International Journal of Electrical and Computer Engineering (IJECE)
Vol. 9, No. 6, December 2019, pp. 4804~4814
ISSN: 2088-8708, DOI: 10.11591/ijece.v9i6.pp4804-4814

Congestion bottleneck avoid routing in wireless sensor networks

Sanu Thomas, Thomaskutty Mathew
School of Technology and Applied Sciences, Mahatma Gandhi University, India

Article Info

Article history:

Received Jul 23, 2018
Revised May 4, 2019
Accepted Jun 26, 2019

Keywords:

Bottleneck avoid routing
Congestion bottleneck nodes
Congestion levels
Dynamic programming
Maximum congestion

ABSTRACT

A new efficient method for detecting congested bottleneck nodes and avoiding them in the route formation in a wireless sensor network is described. Sensor nodes with a higher degree of congestion are excluded while determining the best routing path from a given source to destination in a multi-hop transmission. In a scenario where different communication paths have different maximum congestion levels, selecting that path which has least maximum congestion, is a challenging task. A modified Bellman-Ford algorithm is proposed to solve this problem efficiently. The proposed technique is very much useful for the optimal route selection for vehicles in metropolitan cities that avoids high traffic density junctions. Once the desired destination is specified, the traffic control system can use this algorithm to provide the least congested routes to the intra-city vehicles.

Corresponding Author:

Sanu Thomas,
School of Technology and Applied Sciences,
Mahatma Gandhi University,
Kottayam, Kerala, India.
Email: thomas.sanu@gmail.com

1. INTRODUCTION

Easy availability of low cost but reliable sensor nodes has increased the popularity and utility of Wireless Sensor Networks (WSN's) for diverse applications [1, 2]. When sensors are deployed densely with heavy traffic load, congestion is inevitable. Traffic in Event triggered WSN causes more congestion compared to that of periodic transmission [3]. In general, the traffic pattern in WSN is typically many-to-one. This introduces heavy congestion in nodes near to the sink or the base station [4]. Congestion causes packet losses, delayed delivery, early depletion of battery life *etc*. Several techniques are available to detect, mitigate and avoid congestion [5-7]. In this paper, the main objective is to determine the best multi-hop path that excludes the higher congestion nodes among several available paths from a given source to destination.

One way of controlling node level congestion is to use data rate control [8]. The second method is resource control [9] where more network resources like bandwidth and additional routes are provided to share the load. In resource control, multipath and alternate path routing are used to avoid or bypass the congested nodes or regions. The proposed work chooses the alternate path routing. Here, we introduce *the optimal path (route) selection* technique that avoids nodes having high levels of congestion. These high congestion level nodes, if included in the path, act as bottleneck nodes while forwarding data from a source to a given destination. The proposed solution is to select the optimal path that avoids these bottleneck nodes. The solution involves the application of Bellman's dynamic programming in an ingenious way which is the originality of this work.

Experimental result shows that the proposed method is 15-20 percent faster compared to 'Topology-Aware Resource Adaptation' (TARA) method [9]. Another advantage of our method is, it provides a higher degree of load balancing compared to TARA. In section 2, a brief description of the related work is discussed. Section 3 gives network model and assumptions. The main algorithm is described in Section 4. In section 5, comparison with other methods is discussed. Section 6 gives conclusion.

2. RELATED WORK

In [10], only a few important nodes are selected as representative reporting nodes instead of all the nodes which sense the event. This reduces the traffic load and thereby the congestion is reduced, but the data reliability is compromised to some extent because all the sensing nodes do not send their data. In [6], the authors identify the suitable backward and forward nodes for sharing the load. Then the congestion levels are predicted and the load is properly shared to reduce congestion. Here, the load balancing is not equalized over the transmission paths. TARA: “Topology-Aware Resource Adaptation to Alleviate Congestion in Sensor Networks”, is described in [9]. Here, additional nodes and links are brought into serve the present traffic. Congest prone nodes are partially bypassed to avoid congestion. This is essentially a truncated multi-path routing and load balancing is not fully addressed. CODA: “Congestion Detection and Avoidance in sensor networks", in [11] uses closed-loop multi-source regulation. In designing CODA, importance is given to energy saving in sensor nodes. Here, the computational overhead is relatively high. In [12], the authors propose ECODA: “Enhanced Congestion Detection and Avoidance”. Here, dual threshold buffers are used for congestion detection. Channel utilization is optimized by proper control mechanism. In this case, the computational overhead is higher than CODA and the control mechanism is rather complex. Extensive surveys on congestion alleviation techniques are given in [13, 14].

3. NETWORK MODEL AND ASSUMPTIONS

Consider a WSN with N homogeneous sensor nodes. We assume the presence of a Base Station (BS) or Sink that controls the WSN and collects data from the sensors. The communication is mainly either from the BS towards the sensor nodes or vice-versa. But any node can act as a source as well as a destination. Because of the limited transmission range of individual sensor nodes, the multi-hop communication is adopted between the source and the destination. The intermediate sensor nodes act as relay nodes. The following assumptions are made about the WSN.

a. Sensor nodes are homogeneous and static.
b. Sensors have limited battery energy.
c. The BS has sufficient power and runs the proposed centralized algorithm.

3.1. Wireless sensor network as a graph

The WSN is represented by a planar graph $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E})$ where $\boldsymbol{V}$ is the vertex set of N nodes and $\boldsymbol{E}$ is the edge set. The sensor nodes are identified and represented by the vertices 1, 2,.., N. The vertex or node set V is given by,

$$\boldsymbol{V} = [1, 2,... N - 1, N] \tag{1}$$

The edge set $\boldsymbol{E}$ is the collection of all the links of the network. Edge element $e(i, j)$ represents the edge between node i and j for all i and j in the range 1 to N. The edge value $e(i, j)$ is set to 1 if node i and j are within the transmission range of each other. Otherwise, $e(i, j)$ is set to ∞. Therefore, $e(i, j) = 1$ means nodes i and j are one-hop neighbors and there is connectivity between them and $e(i, j) = \infty$ means, i and j cannot communicate directly. Only connecting edges will be shown in the graph. One hop neighbor nodes are also called adjacent nodes. Here, bidirectional links are assumed between one-hop neighbors. Therefore, $e(i, j) = e(j, i)$. We take $e(i, i) = \infty$ to avoid self-loops. The collection of $e(i, j)$’s form the adjacency or connectivity matrix for the graph of the network.

3.2. Path from the source to the destination

A path from a source node s to a destination node t is a sequence of non-repeating adjacent (one hop) nodes starting from s and ending at t. Non repetition of nodes assures that the path is free of loops. Adjacency of nodes along the path assures continuous connectivity from the source to the destination. There can be several paths from s to t in a given graph (network).

3.3. Measure of congestion level at a node

Several metrics are used to measure the congestion level or the degree of congestion at a node (Akyildiz et al., 2002). Without any loss of generality, we take the queue length of packets at a given node as the measure of the congestion level at that node. It is assumed that the sizes of buffers at nodes are sufficiently large so that there is no loss of packets in any queue due to overflow. It is also assumed that the queue lengths change relatively slowly with respect to time so that during the calculation and rediscovery of the optimal paths, the queue lengths remain nearly constant.

In general, the congestion levels of the sensors nodes remain almost same in a session. The centralized controller, BS collects this information periodically. This period depends on the applications and characteristics of the network. The present congestion level of node *i* is represented by symbol Q_i for *i* = 1 to *N*. The congestion array for the entire WSN is written as,

$$\boldsymbol{Q} = [Q_1, Q_2, \dots, Q_i, \dots, Q_N] \tag{2}$$

3.4. Congestion levels along a path

Once, Q_i's are known for all the nodes, consider all possible loopless paths from a specific source node *s* to a given destination node *t*. Let the number of distinct paths from *s* to *t* be *M*. Let path *j* be represented by $\boldsymbol{P_j}$ as,

$$\boldsymbol{P_j} = [s,\ v_{j,1},\ v_{j,2}, \dots, v_{j,k}, \dots, v_{j,L(j)}, t\ ] \tag{3}$$

for *j* = 1 to *M*. In (3), *L*(*j*) is the total number of intermediate nodes along $\boldsymbol{P_j}$ and $v_{j,k}$ is the *k*th intermediate node of path $\boldsymbol{P_j}$ for *k* = 1 to *L*(*j*) with $v_{j,k} \in \boldsymbol{V}$. In $\boldsymbol{P_j}$ all the nodes along the path are connected. That is, the corresponding elements of the adjacency matrix are 1's as,

$$e(v_{j,k-1}, v_{j,k}) = 1 \tag{4}$$

for *k* = 1 to *L*(*j*) + 1. In (4), $v_{j,0} = s$ and $v_{j,L(j)+1} = t$. The (4) simply means that any two adjacent nodes in $\boldsymbol{P_j}$ are within the communication range of each other.

The Congestion array (or vector) of a path is the sequence of the congestion levels of the nodes along that path. For the path specified by (3), the array that represents the congestion levels is represented by $\boldsymbol{CL}$ as,

$$\boldsymbol{CL_j} = [\ Q_s, Q_{j,1}, Q_{j,2}, \dots, Q_{j,k}, \dots, Q_{j,L(j)}, Q_t\ ] \tag{5}$$

Here, the source and the destination nodes are fixed (specified) for all possible paths from *s* to *t*. Since the packet stops at *t*, the congestion at *t* is not relevant for the packet travelling from *s* to *t*. Therefore, term Q_t in (5) can be ignored. In defining the effective Congestion Vector C_j for the purpose of determining the optimal path, we exclude Q_t from (5). The resulting effective congestion vector is,

$$\boldsymbol{C_j} = [\ Q_s, Q_{j,1}, Q_{j,2}, \dots, Q_{j,k}, \dots, Q_{j,L(j)}\ ] \tag{6}$$

Here, $Q_{j,k}$ is the congestion level of node $v_{j,k}$ in a proper unit and *k* varies from 1 to *L*(*j*). That is, $Q_{j,k} = Q_{v_{j,k}}$. Thus $Q_{j,k} \in \boldsymbol{Q}$.

Example 1: To demonstrate the formations of $\boldsymbol{P_j}$'s and $\boldsymbol{C_j}$'s, a simple network is shown in Figure 1. Here, source *s* = 1 and the destination *t* = 5 with number of nodes *N* = 5 and the number of distinct paths, *M* = 4. Congestion at source is taken as 0, which will be explained later.
The paths from 1 to 5 and their congestion levels are,
$\boldsymbol{P_1}$ = [1, 2, 5]. $\boldsymbol{C_1}$ = [Q_1, Q_2] = [0, 40].
$\boldsymbol{P_2}$ = [1, 2, 4, 5]. $\boldsymbol{C_2}$ = [Q_1, Q_2, Q_4] = [0, 40, 20].
$\boldsymbol{P_3}$ = [1, 3, 4, 5]. $\boldsymbol{C_3}$ = [Q_1, Q_3, Q_4] = [0, 30, 20].
$\boldsymbol{P_4}$ = [1, 3, 4, 2, 5]. $\boldsymbol{C_4}$ = [Q1, Q3, Q_4, Q_2] = [0, 30, 20, 40].

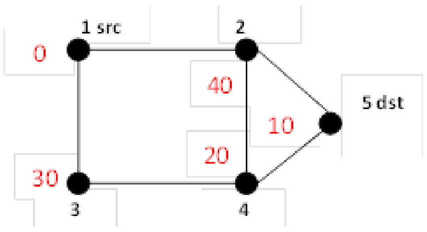

Figure 1. Network with 5 nodes

3.5. Maximum congestion level of a path

The maximum congestion level of path $\boldsymbol{P_j}$, represented by variable R_j, is defined as the maximum of array$\boldsymbol{C_j}$. That is,

$$R_j = max(\boldsymbol{C_j}) \tag{7}$$

The (7) also can be expressed as,

$$R_j = \max_{k\in\{1:L(j)\}}(Q_{j,k}) \tag{8}$$

This gives the maximum of $Q_{j,k}$ over k in the range 1 to $L(j)$. Thus R_j is determined by finding the maximum of the congestion levels of nodes forming the path excluding the source and the destination nodes. In Example 1, $R_1 = 40$. $R_2 = 40$. $R_3 = 30$. $R_4 = 40$.

3.6. Minimum among Rj's

Let $\boldsymbol{R}$ be the array formed by R_j's for $j = 1, 2, ..., M$ as,

$$\boldsymbol{R} = [R_1, R_2, \dots, R_j, \dots, R_M] \tag{9}$$

In Example 1, $\boldsymbol{R} = [40, 40, 30, 40]$.
Our objective is to find that index j, say J, where R_J is the minimum of the array $\boldsymbol{R}$. This can be stated as,

$$J = \underset{j\in\{1:M\}}{argmin}(R_j) \tag{10}$$

In Example 1, the min($\boldsymbol{R}$) occurs at index location 3. Therefore J = 3, R_J =30 and the optimal path is $\boldsymbol{P_J} = \boldsymbol{P_3}$.
Substituting for R_j from (8) in (10), we get,

$$J = \underset{j\in\{1:M\}}{argmin}\left(\max_{k\in\{1:L(j)\}}(Q_{j,k})\right) \tag{11}$$

Once J is obtained, the corresponding minimum among R_j's is R_J (the J^{th} element of array $\boldsymbol{R}$) and it can be expressed as,

$$R_J = min([R_1, R_2, \dots, R_M]) = \min(\boldsymbol{R}) \tag{12}$$

Once J is known, the optimal path from s to t is $\boldsymbol{P_J}$ as specified in (3). This path is designated as $OP(t)$. The values of J, P_J and R_J for a given source s depend on the destination t. Therefore, we designate J as given by (10) as $J(t)$ and the corresponding minimised maximum congestion level value R_J as $f(t)$. Then we rewrite (10) and (12) as,

$$J(t) = \underset{j\in\{1:M\}}{argmin}(R_j) \tag{13}$$

$$f(t) = R_{J(t)} = min([R_1, R_2, \dots, R_M]) \tag{14}$$

That is, $f(t)$ can be written as,

$$f(t) = \min_{j\in\{1:M\}}(R_j) = \min(\boldsymbol{R}) \tag{15}$$

When we select the optimal path *OP(t),* the relatively higher congestion level nodes are avoided while travelling from s to t. *The variable f(t) from (15), represents the maximum congestion level of path OP(t) from s to t.*

3.7. Objective

The objective is to determine $f(t)$ and the optimal path $OP(t)$ for a given s and for all t's in ($t\in\{1{:}N\}\backslash s$) . We designate this path as the Congestion Bottleneck Node Avoid Path (CBNAP) and designate the method to determine CBNAP as the CBNAP algorithm.

4. DETERMINATION OF CBNAP

4.1. Exhaustive search method

In general, for a given WSN, by knowing its topology, we can enumerate all possible paths from a given source node *s* to a destination node t. Along each path, we can find that node which has the highest congestion level among all the nodes of that path. This gives the maximum congestion level of that path. After determining the maximum congestion level of each path from *s* to *t*, we can select that path which has the least maximum congestion level value. But this method is NP hard, because the number of possible paths increases exponentially as *N* increases. Therefore, the dynamic programming approach is adopted to solve this problem.

4.2. Dynamic programming approach

As usual, the source node is denoted by *s*. Let *t* be any other node reachable from *s* with $OP(t)$ as the optimal path from *s* to *t* and $f(t)$ as the minimized maximum congestion level value of that path. Let $OP(t) = [\ v_0, v_1, \ldots, v_n]$ where $v_0 = s$ and $v_n = t$. Then, by knowing $f(v_0)$, the *Dynamic Programming* method solves, $f(v_1)$ in terms of $f(v_0)$, $f(v_2)$ in terms of $f(v_1), \ldots, f(v_n)$ in terms of $f(v_{n-1})$. The sub-optimal problems are solved recursively.

4.2.1. Effective congestion at source s

Whatever the actual congestion level at s, all the paths have to start from s. There is no other option. The congestion level Q_s is common for all paths starting from s. Therefore, for the purpose comparison and calculation of the congestion levels of the paths, effective Q_s is set to zero as,

$$Q_s = 0 \tag{16}$$

The minimized maximum value of Q_s is also 0. Therefore the corresponding $f(s) = 0$.

4.2.2. f(t) values for a one-hop neighbours of s

One hop neighbours of *s* are shown in Figure 2. Here, $t_1, t_2, \ldots, t_K$ are the one hop neighbours of *s*.

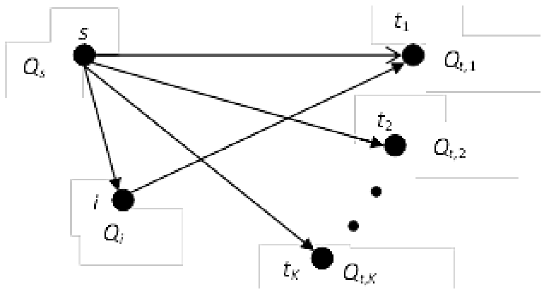

Figure 2. One hop neighbours of *s*

Since, t_1 is directly connected to s, path (s, t_1) is a single link (single hop) path. The minimum as well as the maximum congestion level of path (s, t_1) is Q_s itself which is zero as given by (16). Therefore,

$$f(t_1) = Q_s = 0 \tag{17}$$

This relation holds good even when we have a two hop path from s to t_1 through an intermediate node *i* as shown in Figure 4. Here, we have two paths (s, t_1) and (s, i, t_1). The corresponding maximum congestion levels are. For path $P_1 = (s, t_1)$, $R_1 = max(Q_s) = Q_s = 0$. For path $P_2 = (s, i, t_1)$, $R_2 = \max([Q_s, Q_i]) = \max(0, Q_i) = Q_i$. The minimised maximum value $f(t_1)$ is,

$$f(t_1) = \min(R_1, R_2) = \min(0, Q_i)) = 0 \tag{18}$$

The (18) holds good even when there are additional multi-hop paths to t_1 from *s*. Similar relation holds good for $t_2, t_3, \ldots, t_K$ which have one hop connectivity with s, as

$$f(t_2) = f(t_3) = \ldots = f(t_K) = 0 \tag{19}$$

The (18) and (19) can be combined to state an important property of *f*(.) as follows.
Property 1: When a node *i* belongs to the one hop neighbour set of *s*, then,

$$f(i) = 0 \text{ for } i \in \text{one hop neighbours of } s \tag{20}$$

Thus, $f(i)$'s of one hop neighbours of s are directly calculated using (20). Let the one hop neighbours of s be grouped into a set designated as $\boldsymbol{A_0}$. That is,

$$\boldsymbol{A_0} = \{\text{one hop neighbours of } s\} \tag{21}$$

Then, for i ∈ $\boldsymbol{A_0}$, the values $f(i)$'s are 0 as given by (20). Starting from the known values of $f(i)$'s (for $i \in \boldsymbol{A_0}$), the values of $f(i)$'s of other nodes (i $\notin \boldsymbol{A_0}$) are calculated using the principle of dynamic programming.

4.3. Calculation of f(t) by dynamic programming for any t

All the nodes of the network are grouped into two disjoint sets designated as $\boldsymbol{A}$ and $\boldsymbol{B}$. Set $\boldsymbol{A}$ holds those nodes whose $f(i)$'s have been already calculated and are known. Thus the optimal paths OP(i)'s are known for i ∈ $\boldsymbol{A}$. Nodes in set $\boldsymbol{B}$ holds those nodes whose $f(i)$'s are not known and yet to be calculated. Therefore $\boldsymbol{B} = \{[1{:}N] - \boldsymbol{A}\}$

$$\boldsymbol{A} = \{\text{Nodes whose f(i)'s have already been calculated and known}\} \tag{22}$$

$$\boldsymbol{B} = \{\text{Nodes whose f(i)'s have yet to be calculated and to be refined}\} \tag{23}$$

Unknown and uncalculated $f(i)$'s are initialized to ∞ so that they are excluded while calculating the minimum as explained later.

4.3.1. Initialization of f(i)'s

The calculation of $f(i)$'s for all i's is an iterative process. Initially, the one hop nodes of s are calculated to get $\boldsymbol{A_0}$. The $f(i)$'s for $i \in \boldsymbol{A_0}$, are set to 0 and $f(i)$'s for $i \notin \boldsymbol{A_0}$, are set to ∞. These are the initial values of $f(i)$'s. Initialization operations can be called as iteration 0. For the first iteration, the previous iteration is taken as iteration 0. The values of f(i)'s for iteration 0 are the initial values which are already known. In the first iteration, consider a target node t that belongs to $\boldsymbol{B}$. Now $f(t)$ is ∞. Let M be the number one hop neighbour nodes of t which are also members of set A. Let these nodes be denoted by $i_1, i_2, \ldots, i_M$ as shown in Figure 3. That is $i_k \in \boldsymbol{A}$ and $e(i_k, t) = 1$. If M = 0, then the next node from set B is taken as t and again M is determined. This process is repeated until M becomes greater than zero. In general these neighbour nodes will be all around t. But for the purpose of ease of explanation, they are shown in a single column to the left of t. Congestion level $Q_{i,k}$ of i_k are also marked in Figure 3 for k = 1 to M.

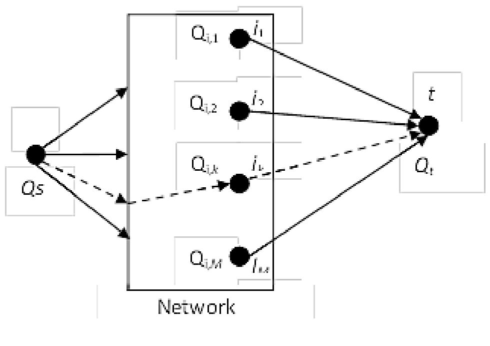

Figure 3. Multi hop paths for node *t*

In Figure 3, consider the path $[s - i_k - t]$. It is made up of $[s - i_k]$ in cascade with $[i_k - t]$. Here, path $[s - i_k]$ is the optimal path, because $f(i_k)$ has already been calculated and is known. $f(i_k)$ Gives the maximum congestion along $[s - i_k]$. The congestion level contributed from node i_k to path $[i_k - t]$ is $Q_{i,k}$. Therefore, the overall maximum congestion level along the path $[s - i_k - t]$, is given by,

$$\max[s - i_k - t] = R(i_k) = max\big(f(i_k), Q_{i,k}\big) \tag{24}$$

Here all i_k's belong to $\boldsymbol{A}$. Using (24), $R(i_k)'s$ are calculated for all i_k's for k = 1 to M. Then, the minimized maximum for this path, in the present iteration is represented by g(t) and it is calculated as,

$$g(t) = \min_{\mathrm{k=1:M}} \left(R(i_k)\right) \tag{25}$$

Substituting for $R(i_k)$ in (25) rom (24),

$$g(t) = \min_{\mathrm{k=1:M}} \left(max\left(f(i_k), Q_{i,k} \right)\right) \tag{26}$$

Once $g(t)$ is calculated, it is compared with the existing value of $f(t)$ (from the previous iteration) and the present $f(t)$ is updated only if g(t) is less than f(t). That is,

$$f(t) = \begin{cases} g(t) & \text{if } g(t) < f(t), \\ f(t) & \text{otherwise} \end{cases} \tag{27}$$

Now t is removed from $\boldsymbol{B}$ and added to $\boldsymbol{A}$. Set $\boldsymbol{A}$ grows and $\boldsymbol{B}$ contracts. Now next t from B is taken up and $f(t)$ for this t is updated as given by (27). Once all the elements in $\boldsymbol{B}$ are covered, ($\boldsymbol{B}$ goes empty), the present iteration is over and the next iteration starts. In the next iteration, same process as in first iteration repeats but with the updated set of $f(i)$'s.

4.3.2. Stopping criterion

During successive iterations, $f(t)$'s are updated according to (27). To express this in a compact form, let the collection of all $f(t)$'s for t = 1 to N for a given s, be represented by the array $\boldsymbol{F}$ as,

$$\boldsymbol{F} = [f(1), f(2), \dots, f(N)] \tag{28}$$

Then, $\boldsymbol{F}$ is updated in successive iterations. The theoretical maximum number of iterations is (N−1) [15]. In practice, if $\boldsymbol{F}$ does not change from the previous to the present iteration, then F will not change in further iterations too. After this, the process can break out of the iteration loop and there is no need to complete the (N−1) theoretical iterations. To facilitate the termination of the iterations, the value of $\boldsymbol{F}$ at the end of the previous iteration is stored in $\boldsymbol{F}_{\text{old}}$. At the end of the present iteration, the updated $\boldsymbol{F}$ is compared to $\boldsymbol{F}_{\text{old}}$. If $\boldsymbol{F} = \boldsymbol{F}_{\text{old}}$, the iteration loop is terminated.

The optimal path is obtained using the predecessor vector *pred* of size N as usual [15]. Determination of $f(t)$'s and the *pred* vector is described in Algorithm 1. This is basically a centralized algorithm. But can be converted into its equivalent distributed algorithm. The algorithm is a modification of Bellman-Ford shortest path algorithm [15].

Algorithm CBNAP

```
Inputs:   $N$ = No. of nodes in the WSN. $\boldsymbol{E}$ = Edge connectivity (Adjacency) matrix.
          $\boldsymbol{Q}$ = Congestion level vector for all nodes.   $s$ = Source node.
Outputs:  $OP(t)$ = Optimal path from $s$ to, $t$ for $t$ =1 to $N$
          $f(t)$ = Minimized maximum congestion level of path $OP(t)$ for $t$ = 1 to $N$.
 1. Initialize all $f(t)$'s to $\infty$ and pred($t$)'s to 0 as,
    For $t$ = 1 to $N$, $f(t) = \infty$; pred($t$) = 0; Endfor $t$
 2. Set $f(s) = 0$ ; Take $\boldsymbol{A}_0$ = [s]   //start with
 3. Get the one hop neighbours of $s$ and calculate $f(t)$'s for them as,
    For $t$ = 1 to $N$
        If $e(s, t)$ =1,  $f(t) = 0$; pred($t$) = s; $\boldsymbol{A}_0 = [\boldsymbol{A}_0, t]$; Endif
    Endfor t   //Set $\boldsymbol{A}_0$ is ready
 4. Take set A = A0,  Take B = [1:N]−A
 5. Store f(t)'s for t ∈ {1: N} in array F as, F = [f(1), f(2), …, f(N) ]
    // Initialization over. First iteration starts.
    Set $\boldsymbol{F}_{\text{old}} = \boldsymbol{F}$      //save $\boldsymbol{F}$ in $\boldsymbol{F}_{\text{old}}$.
 6. For h = 1 to N−1  //first iteration.  //h is the iteration count.
    For each $t$ in $\boldsymbol{B}$  while $\boldsymbol{B}$ is not empty
        For $i$ = 1 to $N$  // Neighbour nodes of $t$
        $R(i) = \infty$  //initial value
```

```
          if ( $i == t$) continue; if ( $e(i , t) == \infty$) continue ;
          calculate $R(i)$ from (24) as, $R(i) = max(f(i), Q_i)$
          end for $i$
    7.  Find the minimum of the array R as,
        [ $g(t)$, $index$] = $min(\boldsymbol{R})$;  If $g(t) == \infty$  continue endif
    8.  if  g(t) < f(t),   f(t) = g(t); pred(t) = index; endif
        endfor $t$
    9.  Get the updated array F for t =1 to N as,  F = [f(1), f(2), …, f(N) ];
        If $\boldsymbol{F} == \boldsymbol{F}_{old}$ Break (Go to 11) endif
    10. Store F in Fold for the comparison in the next iteration as, Fold = F;
        Endfor $h$   //end of $h$ loop
    11. Over
```

Once *pred*(*t*) is ready for *t* =1 to *N*, the corresponding *OP*(*t*)'s are easily obtained [15] using the function **get_TS(…)** as given below.
function TS = get_TS(pred, t, s)

```
TS=[t]; while t~=s, TS=[TS,pred(t)];t=pred(t);end
```

Vector *TS* gives the path from *t* to *s*. Path *OP*(*t*) which is the path from s to t is obtained by reversing the sequence *TS*. Algorithm CBNAP in association with function **get_TS(…)** gives *f*(*t*)'s and the Congestion Bottleneck Node Avoid Paths, *OP*(*t*)'s, from *s* to all other nodes. Once *f*(*t*)'s are determined for *t* = 1 to *N* for a given *s*, those high congestion level nodes whose *Q*'s are greater than *max*(***F***) are excluded from participating as intermediate nodes in the routing process in the present session. Thus these bottleneck nodes are avoided acting as *intermediate nodes* during the discovery of the optimal path. Here, OP(t)'s are the optimal paths from BS to sensors and the reverse of OP(t)'s provide the optimal paths from each sensor to BS.

Example 1: A network with 10 nodes, is represented as an undirected graph as shown in Figure 4. Nodes are shown in black. Congestion levels at nodes are shown in red. The node locations in Figure 4 are not exact physical locations. They are just representative for visual identification. The connectivity among nodes is represented by the connected links. In Figure 4, *s* =1. The present congestion levels, with Q_s= 0, are given as, Q =[0,70,11,110,15,40,80,2,30,12];

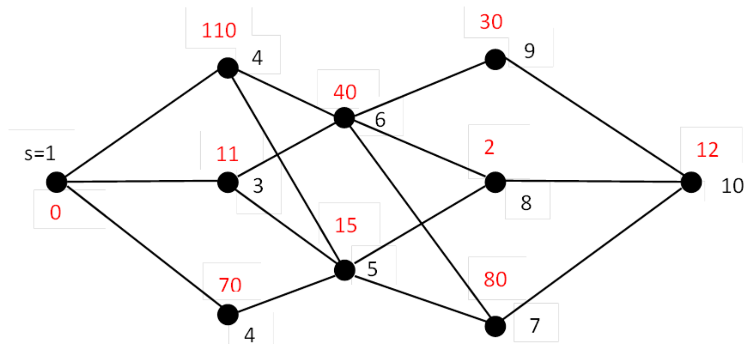

Figure 4. Graph layout for example 2

After running CBNAP, the resulting *f*(*t*) and *pred*(*t*) values are shown in Table 1, for *t* = 1 to 10. For lack of space, column heading Up date is represented by Ud and the variable *pred*(*t*) by *P*(*t*) in Table 1. We can see the updated values of *f*(*t*) after each update.

After update 9, the ***F*** vector is same as that of update 8. That is, $\boldsymbol{F} = \boldsymbol{F}_{old}$ and then the process converges. In this example, the main outer loop starting at step 8 of CBNAP algorithm terminates after 3 iterations. Here, max (F) = 15 and the bottleneck nodes having Q values greater than 15 are nodes 2, 4, 6, 7, 9. These nodes are excluded from acting as intermediate nodes in any optimal path originating from s. From Table 2, it can be clearly seen that nodes 2, 4, 6, 7, 9 are absent as intermediate nodes in all the optimal paths.

Table 1. Values of $P(t)$ and $f(t)$ after successive updates

Ud	t	1	2	3	4	5	6	7	8	9	10
1	$P(t)$	0	1	1	1	-	-	-	-	-	-
	$f(t)$	0	0	0	0	∞	∞	∞	∞	∞	∞
2	$P(t)$	0	1	1	1	3	-	-	-	-	-
	$f(t)$	0	0	0	0	11	∞	∞	∞	∞	∞
3	$P(t)$	0	1	1	1	3	3	-	-	-	-
	$f(t)$	0	0	0	0	11	11	∞	∞	∞	∞
4	$P(t)$	0	1	1	1	3	3	5	-	-	-
	$f(t)$	0	0	0	0	11	11	15	∞	∞	∞
5	$P(t)$	0	1	1	1	3	3	5	5	-	-
	$f(t)$	0	0	0	0	11	11	15	15	∞	∞
6	$P(t)$	0	1	1	1	3	3	5	5	6	-
	$f(t)$	0	0	0	0	11	11	15	15	40	∞
7	$P(t)$	0	1	1	1	3	3	5	5	6	8
	$f(t)$	0	0	0	0	11	11	15	15	40	15
8	$P(t)$	0	1	1	1	3	3	5	5	10	8
	$f(t)$	0	0	0	0	11	11	15	15	15	15
9	$P(t)$	0	1	1	1	3	3	5	5	10	8
	$f(t)$	0	0	0	0	11	11	15	15	15	15

Table 2. Optimal *OP(t)*'s and *f(t)*'s

s	t	Optimal path *OP(t)*	*f(t)*	*P(t)*
1	2	1→2	0	1
1	3	1→3	0	1
1	4	1→4	0	1
1	5	1→3→5	11	3
1	6	1→3→6	11	3
1	7	1→3→5→7	15	5
1	8	1→3→5→8	15	5
1	9	1→3→5→8 →10→9	15	10
1	10	1→3→5→8→10	15	8

5. COMPARISON WITH OTHER METHODS

Congestion avoid route can be determined by the simple 'GREEDY' method. The basic idea here is, starting from the source, to select the least congested neighbor node as the next forwarding node until the final destination is reached. Greedy method is invariably sub-optimal, because it will not foresee all possible alternatives. But it is fast. Another available solution is to use 'TARA' [9] to alleviate congestion in WSNs. Our Method CBNAP is compared to 'TARA' and 'GREEDY' methods.

5.1. Time complexity

The time complexity of CBNAP is O(N^3) [15]. The experimental completion time taken to get the optimum result for successive values of N is shown in the graph of Figure 5. Here, the number of edges and the adjacency matrix are randomly generated. The congestion level values of nodes are chosen using uniform random distribution. From Figure 5, it can be seen that, the time taken to generate the optimal solution increases as the number of nodes increases. The GREEDY method is faster compared to CBNAP while TARA is slower.The time saved in CBNAP is about 15-20% when the number of nodes is in the range 160-180.

5.2. Load Balance Index

Load balancing is an effective technique for congestion control [16-20]. CBNAP selects path with lower congestion levels. Therefore, when communication takes place using this path, the overall load balance improves because only the less congested nodes carry the present load. The fairness of load balance is measured using the *load balance index* (LBI) [16]. LBI is defined as,

$$\text{LBI} = \frac{\left(\sum_{i=1}^{N} Q_i\right)^2}{N * \sum_{i=1}^{N} Q_i^2} \qquad (29)$$

where Q_i is the congestion level at node i, for i = 1 to N.

When the loads are perfectly balanced (Q_i's are all equal) the LBI is one. On the other hand, LBI is low when the distribution of congestion levels is highly skewed (unbalanced). In the simulation experiment, the minimum congestion level is kept constant in each trial. The Maximum Congestion Level (MCL) of the network is successively increased in steps of 100 and the corresponding load balance indices are calculated for CBNAP, GREEDY and TARA algorithms. The MCL value represents the degree of unbalance of the pending traffic loads at nodes. The simulation result is shown in Figure 6. Here, the LBI's are very nearly same at lower values of MCL and LBI's diverge at higher values of MCL. From the plotted results, it can be seen that CBNAP provides better load balance. This is because CBNAP utilizes lower congested nodes for constructing the paths, even though the path length may be longer. Thus CBNAP achieves better LBI.

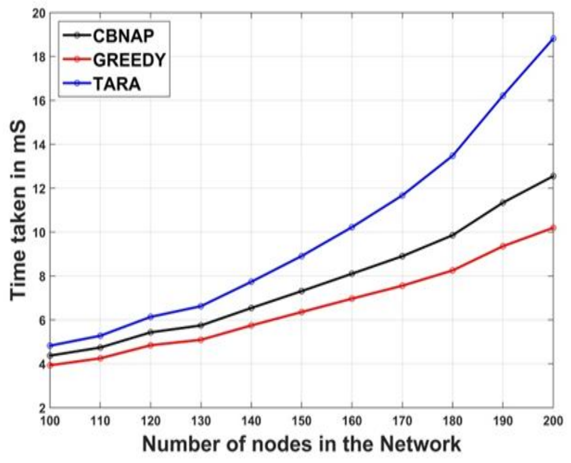

Figure 5. Execution time vs number of nodes

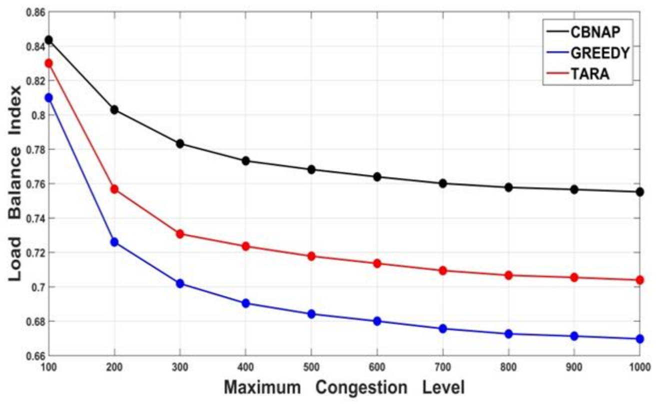

Figure 6. Load balance index vs maximum congestion level

6. CONCLUSIONS

A centralized algorithm has been described for finding the minimized maximum congestion level paths from a given source to every other destination. Those bottleneck nodes having higher congestion levels are excluded from acting as forwarding nodes. This centralized algorithm can be converted into an equivalent distributed algorithm that can be implemented at individual nodes. The proposed technique can be applied to determine the minimized maximum delay and maximum cost paths, maximized minimum remaining energy path and so on. This technique can be adopted by the vehicular traffic control system at metropolitan cities to avoid densely congested junctions for smooth flow of automotive traffic.

REFERENCES

[1] I. Akyildiz, et al., "A Survey on Sensor Networks," *IEEE Communications Magazine*, vol. 40, pp.102-114, 2002.
[2] R. Sharma and N. T. S. Kumar, "Review paper on wireless sensor networks," *Proc. of the Intl.Conf. on Recent Trends in Computing and Communication Engineering – RTCCE*, pp. 255-258, 2013.
[3] B. Hull, et al., "Mitigating congestion in wireless sensor networks," *International Conference on Embedded Networked Sensor Systems*, Maryland, 2004.

[4] T. E. Cheng and R. Bajcsy, “Congestion Control and Fairness for Many-to-One Routing in Sensor Networks,” *Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems*, 2004, pp. 148-161.
[5] H. Tall, et al., “M-CoLBA: Multichannel Collaborative Load Balancing Algorithm with queue overflow avoidance in WSNs,” *13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia*, 2017, pp. 2127-2133.
[6] Heikalabad S. R., et al., “DPCC: Dynamic Predictive Congestion Control in Wireless Sensor Networks,” *International Journal of Computer Science (IJCSI),* vol. 8, pp. 1694-0814, 2011.
[7] Q. Pang, et al., “Reliable data transport and congestion control in wireless sensor networks,” *Int. J. Sensor Networks,* vol. 3, pp. 16-24, 2008.
[8] N. Prabakaran, et al., “Rate Optimization Scheme for Node Level Congestion in Wireless Sensor Networks,” *Devices and Communications (ICDeCom), 2011 International Conference*, Mesra, 2011, pp. 1-5.
[9] J. Kang, et al., “TARA: Topology-Aware Resource Adaptation to Alleviate Congestion in Sensor Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 919-931, 2007.
[10] W. Fang, et al., “Congestion avoidance, detection and alleviation in wireless sensor networks,” *Journal of Zhejiang University-Science,* vol. C11, pp. 63-73, 2009.
[11] C. Y. Wan, et al., “CODA: congestion detection and avoidance in sensor networks,” *Proceedings of the 1st international conference on Embedded networked sensor systems,* ACM, 2003, pp. 266-279.
[12] L. Tao and F. Yu, “A novel congestion detection and avoidance algorithm for multiple class of traffic in a sensor network,” *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), IEEE International Conference on*, 2011, pp. 72-77.
[13] M. A. Kafi, et al., “Congestion control protocols in wireless sensor networks: A survey,” *Communications Surveys & Tutorials, IEEE,* vol. 16, pp. 1369-1390, 2014.
[14] S. A. Shah, et al., “Congestion control algorithms in wireless sensor networks: Trends and opportunities,” *Journal of King Saud University - Computer and Information Sciences*, 2016.
[15] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, New Delhi: Galgotia Publications, 1997.
[16] P. Hsiao, et al., “Load Balancing Routing for Wireless Access Networks,” *Proceedings IEEE Infocom 2001*, pp. 986-995, 2001.
[17] G. P. Sunitha, et al., “Optimized congestion aware energy efficient traffic load balancing scheme for routing in wireless sensor networks,” *International Conference on Information Processing (ICIP), Pune*, 2015, pp. 696-701.
[18] O. Chughtai, et al., “A congestion-aware and energy efficient traffic Load balancing Scheme for routing in WSNs,” *TENCON 2014 - 2014 IEEE Region 10 Conference*, Bangkok, 2014, pp. 1-6.
[19] C. Basaran, et al., “Hop-by-hop congestion control and load balancing in wireless sensor networks,” *IEEE Local Computer Network Conference,* Denver, CO, 2010, pp. 448-455.
[20] N. Goyal, et al., “Congestion control and load balancing for cluster based underwater wireless sensor networks,” *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Waknaghat, 2016, pp. 462-467.

BIOGRAPHIES OF AUTHORS

Sanu Thomas holds B.E. degree in Electronics & Communication and M.Tech. Degree in Computer Engineering from University of Mysore, India. Since 1998, he is working as Faculty in the Department of Computer Science, School of Technology and Applied Science, Kottayam, Kerala, India. He is currently pursuing the Ph.D. degree at School of Technology and Applied Sciences, Mahatma Gandhi University, Kottayam, Kerala, India. His area of research is Wireless Sensor Networks.

Dr. Thomaskutty Mathew received his Ph.D degree in Microwave Electronics from Cochin University of Science and Technology Cochin, India in 1997. From 1995 to 1999 he worked as a Lecturer in Physics at Christ College, Irinjalakuda, Kerala, India. Since 1999, he is working as Faculty in the Department of Electronics, School of Technology and Applied Science, Kochi, Kerala, India and presently working as Reader. During the period 20 06-20 08, he worked as a Post Doctoral Research Associate at Department of Electronics, University of Kent, Canterbury, U.K. He is a Rerearch Guide in Electronics, Mahatma Gandhi University, Kottayam, Kerala, India. His current area of research are : Microstrip Antennas, Radar Cross Section, RFID, Wireless Sensor Networks etc. He is a member of IEEE Antennas and propagation society and IET (U.K).