function Alpha-Gossip is proposed for exchanging information and update local model w_i, corresponding to *α-gossip* step. It takes local model w_i, network analyze result Ω_i from MNLRS, rest communication and computing resources B_i, C_i, rest number of iterations T^* and coefficients $k_{c\alpha}$, $k_{b\alpha}$, k_{cn}, k_{bn}, c_l for links chosing as its input. Note that, the input valuables k_{cn}, k_{bn}, c_l is computed or taken form the same iteration, but $k_{c\alpha}$ and $k_{b\alpha}$ is calculated from the resource consumption $\hat{c}_\alpha$, $\hat{b}_\alpha$ at last iteration [5]. Basis on these parameters, these function can return an updated model w_i' and the consumed resources $\hat{c}_\alpha$ and $\hat{b}_\alpha$. Its pseudocode is shown in Algorithm 4.

In Equation (24) and (28), the value of α is equal to the ρ_i^{th} element in Ω_i, equation (29) can be equivalent to find a index ρ_i for Ω_i:

$$\rho_i = \max\left\{\rho_i^c, \rho_i^b\right\} \tag{30}$$

where ρ_i^c, ρ_i^b is obtained from equation (24) and (28).

We use equation (30) to simplify the choose process for links, on which the chosen links can be seen as the first ρ_i elements in Ω_i. After chosing, AlphaGossip send its local model w_i to chosen links and listen messages form its partners untill $timer > t_c$. Finally, it average all received models and local model and estimate resource consumption $\hat{c}_\alpha$, $\hat{b}_\alpha$ before the reserved time t_o is spent off.

Algorithm 4 Alpha-Gossip

Input: w_i, Ω_i, B_i, C_i, T^*, t_c, $k_{c\alpha}$, $k_{b\alpha}$, k_{cn}, k_{bn}, c_l
Output: w_i', $\hat{c}_\alpha$, $\hat{b}_\alpha$, ρ_i
function AlphaGossip(w, Ω_i, B_i, C_i, T^*, t_c, $k_{c\alpha}$, $k_{b\alpha}$, k_{cn}, k_{bn}, c_l)
  initialize $timer \leftarrow 0$, $listener \leftarrow [null]$, $W \leftarrow [null]$, $\partial_i \leftarrow length[\Omega_i]$, $temp \leftarrow 0$;
  activate $timer$, $listener$;
  if $k_{c\alpha} = 0$ and $k_{b\alpha} = 0$ **then**
    $\rho_i \leftarrow \partial_i$;
  else
    compute ρ_i^c according to equation (24);
    compute ρ_i^b according to equation (28);
    compute ρ_i according to equation (30);
  end if
  for $j = 0 \rightarrow \rho_i$ **do**
    send w to node $\Omega_i[j][1]$;
  end for
  repeat
    while $length[listener] > temp$ **do**
      append w_j to W;
      $temp \leftarrow temp + 1$;
    end while
  until $timer > t_c$;
  append w to W;
  $w_i' \leftarrow avg(W)$;
  estimate resource consumption $\hat{c}_\alpha$, $\hat{b}_\alpha$;
  return w_i', $\hat{c}_\alpha$, $\hat{b}_\alpha$, ρ_i;
end function

5. For the first iteration, i.e. $T^* = T$, α is 0 directly. That means exchange to every neighbors in Ω_i).