International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-3, January 2019

Shortest-Path Routing in Wireless Mesh Networks under Time Varying Link-Cost Metric

Nandini K. S, S.A. Hariprasad

Abstract*— *In a wireless Mesh Network, with cognitive radios, the cost of the available communication links can vary with respect to time due to the schedules of the primary users. Under this condition, to find the time dependent shortest paths for the secondary users is a challenging task. At a particular time slot, if no forward links are available at a node, the data packet which has just arrived have to halt (or wait) until the availability of one or more forward paths. Then the data packet suffers halting delay. Thus the overall arrival time could be more than that of an alternate path. Therefore, in this paper we provide an innovative method of reducing the halts along the path. We also have to minimize the length of the topological (physical) distance along the selected path. Thus, in our paper, two objectives, the earliest arrival time as well as minimum shortest topological path, are realized. The weight assigned to each objective can be adjusted according to the requirements. To solve this bi-objective optimization problem, we propose a new modification to the well-known Dijkstra shortest path algorithm which takes care of the actual link cost and also the halting cost.

Key words: Cognitive Radio Network, Spectrum sensing, Secondary users, Time dependent cost Matrices, Probability of Interference, Optimal Assignment

1. INTRODUCTION

In recent times, Cognitive Radio Networks (CRN's) have become very popular and useful as the demand for radio spectrum is increasing exponentially [1-2]. Application of Cognitive Radio principles to a Wireless Mesh Network (WMN) has several advantages [3-7]. Let us call such a network as Cognitive Radio Empowered WMN (CRE-WMN).

In general, a CRE-WMN has several Wireless Mesh Routers (WMR's) and Wireless Mesh Clients (WMC's). In this paper, we classify the users of the CRE-WMN into the primary users and the secondary users. The primary users have higher priority in availing and utilizing the network resources like spectrum, bandwidth, time-slots, connecting links, gateways, printers, *etc*. The secondary users have to wait for the availability of the resources whenever released by the primary users. Since the primary users have their own time schedules, the availability of the resources for the secondary users is time dependent and intermittent. In this paper, the time dependencies of the availability of communication links are modeled as the time varying link costs for the secondary users. Then, using these time varying link costs, we find the shortest multi-hop paths for the secondary users. Because of the intermittent nature of the availability of links to the secondary users, the packets sent by them may have to wait or halt at intermediate nodes. In this paper we take the halts at nodes into consideration and determine the optimal halt locations and the number of halts there in. This is our main contribution.

Manuscript published on 30 January 2019.
***Correspondence Author(s)**
Nandini K. S, Research Scholar, Jain University, Bangalore, India, Department of Electronics and Communication Engineering, RNS Institute of Technology, Bangalore, Karnataka, India.
S.A. Hariprasad, Director of School of Engineering and Technology, Jain University, Bangalore, Karnataka, India.

A. Related work

Several well-known Algorithms are available to determine the shortest paths in a fixed edge weight graph [8]. For time dependent edge weights, the earliest work was by Cooke and Halsey [9]. In [9], the authors use Bellman's dynamic programming method and there is no provision to find the exact path and the minimum halts (waiting) at nodes. Even though Klafszky [10] did introduce the halt times at nodes, these times were taken as the given inputs and not as the desired optimal outputs. Also, in [10], the problem is not solved directly, but an equivalent dual problem is formulated and solved. Dreyfus [11] and Halpern [12] have used algorithm similar to the dynamic programming method as in [9]. But they mainly solve the minimum total travel time or the earliest arrival time problem instead of shortest physical distance problem. The algorithms proposed in [9-12] hold good only for those cases where the First In First Out (FIFO) property is satisfied as shown by Kaufman and Smith [13]. In [14], time dependency is taken care by expanding the graph with respect to time. Orda and Rom [15] have discussed different restrictions on waiting at nodes and the algorithms presented in [15] mainly concentrate on the earliest arrival times. The physical shortest path which is also called as the *topological shortest path* in [15] is calculated indirectly. The algorithm in [16] solves the starting time selection problem in a time dependent graph. Continuous time model is fully covered in [17]. In [18] the authors have used the control system model to solve the earliest arrival path problem. In [19], the authors have studied the time dependent multimodal graph. In [20], modified Djikstra algorithm is described for time dependent graphs. Here, the author does not discuss the *halt at nodes* issue exclusively. In [21], modified Floyd-Warshall algorithm is described to determine the shortest, non-crossing, rectilinear paths in a two dimensional grid graph.

II. BASIC CRE-WMN MODEL, SYMBOLS AND NOTATIONS

The basic CRE-WMN is modeled as a square grid undirected graph for convenient presentation as shown in Fig.1. WMR's are the nodes (vertices) of the graph. The communication links between the nodes are the edges of the graph. For easy presentation, nodes are arranged in *H* rows and *W* columns. The total number of nodes, $N = W*H$.

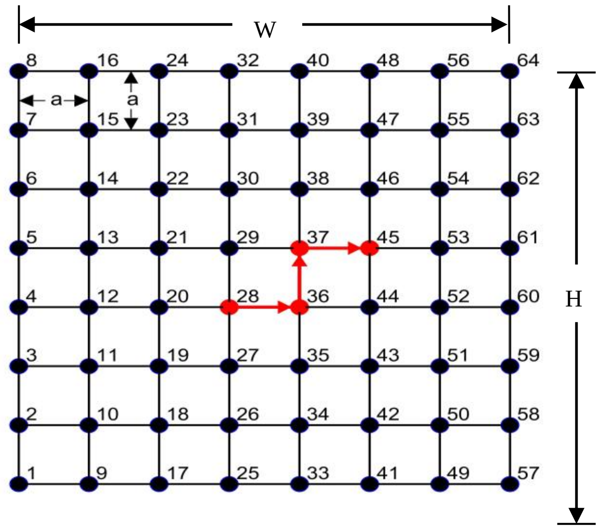

Fig.1. Square grid graph. Primary path

The horizontal as well as the vertical grid spacing is taken as ***a*** meters (say ***a*** = 50 meters) as marked in Fig. 1. The nodes are assumed to be homogeneous and the communication radius (range) of each node, represented by R_C, is taken as ***a***. Therefore the network is 4-connected. The interference radius for each node, represented by R_I, is taken as $\sqrt{2} * \boldsymbol{a}$ so that the immediate diagonal nodes and the adjacent nodes are within the interference range of one another. Thus we have,

$$\text{Communication Range} = R_C = \boldsymbol{a} \quad (1)$$

$$\text{Interference Range} = R_I = \sqrt{2} * \boldsymbol{a} \quad (2)$$

A. Initial Cost (or Connectivity) Matrix C

The cost or the connectivity matrix ***C*** gives the connectivity between the nodes of the network. In our scheme the node connectivity changes with time. The initial connectivity status is given by matrix ***C***. The size of ***C*** is *N*x*N* where *N* is the total number of nodes. The element *c*(*u*, *v*) of ***C*** represents the connectivity between node *u* and node *v* (or edge connectivity) as,

$$c(u,v) = \begin{cases} 0, & \text{if } u = v \\ 1, & \text{if } u \neq v \text{ and } u, v \text{ are connected} \\ \infty, & \text{if } u \neq v \text{ and } u, v \text{ are not connected} \end{cases}$$

(3)

The value ∞ means no connection. When node u and v are same, the cost of going from u to v is zero. That is the hop count from *u* to *u* itself is zero. When *v* is different from *u*, node *u* and *v* are connected if the Euclidean distance between them is less than or equal to the communication radius R_C. Therefore (3) can be rewritten as,

$$(u,v) = \begin{cases} 0, & \text{if } u = v \\ 1, & \text{if } u \neq v \text{ and } d(u,v) \leq R_C \\ \infty, & \text{if } u \neq v \text{ and } d(u,v) > R_C \end{cases} \quad (4)$$

Here, *d*(*u*, *v*) is the Euclidean distance between node *u* and *v*. The basic connectivity matrix ***C*** depends on the geographical locations of the nodes and R_C. Since the links are bidirectional, *c*(*u*, *v*) = c(*v*, *u*). Thus ***C*** is symmetric. When *c*(*u*, *v*) = 1, it means a packet can be transmitted between *u* and *v* in a single hop. The matrix ***C*** represents the initial value of the connectivity which is available for the primary user. The connectivity matrix ***C*** is also the cost matrix. *c*(*u*, *v*) gives the cost of the link or edge between node *u* and *v*. Thus the cost of link (*u*−*v*) is *c*(*u*, *v*). Therefore *c*(*u*, *v*)'s would have been used in determining the minimum cost paths if the link costs were static or unchanging with time. For the primary user, the link costs are static. Therefore c(*u*, *v*)'s are used to determine the minimum cost path or the shortest path for the primary or the first user.

The basic feature of our mesh network is *time varying link cost* for the secondary user which will be discussed later.

B. Primary User

In our scheme, nodes are basically mesh routers which can be deployed for either primary or secondary communication. Here we have one primary user who has the highest priority and the secondary users have to wait for the availability of free (unaffected by the primary user) nodes and links. Nodes directly used by the primary user and also those nodes interfering the primary user are the forbidden nodes. For example, in Fig. 1, we have 64 nodes and the nodes are numbered from 1 to 64. The primary usage is taken as a multi-hop point to point communication from node 28 to 45 in 3 hops along the path [28−36−37−45].

C. Discretization of Time

The communication time is discretized into uniformly spaced time points as *t*(0), *t*(1) = t(0)+1, t(2) = t(0)+2,…, *t*(*k*) = t(0)+k,…*etc*. as shown in Fig. 2. For the convenience of narration, the magnitude of the successive time intervals are set to 1 (in appropriate time unit). That is, we take *t*(*k*) − *t*(*k*−1) = 1 for *k* = 0, 1, 2, …, etc. We also set *t*(0) to 0. Now the time variable *t* is discretized as *t* ={0, 1, 2, …, *k*,…, *kmax*} where *kmax* is the maximum time that covers the problem under consideration. Thus, in this paper, time points are integer values in the range 0 to *kmax* and *t(k)* = *k*.

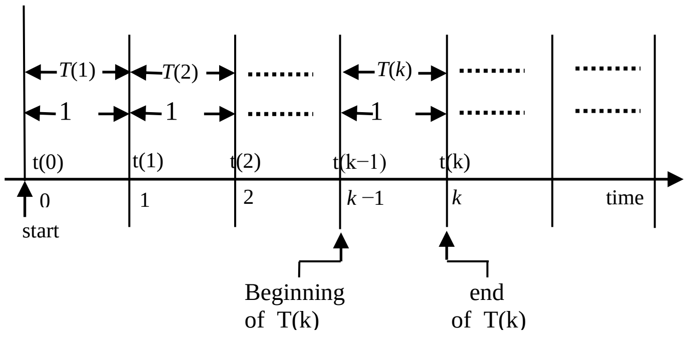

Fig. 2. Discretization of time and time

Successive time slots are labeled by the sequence $T(1)$, $T(2)$,…, $T(k)$,…*etc.*, as shown in Fig. 2. Here, $T(k)$ is the k^{th} time slot and k is the time slot index. T(k) represents the time interval from time instant $(k-1)$ to time instant k. Note that, the time slot index k of T(k) is the value of t(k). An important feature of time slot T(k) is, the beginning of T(k) is $(k-1)$ and the end of T(k) is k.

The time required by the primary user to complete one hop of communication is taken as the duration of one time slot which is taken as 1 (in appropriate time unit). For example, for 3 primary hops, we need 3 time slots which is equal to 3.

For the convenience of demonstrating the basic principle of our method, the primary transmission is taken as periodic (cyclic) in nature. In the example of Fig. 1, the primary communication in the first cycle is,

$$\left.\begin{array}{l}[28-36]\ \text{--- hop(1) — time slot } T(1)\\ [36-37]\ \text{--- hop(2) — time slot } T(2)\\ [37-45]\ \text{--- hop(3) — time slot } T(3)\end{array}\right]\quad \text{Cycle(1)}$$

The primary communication pattern repeats the same path in the next cycle as,

$$\left.\begin{array}{l}[28-36]\ \text{--- hop(1)—time slot } T(4)\\ [36-37]\ \text{--- hop(2)—time slot } T(5)\\ [37-45]\ \text{--- hop(3)—time slot } T(6)\end{array}\right]\quad \text{Cycle(2)}$$

This pattern continues in successive cycles. The period of the primary cycle is represented by P. In this example P =3. The time schedule for the primary user is shown in Fig. 3.

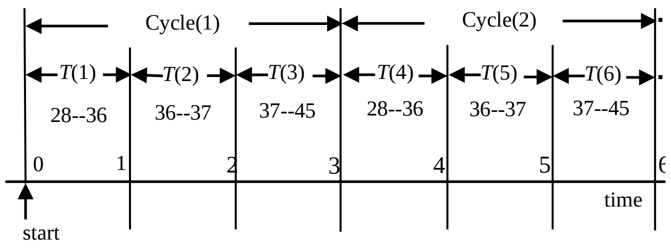

Fig. 3. Time schedule for the primary user

D. Cost of Links During Successive Time Slots

During successive time slots, the primary user occupies different links. Then these and associated links are not available to the secondary users. Then the cost of these links change over time. Now the link cost is a function of time. The symbol $cost(u, v, k)$ is used to represent the cost (or weight) of link from node u to v in time slot $T(k)$ for k = 1, 2,.., and so on. In general, for a bidirectional link $(u-v)$, the value of $cost(v, u, k)$ is same as that of $cost(u, v, k)$.

E. Forbidden nodes and links.

In a single channel wireless network, when the primary user is communicating over a link between two one hop neighbouring nodes in a specific time slot, those nodes which are within the *interference range* of the communicating nodes are designated as the *forbidden nodes* during that time slot. Forbidden nodes should not participate in any other communication during that time slot. Those links incident on the forbidden nodes are called *forbidden links* because these links are connected to the forbidden nodes.

Let us find the forbidden nodes and links for the primary communication shown in Fig. 1.

Forbidden nodes during T(1): In the example of Fig. 1, during T(1), data transmission takes place from node 28 to node 36. Therefore the forbidden (interfering) nodes which are within the interference range of node 28 and 36 are shown in Fig. 4, in cyan. (The radii of the circles are taken slightly larger than R_I to fully include the forbidden nodes at the circular border). Let us represent the set of forbidden (interfering) nodes of node 28 by **F**(28). Then,

$$\boldsymbol{F}(28) = [19, 20, 21, 27, 28, 29, 35, 36, 37]$$

Similarly,

$$\boldsymbol{F}(36) = [27, 28, 29, 35, 36, 37, 43, 44, 45]$$

Now, during $T(1)$, the forbidden node set for the link (28−36) is the union of $\boldsymbol{F}(28)$ and $\boldsymbol{F}(36)$ represented by $\boldsymbol{G}(1)$ as,

$$\boldsymbol{G}(1) = \boldsymbol{F}(28) \cup \boldsymbol{F}(36)$$
$$= [19, 20, 21, 27, 28, 29, 35, 36, 37, 43, 44, 45]$$

Nodes of G(1) are shown in cyan in Fig. 3.

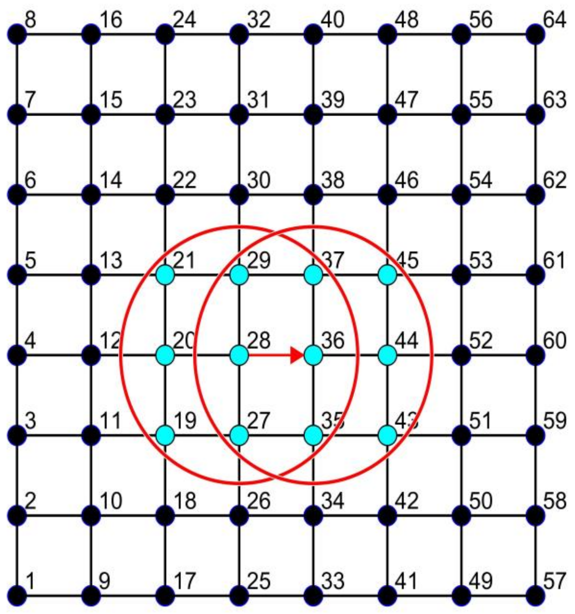

Fig. 4. Forbidden Nodes during *T*(1)

1.Forbidden Links during T(1): The forbidden links are those which incident on the forbidden nodes. The forbidden nodes as well as the forbidden links due to the communication from node 28 to 36 in $T(1)$ are shown in cyan and red in Fig. 3. (Nodes 28, 36 and the link(28−36) are also forbidden). The secondary users cannot use these nodes/links during T(1). The available nodes/links for the secondary users are shown in black.

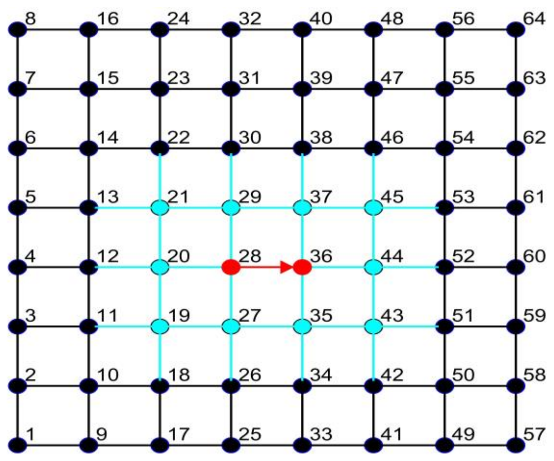

Fig.5. Forbidden Nodes and Links during *T*(1)

In general, in time slot $T(k)$, when the primary communication is from node $u(k)$ to its one hop neighbour $v(k)$, the forbidden node set $\mathrm{F}(\mathrm{u}(\mathrm{k}))$ is given by those nodes whose distances are less than or equal to the interference range of $u(k)$ as,

$$\mathrm{F}(\mathrm{u}(\mathrm{k})) = \{ j \mid d(u(k), j) \le R_{\mathrm{I}} \text{ and } j \in [1{:}N] \} \tag{5}$$

Here, $d(u(k), j)$ is the distance from $u(k)$ to j.

Similarly, for node $v(k)$,

$$\mathrm{F}(\mathrm{v}(\mathrm{k})) = \{ j \mid d(v(k), j) \le R_{\mathrm{I}} \text{ and } j \in [1{:}N] \} \tag{6}$$

In time slot $T(k)$, for the primary communication link $(u(k) - v(k))$, the forbidden node set is represented by $G(k)$ which is given by combining the nodes given by (5) and (6) as,

$$G(k) = \mathrm{F}(\mathrm{u}(\mathrm{k})) \cup \mathrm{F}(\mathrm{v}(\mathrm{k})) \tag{7}$$

2. Forbidding or disabling a link during T(k)

The basic principle of forbidding or disabling a link is to set its cost (or its effective weight) to ∞. Then the shortest (or minimum cost) path algorithm for a secondary user avoids these links. Thus the interference between the primary user and the secondary user is avoided.

Setting the cost of forbidden links to Infinity: Consider the time slot $T(k)$. Let g be a forbidden node which belongs to $\boldsymbol{G}(k)$. The forbidden links of node g are the incident edges of node g from its neighbours. $cost(g, h, k)$ gives the cost of the outgoing link $(g-h)$ during $T(k)$. Let $\boldsymbol{H}(g)$ be the set of neighbour nodes of g. Then the out going links of g are $(g-h)$'s for $h \in \boldsymbol{H}(g)$. Therefore, in time slot $T(k)$, the cost of the outgoing links of g for all g's belonging to $\boldsymbol{G}(k)$ are set to ∞ as,

$$cost(g, h, k) = \infty \quad \text{for } h \in \boldsymbol{H}(g) \text{ and for } g \in \boldsymbol{G}(k) \tag{8}$$

Assignment (8) disables or effectively disconnects all the outgoing links of g during $T(k)$. Similarly, all the incoming links of g from $\boldsymbol{H}(g)$ are disabled by the assignment,

$$cost(h, g, k) = \infty \quad \text{for } h \in \boldsymbol{H}(g) \text{ and for } g \in \boldsymbol{G}(k) \tag{9}$$

Getting the neighbor nodes of g: Let h be a neighbor node of g. Then h is one hop connected to g. Therefore, from (3) we see that $c(g, h)$ =1. Since matrix $\boldsymbol{C}$ is known, we can find those h values (column indices) which satisfy $c(g, h) = 1$. Using Matlab notation, we can find the set H(g) as,

$$H(g) = find(\boldsymbol{C}(g,:) == 1) \tag{10}$$

Here, $\boldsymbol{C}(g, :)$ represents row g of matrix $\boldsymbol{C}$.

Calculation of overall cost(u, v, k)

For determining the shortest path from a source to a destination, we should know the cost of all the links. Thus cost(u, v, k) should be known for $u \in \{1{:}N\}$ and $v \in \{1{:}N\}$. Then all the links are covered.

When the primary user is absent, all the links are available for the secondary user for all the time slots. Constraints (8) and (9) do not apply to the link costs. Then, $cost(u, v, k)$ is independent of k and *it* is governed by the link connectivity relation as given by (4). Then $cost(u, v, k)$ can be calculated for $u \in \{1{:}N\}$, $v \in \{1{:}N\}$ and all k's as,

$$cost(u, v, k) = c(u, v) \tag{11}$$

For $u = v$, from (3) we know that $c(u, v) = 0$. Therefore, $cost(u, v, k)$ is also zero when $u = v$. Equation (11) gives the unconstrained $cost(u, v, k)$ which is independent of k. Then the constrained $cost(u, v, k)$ which is time dependent according to the primary usage, is obtained by implementing the assignment (8) and (9). The algorithm to get the time dependent $cost(u, v, k)$ is given below.

Algorithm *get_ cost*(u, v, k).

```
Inputs: Matrix C, Value of k, Link $(u(k) - v(k))$ occupied
        by the primary user in time slot $T(k)$.
Output: Constrained $cost(u, v, k)$ for $u \in \{1{:}N\}$, $v \in \{1{:}N\}$
1. Get unconstrained $cost(u, v, k)$ using (11) as,
        for $u \in \{1{:}N\}$, $v \in \{1{:}N\}$
     cost(u,v,k) = c(u,v)
2. Get $\boldsymbol{G}(k)$ using (7).
3. For each node g in $\boldsymbol{G}(k)$
     Find H(g) using (10).
     For each $h$ in $\boldsymbol{H}(g)$
         Set $cost(g, h, k) = \infty$ .
         Set $cost(h, g, k) = \infty$.
     Endfor
   Endfor
4. Constrained $cost(u, v, k)$ is ready.
```

From algorithm *get_ cost*(u, v, k), we see that $cost(u, u, k)$ is 0, 1 or ∞. Equality $cost(u, v, k) = 0$, which occurs when $v = u$, means there is zero hop between v and u. When $cost(u, v, k) = 1$, it means there is a 1 hop connectivity (cost) between node u and v. When $cost(u, v, k) = \infty$, it means there is no connectivity between node u and v.

Periodicity of cost(u, v, k) : Since the primary traffic is periodic with period P, the formation of forbidden links also repeats periodically at period P. Therefore,

$$cost(u, v, k) = cost(u, v, 1 + mod(k-1, P)) \tag{12}$$

for $k = 1, 2, \ldots, etc$. Here, $cost(u, v, q+n*P) = cost(u, v, q)$ for q = 1, 2, …, P and n = 0, 1, 2,…*etc*.

Size of the cost matrix: In the link cost term $cost(u, v, q)$, u and v vary from 1 to N and q is varying from 1 to P. Therefore, the size of the ***cost*** matrix is NxNxP. This is the storage size requirement for the ***cost*** matrix.

III.TIME DEPENDENT DIJKSTRA ALGORITHM

In Time Invariant Dijkstra Algorithm (TIDA), the algorithm finds the shortest (or minimum cost) path from a given a single source node to all other nodes by successively updating *dist*(*u*)'s, the distances (total costs) of these nodes from the source. The update of *dist*(*u*) is done such that it is the minimum (shortest) total distance from the source. The basic well known TIDA (or conventional Dijkstra algorithm, without speedup) is presented here as a function so that we can explain the Time dependent Dikstra Algorithm (TDDA) with reference to TIDA. Cost matrix *cost* and the source *s* are the inputs.

Algorithm TIDA
```
Function [dist, prev] = dijkstra(cost,s)
  %Initialization
  for u = 1:N
     dist(u)  = ∞;    %from s to u
     prev(u)  = 0;   %undefined value
  end
  dist(s)=0;          %from s to s
  %Initialization over
  for j=1:N     %update loop begins
     for u=1:N       %cover all nodes
       if u==s      %except s
         continue;
       end
       for v = 1:N  %consider all
                    %intermediate nodes
         if dist(u)> dist(v)+cost(i, u)
            dist(u)  =  dist(v)+cost(i,
u);
                          %update        of
dist(u)
            prev(u) = v; %previous of u
         end
       end     %end of v-loop
     end       %end of u-loop
  end          %end of j-loop
```

Now, we introduce our TDDA which is a modification of Dijkstra Algorithm that takes care of Time Dependent link costs.

A. Total Cost to Reach Node u

In TDDA, the link cost is a function of time and therefore, the total cost to node *u* (total cost of travelling from source node *s* to node *u*) is also a function of time. In this paper, this time dependent cost is represented by *tot_cost*(*u*, *T*(*k*)). For simplicity *tot_cost*(*u*, *T*(*k*)) is written as *tot_cost*(*u*, *k*). In TDDA, *tot_cost*(*u*, *k*) is analogous to *tot_cost*(u) term of TIDA.

Definition of tot_cost(u, k): Consider the case of a data packet travelling from *s* to *u* in multiple hops along the shortest path. *tot_cost(u, k)* is defined as the total cost of the links along the path taken by the travelling packet while reaching node *u* during the the time slot *T*(*k*). Consider a trivial example of just two nodes *s* and *u* with *cost*(*s*, *u*, 1) = ∞ and *cost*(*s*, *u*, 2) = 1. Then, *tot_cost*(*u*, 1)= *cost*(*s*, *u*, 1) = ∞. When *cost*(*s*, *u*, 1) = ∞, the packet does not traverse the s to u link in *T*(1). In time slot *T*(2), *cost*(*s*, *u*, 2) = 1. Therefore, traversal from s to u takes place in *T*(2) and *tot_cost*(*u*, 2)= *cost*(*s*, *u*, 2) = 1. The update of distance *tot_cost*(*u*, *t*) takes place in successive time slots. The update of *tot_cost*(*u*, *t*) in the present time slot *T*(*k*) is based on the distance of intermediate nodes in the previous time slot *T*(*k*−1) and the cost of the connecting link in the present time slot *T*(*k*) as will be described later.

B.Previous or Predecessor array prev

In TDDA as well as in TIDA, when the minimum distances are updated along the estimated shortest path, the previous or the predecessor node of the present node *u* is stored in the array *prev*(*u*) as,

$$prev(u) = v \tag{13}$$

Given the destination node *d*, we start with *d* and successively get back the preceding nodes until we reach the source node *s*. The well-known topological shortest path function is given below.

```
function path=get_path(prev,d,s)
  path=[];
  if prev(d)==0 % t is not accessible
    return;
  end
  i=1;
  path(i)=d;  %start with d
  %iterate until s is reached
  while path(i)~=s
    path(i+1)=prev(path(i));
    i=i+1;
  end
   % reverse the order of path
   % to get the forward path from s to d.
  path=seqreverse(path);
```

Inaccessibility of d: During the initialization process *prev*(*d*) has been set to 0 for all d's.. If *d* had been reached in any time slot, Equation (13) would have been satisfied and *prev*(*d*) would have been *v*, other than zero. Thus *prev*(*d*) remaining at 0 implies that *d* is not reachable from *s*. Thus the condition *prev*(*d*) = 0 implies *d* is inaccessible to *s* and the *path* is an empty set.

The definition and usage of *prev*(*u*) is same both in TIDA and TDDA. As *u* can vary from 1 to *N*, the size of array ***prev*** is 1x*N*.

C.Arrival Time Slot Index atsi

Consider the case of a data packet travelling from *s* to *d* in multiple hops along the shortest path. In time dependent link cost network, the optimal (minimum) arrival time of the packet from node *v* to *u* depends on the link costs along the shortest path. The specific time slot at which the packet arrives at node u along the shortest path, for $1 \leq u \leq N$, is designated by the variable *atsi*(*u*). Here, the array ***atsi*** stands for Arrival Time Slot Index (***atsi***). For example, if the link cost from node *v* to *u* during *T*(*k*), represented by, *cost*(*v*, *u*, *k*) is ∞, then packet traversal from *v* to *u* does not happen.

Then arrival at u in time slot T(k) does not occur. It may occur sometime later. On the other hand if *cost*(*v*, *u*, *k*) = 1 and the total distance to *u* from the source *s* via node *v* is the shortest one, then the traversal occurs from v to u in T(k). Then, *atsi*(*u*) is set to *k*. The arrival time slot index ***atsi*** for the source node s is taken as zero, because the packet is already there at *s* at start. Therefore,

$$atsi(u) = 0 \;\; for\; u = s \qquad (14)$$

For other nodes *atsi*(*u*) is defined as,

$$atsi(u) = \begin{cases} k, & \text{if the packet arrives at } u \text{ in } T(k) \\ \infty, & \text{if the packet does not arrive at } u \text{ in } T(k) \end{cases} \quad ---- (15)$$

for *u*=1 to *N* except s. For some reason, for certain *u*, let *atsi*(*u*) = ∞ for the present *T*(*k*). This fact does not prevent *atsi*(*u*) from assuming a finite value in some future time slot say *T*(*k*+*n*) where *n* is a positive integer greater than zero. From (14) and (15), we see that *u* can take values from 1 to N. Therefore, ***atsi*** is an array of size 1x*N*.

D.Upper Limit on the number of time slots

In TIDA the update of `dist(u)` is completed within at most *N* iterations. In TDDA, the upper limit on the number of time slots to be used is represented by *kmax*. In TDDA, as given by (12), the same edge cost pattern repeats every *P* time slots. A path cost (sum of its edge costs) can take P dissimilar values in one cycle. But the path cost repeats after every P time slots. Therefore we choose *kmax* = *P***N* so that the unreachable nodes within N iterations get at most N similar cost pattern iterations. Thus we select *kmax* as,

$$kmax = P*N \qquad (16)$$

E. Initialization in Time Dependent Dijkstra Algorithm

Initially, the distance of the source node s (from itself) is set to zero and that of all other nodes are set to ∞ for k = 1 : k_{max}. For u = 1:*N*, array *prev*(*u*) is set to all zeros to mean undefined values. Array *atsi*(*s*) is set to 0 as given by (14). The initialization process is given as,

```
%Initialization in TDDA
  for u = 1:N
    for k = 1:kmax
       if u==s
         tot_cost(u,k) = 0; %from s to s
       else
         tot_cost(u,k) =∞;    %from s to u
       end
    end      %end of k-loop
    prev(u) = 0; %undefined values
    if u == s
      atsi(u)= 0  %to satisfy (13)
    else
      atsi(u)= ∞; %Initialization
    end   %end of if
  end    %end of u-loop
  %Initialization over
```

F. Non-increasing property of tot_cost(u, k)

In TDDA, *tot_cost*(u, k) represents the minimum total distance of node *u* from source *s* in time slot *T*(*k*). An important requirement of *tot_cost*(u, k) is that it should is a monotonically non-increasing function of *k*. In TDDA, *tot_cost*(u, k) is initially set to ∞ for all k's and all u's except u = s. Then *tot_cost*(u,k) is updated in the successive time sloys. At time slot T(k_1), let the updated value of *tot_cost(*u, k_1) be c_1 which is a finite value (less than ∞). Then for any $k_2 > k_1$, *tot_cost(*u, k_2) cannot be ∞ or any other value higher than c_1. This requirement is ensured by setting *tot_cost(u,* k_2) also to c_1 for all k_2's greater than k_1 as,

$$tot\_cost(u, k_2) = c_1 \quad \text{if} \;\; tot\_cost(u, k_1) = c_1 \qquad (17)$$

for all k_2's greater than k_1 up to *kmax*.

G.Assignments of in time slot T(1)

In TDDA, for time slot *T*(1), there is no previous time slot. Therefore, the assignments are direct without any intermediate nodes. We set *k* = 1 to identify *T*(1).

Assigning values for tot_cost(u, k)'s: With *k* = 1, we set *tot_cost*(*u*, *k*) = *cost*(*s*, *u* , *k*) for *u* = 1 to *N* only if *cost*(*s*, *u*, *k*) is finite (non-infinity). When *cost*(*s*, *u*, *k*) = ∞, assignment *tot_cost*(u, k) = *cost*(*s*, *u* , *k*) is reduntant because *tot_cost*(u, k) has already been set to ∞ in the initialization process of TDDA as described in section III. D.

Assigning values for ***prev*** *and* ***atsi*** *:* Consider any node *u* other than *s*. When *k* = 1 and *cost*(*s*, *u* , *k*) = 1, node *u* is directly connected to s. A packet from *s* can reach node *u* in a single hop Therefore the predecessor of u is s . Hence we set *prev*(*u*) = s if *cost*(*s*, *u* , *k*) = 1. On the other hand if *cost*(*s*, *u*, *k*) = ∞, node *u* is not connected to *s* directly and *prev*(*u*) remains at 0 (undefined) in *k* =1. similarly, when *k* = 1, the traversal takes place from node *s* to node *u* if *cost*(*s*, *u*, *k*) = 1 and the packet arrives at *u* in *T*(1). Therefore *atsi*(*u*) is set to 1. If *cost*(*s*, *u*, *k*) = ∞, the packet does not arrive at *u* in *T*(1). Therefore, *atsi*(*u*) remains at ∞.

Assignments of *tot_cost*(*u*, *k*), *prev*(*u*) and *atsi*(*u*) in *T*(1), that is k = 1, are given below.

```
% Assignments in time slot T(1)in TDDA
      k = 1       %first time slot T(1)
      for u =1:N
        if cost(s,u,k)= 1 %then assign
          for j=k:kmax %to satisfy (15)
            tot_cost(u,j)         =
cost(s,u,k);
          end
          prev(u) = s;
          atsi(u) = k;
        end
      end
```

H. Assignment and Update of tot_cost(u, k) in T(k)'s for k greater than one

Through direct connection: Source *s* will not have direct connection to another node *u* at k = 1 (That is *T*(1)), if *cost*(*s*, *u*, 1) = ∞. But if *cost*(*s*, *u*, k) = 1 for some k > 1, then direct connection is available for *k* > 1 between *s* and *u*. Then the *tot_cost*(*u*, *k*), *prev*(*u*) and *asti*(*u*) are assigned for *k* > 1 as follows.

```
          if  dist(u,k)>cost(s,u,k)
```

```
                 dist(u,k:kmax)=cost(s,u,k);
                          pred(u)=s;
                          asti(u)=k;
        end
```

Through intermediate Nodes: When shorter paths are possible through intermediate nodes, TDDA updates tot_cost(u, k) through those intermediate nodes. Here, for k > 1, in time slot *T*(*k*), update of *tot_cost*(*u*, *k*) is based on the distance of intermediate nodes in the previous time slot *T*(*k*−1) and the cost of the connecting link in the present time slot *T*(*k*). This is true, because the present shortest distance can be determined only from the immediate previous shortest value.

The over all assigbnent and update is carried out as,

```
    %Update for k>1
   for k=2:kmax          %update time loop
    for u=1:N        %cover all nodes
     if u==s      %except s
       continue;
     end
  % Assignment through direct connection
   if  tot_cost(u,k)>cost(s,u,k)
             tot_cost(u,k:kmax)=cost(s,u,k);
             pred(u)=s;
             asti(u)=k;
      end

  % Update through intermediate nodes
     for v = 1:N  %consider all
                %intermediate nodes
  %conditional           update           of
tot_cost(u,k:kmax)begins
       If tot_cost(u,k)>
          tot_cost(v,k-1)+cost(v,u,k)
          for j = k:kmax %to satisfy (15)
             tot_cost(u,k:kmax)=
             tot_cost(v,k-1)+cost(i,u,k);
          end
  % conditional update of
  tot_cost(u,k:kmax)ends
          prev(u)=v; %previous node of u
          atsi(u)=k; %assignment of atsi(u)
       end
     end   %end of v-loop
    end     %end of u-loop
   end       %end of k-loop
```

When the update of *tot_cost*(u, k) occurs, prev(u) and atsi(u) are set as shown above.

Our main contribution in TDDA is the new update operation of *tot_cost*(*u*, *k*) which provides smooth transition from the previous shortest node to the present one.

I. Halt at Nodes

In time dependent link cost network, the shortest path from source node *s* to another node *u* also depends on time. To model this, we treat the shortest path as the path taken by a packet that travels from *s* to *u* in the least possible number of time slots. Now, consider the scenario where a packet is travelling from source *s* to node *u* in successive time slot *T*(*k*)'s using TDDA. Consider the condition for update of *tot_cost*(*u*, *k*) which is,

$$tot\_cost(u, k) > tot\_cost(v, k-1)+cost(v, u, k) \quad (18)$$

If (18) is not satisfied for any *v* in {1: *N*}, then the update assignment represented by,

$$tot\_cost(u, k) = tot\_cost(v, k-1)+cost(v, u, k) \quad (19)$$

does not happen and the travelling packet has to halt in the k^{th} time slot which is *T*(*k*). Then the assignment given by,

$$atsi(u) = k \quad (20)$$

will be missed for this *k*. This missing assignment *atsi*(u) = *k* indicates that *tot_cost*(u, k) is not updated and there is no traversal from the previous node to the present node in *T*(*k*). Here, the missing *k* implies the halting state of the packet at the previous node in the time slot *T*(*k*).

Example 1: Consider a toy example of four nodes *v*(1), *v*(2), *v*(3), *v*(4) with *v*(1) = *s* as the source and *v*(4) = *d* as the destination as shown in Fig. 6. The link costs in the first 4 time slots are shown in Table 1.

Table 1. Link cost values

Time Slot	T(1)	T(2)	T(3)	T(4)
Value of k	1	2	3	4
cost(s, v(2), k)	1	1	∞	1
cost(v(2), v(3), k)	1	∞	1	1
cost(v(3), d, k)	1	1	1	1

in T(1), cost(*s*, *v*(2), 1) = 1. Therefore, the traversal of the packet from s to v(2) occurs in T(1). Therefore we set *prev*(*v*(2)) = *s* and *atsi*(*v*(2)) = 1.

In T(2), cost(v(2), v(3), 2) = ∞. Therefore *v*(2) to *v*(3) traversal does not occur. The packet halts at *v*(2) which is indicated by the red self-loop in Fig. 4. In T(2), *prev*(*v*(3)) remains at 0 and *atsi*(*v*(3)) remains at ∞.

But in T(3), *cost*(*v*(2), *v*(3), 3) = 1 and hence the traversal *v*(2) to *v*(3) occurs. Then the assignments of prev(v(3)) and *atsi*(v(3)) are prev(v(3)) = v(2) and *atsi*(v(3)) = 3. (The older assignments are over written.)

In *T*(4), *cost*(*v*(3), *d*, 4) = 1 and hence the traversal *v*(3) to *d* occurs. the assignments of *prev*(*d*) and *atsi*(*d*) are *prev*(*d*) = *v*(3) and *atsi*(*d*) = 4.

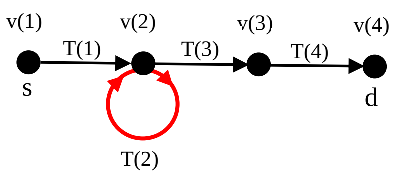

Fig. 6. Travel schedule of Example 1

The travel schedule of this example is shown in Table 2. We see that the topological distance covered by the travelling packet is 3 hops while the time taken to arrive at d is 4 time slots.

Table 2. Travel schedule for example 1.

Time Slot	Traversal	***prev***	***atsi***
T(1)	*s*→*v*(2) (hop)	*prev*(*v*(2)) = *s*	*atsi*(*v*(2)) = 1
T(2)	*v*(2)→*v*(2) (halt)	No new assignment	No new assignment
T(3)	*v*(2)→*v(3)* (hop)	*prev*(*v*(3))=*v*(2)	*atsi*(*v*(3)) = 3
T(4)	*v*(3)→*d* (hop)	*prev*(*d*) = *v*(3)	*atsi*(*d*) = 4

IV. HALTS ALONG THE SHORTEST PATH

At the successful completion of TDDA with source s, the array ***prev*** is available for all nodes from 1 to N. Let d be the destination node. Then, we get the forward path using the get_path(...) function as described in section III. B. Let the nodes along the shortest forward path from s to d be written as,

$$\textit{path\_nodes} = [v(1), v(2), \dots, v(L)] \qquad (21)$$

where v(1) and v(L) are respectively,

$$v(1) = s \quad \text{and} \quad v(L) = d \qquad (22)$$

Here, $v(1)$, $v(2)$, …, $v(L)$ are the consecutive nodes along the forward path.

Number of halts at node v(j): Consider two consecutive nodes $v(j)$ and $v(j+1)$ along the forward path for $1 \le j < L$. Let the arrival into v(j) be in time slot $T(m_1)$ and that into v(j+1) be in $T(m_2)$ as,

$$atsi(v(j)) = m_1 \qquad (23)$$

$$atsi(v(j+1)) = m_2 \qquad (24)$$

Consider the case 1, when $(m_2 - m_1) = 1$. In this case $m_2 = m_1+1$. This means, the arrival from $v(j)$ to $v(j+1)$ occurs immediately after that from $v(j-1)$ to $v(j)$. Therefore there is no halt at $v(j)$. The conclusion is, when there is no halt at $v(j)$, the condition $m_2 = m_1+1$ is satisfied.

Consider the case 2, when $(m_2 - m_1) > 1$. Let $(m_2 - m_1)$ value be designated by m_3 as,

$$m_3 = (m_2 - m_1) \qquad (25)$$

Then, $m_2 = m_1 + m_3 = m_1+1+(m_3-1)$. This means (m_3-1) extra time slots are spent at $v(j)$, before arriving at $v(j+1)$. Therefore, the number of halts (expressed in terms of time slots) at $v(j)$ designated by $halt(v(j))$ is given by (m_3-1). From (25), we see that $(m_3-1) = (m_2 - m_1) - 1$. Hence, the halt at $v(j)$ can be expressed as,

$$halt(v(j)) = m_2 - m_1 - 1 \qquad (26)$$

Equation (26) holds good in case 1 also, where $m_2 - m_1 - 1 = 0$ which means the number of halts at $v(j) = 0$. (No halts). Thus $halt(v(j)) = 0$ implies **no halt at** v.

Now, substituting for m_2 and m_1 from (24) and (23) in (26), we get,

$$halt(v(j)) = atsi(v(j+1)) - atsi(v(j)) - 1 \qquad (27)$$

When $j = 1$, Equation (27) becomes,

$$halt(v(1)) = atsi(v(2)) - atsi(v(1)) - 1 \qquad (28)$$

Since $v(1) = s$ (see (21)) and atsi(s) = 0 (see (13)),

$$atsi(v(1)) = 0 \qquad (29)$$

From (28) and (29),

$$halt(v(1)) = halt(s) = atsi(v(2)) - 1 \qquad (30)$$

Equation (30) means that if $atsi(v(2)) = 1$, then traversal from v(1) to its one hop neighbor $v(2)$ takes place in T(1). Hence there is no halt at s. If $atsi(v(2)) - 1 > 0$, then $atsi(v(2)) - 1$ gives the number of halts at s. Since (30) is a special case of (27), Equation (27) gives the number of halts for all j's from 1 to $L-1$. That is, for all nodes along the path from s to the penultimate node $v(L-1)$. Since d is the final destination, there is no need to find $halt(d)$. But as a requirement in a later algorithm it is set to zero.

By knowing ***atsi***, we can determine the number of halts at nodes along the forward path using (27). The algorithm to find the number of halts named as Num_of_Halts, is given below.

Algorithm Num_of_Halts.

```
1. Get the forward path  path_nodes as given by (20).
2. For $j = 1$ to $L-1$
   Get $halt(v(j))$ using (27) as,
   $halt(v(j)) = atsi(v(j+1)) - atsi(v(j)) - 1$
3. Endfor
4. $halt(v(L)) = 0$  //Later requirement
```

V. PATH ITINERARY WITH HALTING AT NODES

Consider the shortest path as represented by (21). Here v(1) = s = starting (source) node and v(L) = d = final destination node. By definition *asti*(d) gives the last time slot of the path. Let us call this last time slot index as *klast*. Then the travelling packet has arrived at d in $T(klast)$. From the successful assignment of asti(u) as given in (20), the value of *klast* is given by,

$$klast = asti(d) \qquad (31)$$

Now we can explore the traversal of the packet in each time slot, starting from T(1) and going all the way along the time slots T(1), T(2),…T(klast). That is, we explore the packet traversal for each T(k) for k = 1 to klast.

Path itinerary for Example 1: Let us consider the path itinerary for Example 1. From Table 2, we have halt at node v(2) in T(2) and there are no halts in T(1), T(3) and T(4). This itinerary can be depicted as shown in Fig. 7.

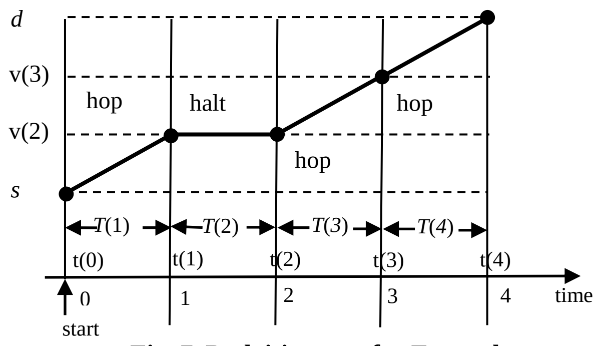

Fig. 7. Path itinerary for Example

Fig.7 shows the plot of 'the location of the packet' versus 'time points'. In Fig. 7, the hop is represented by an inclined line that connects the hop-from node and the hop-to node. The halt state is represented by a flat line connecting the same node. In Fig. 7, the travelling packet starts from node s at t(0), reaches v(2) at t(1) and so on. The different nodes visited by the packet at different time points are shown in Table 3.

Table 3. Location of the packet at different time instants

Time points	t(0)	t(1)	t(2)	t(3)	t(4)
Location of the packet	s	v(2)	v(2)	v(3)	d

Packet LocationVariable: Let the variable loc(t(k)) represent the location (position) of the travelling packet at time point t(k) . The variable loc(t(k)) gives the id of the node at which the packet is found (located) at t(k). In this paper we have taken t(k) = k. Therefore, the location variable is simply written as loc(k) instead of loc(t(k)).

Determination of loc(k)*:* Let the optimal topological path found to be, (as given by (21)),

path_nodes = [$v(1)$, $v(2)$, …, $v(L)$] with v(1) = s and v(L) = d.

Determination of *loc*(k) starts with $k = 0$. That is from $t(0)$. At $t(0)$, which is just the beginning of $T(1)$, the packet is just at the source node s or $v(1)$. Therefore,

$$loc(0) = s = v(1) \qquad (32)$$

Now consider node $v(j)$, the j^{th} node among the ***path_nodes***. At $v(j)$, from the successful assignment as given in (20), we know the value of *atsi*($v(j)$). Let

$$atsi(v(j)) = k \qquad (33)$$

By definition, the value of *atsi*($v(j)$) gives the time slot $T(k)$ in which the packet has moved from $v(j-1)$ to $v(j)$. That is, the packet is at $v(j)$ by the end of $T(k)$ or at time point k. Therefore,

$$loc(k) = v(j) \qquad (34)$$

Let, $$halt(v(j)) = m \qquad (35)$$

Here, m gives the number of halts at $v(j)$. Then the packet halts at $v(j)$ for m consecutive time slots starting from $t(k)$. Therefore,

$$loc(k) = loc(k+1) = \ldots = loc(k+m) = v(j) \qquad (36)$$

Equation (36) can be rewritten using the Matlab colon notation as,

$$loc(k : k + m) = v(j) \text{ where } halt(v(j)) = m \qquad (37)$$

When $m = 0$, Equation (37) becomes $loc(k : k) = v(j)$. This means $loc(k) = v(j)$ which is same as Equation (34). Therefore, Equation (37) is a general equation which includes (34) as a special case.

At the end of m halts, the packet moves (hops) from $v(j)$ to the next node $v(j+1)$. Then the itinerary of the packet continues until it reaches $v(L)$ which is d. Therefore, Equation (37) is successively applied for all $v(j)$'s from $j = 1$ to L to determine *loc*(k) for all k's from $k = 1$ to *klast*. The algorithm to find *loc*(k) for all k's is given below as GET_LOC.

```
Algorithm GET_LOC.
Inputs: array path_nodes, atsi and halt
Output: Array loc.
1. For $j = 1: L$
    $k = atsi(v(j))$;
    $m = halt(v(j))$;
    $loc(k : k + m) = v(j)$  // Equation (37)
2. Endfor $j$
```

VI. SIMULATION RESULTS

The TDDA is implemented in Matlab for different cases.

Example 2. The basic graph (network) is an 8x8 grid of 64 nodes with 112 links. The link cost variation has a period of 3.

Hence $cost(u, v, k) = cost(u, v, 1+ mod(k-1, 3))$. The costs of links during different time slots are shown in Table 4.

Table 4. Link cost variation with respect to time slots

Time Slots	Cost of links *cost*(u, v, k)'s	Figure	color
T(1), T(4), …, T(1+3*n)	1	Fig. 8	Black
	∞		Cyan
T(2), T(5), …, T(2+3*n)	1	Fig. 9	Black
	∞		Cyan
T(3), T(6), …, T(3+3*n)		Fig. 10	Black
			Cyan

T(1), T(4), …, T(1+3*n) are shown in Fig. 8. Black colored links have a cost of 1 (in appropriate units) and the cyan colored ones have a cost of ∞. Fig. 9 and Fig. 10.

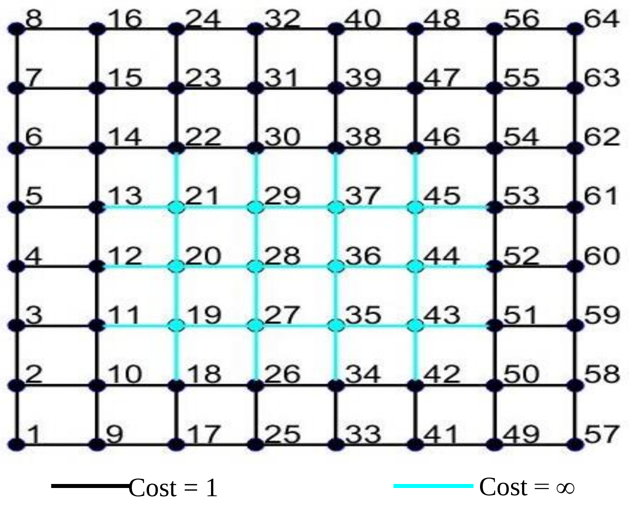

Fig. 8. Cost of links in time slots T(1+3*n)

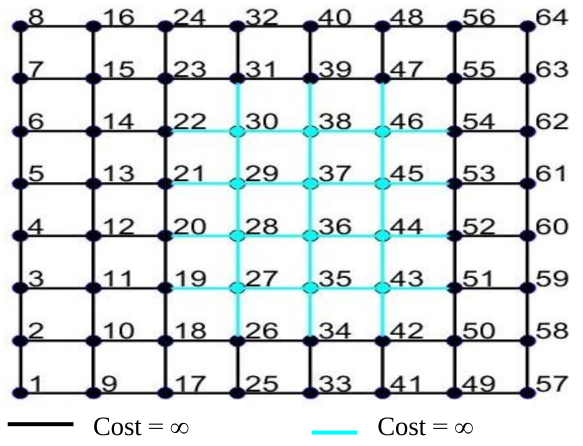

Fig. 9. Cost of links in time slots T(2+3*n)

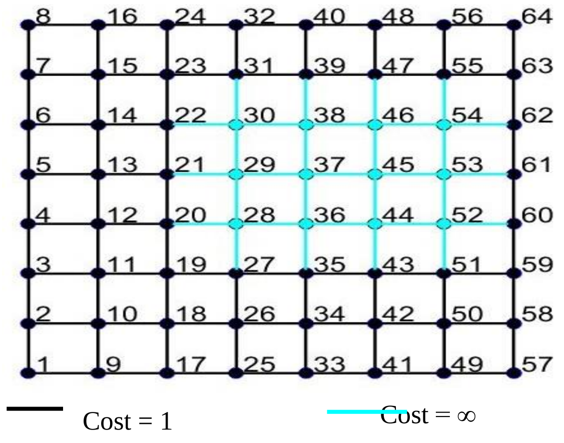

Fig. 10. Cost of links in time slots T(3+3*n)

From Fig. 8, 9 and 10, we see that the forbidden region moves from southwest towards northeast from *T*(1) to *T*(3).

Example 2(a). In this case the source s = 19 and the destination d = 43. The results are as follows. The shortest path vector is found using the ***get_path*(…)** function. Corresponding *atsi*(*v*(*j*))'s and *halt*(*v*(*j*))'s are found and these are listed in Table 5.

Table 5. path_nodes,atsi and halt values for Example 2(a).

	source	Inter-mediate nodes		destination
path_nodes	19	27	35	43
atsi	0	3	6	9
halt	2	2	2	0

The minimum cost topological length = 3. The path itinerary with halts (in red) is shown in Fig. 11. The travel time = 9.

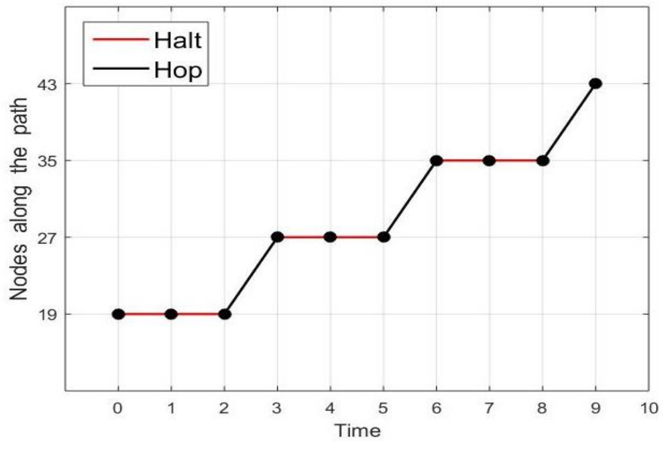

Fig. 11. Path itinerary with halts. s = 19 and d = 43

In this example, from Fig. 8 and 9, we see that during time slots T(1) and T(2) *cost*(19, 27, 1) = *cost*(19, 27, 2) = ∞. Therefore the packet is forced to halt by two time slots at the source node 19 itself. If it had taken a detour along 19−18−26−27, the travel time would have been three time slots. The path itinerary is given in Table 6.

Table 6. Path itinerary showing Halts and Hops

Time slots	From	to	Traversal type
1	19	19	Halt
2	19	19	Halt
3	19	27	Hop
4	27	27	Halt
5	27	27	Halt
6	27	35	Hop
7	35	35	Halt
8	35	35	Halt
9	35	43	Hop

Example 2(b). Here, s =18 and d = 30. The calculated path itinerary is shown in Fig. 12. The minimum cost topological length = 5. The travel time = 7.

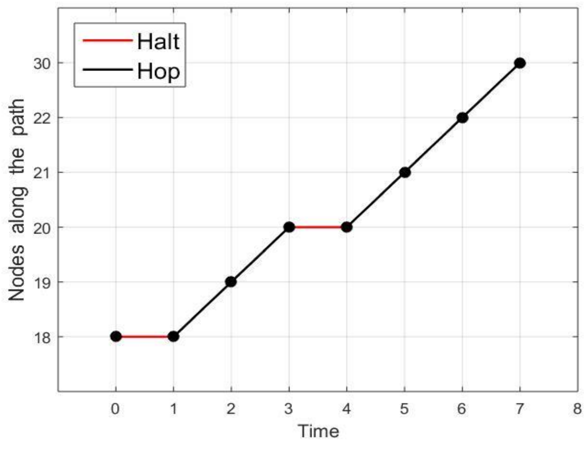

Fig. 12. Path itinerary with halts. s = 18 and d = 30

VII. INCLUSION OF HALTING COSTS

In TDDA algorithm, the traversal from an intermediate node *v* to node *u* occurs in time slot *T*(*k*), only if condition given by (18) is true. Constraint (18) is reproduced here.

tot_cost(u, k) > *tot_cost(v, k−1)+cost*(*v, u, k*)

If *cost*(*v*, *u* , *k*) = ∞, for all v's from 1 to N, constraint (18) will not be satisfied for any v and the packet has to halt (wait) at the preceding node of *u*. Then (18) is again checked in the next time slot and so on until (18) is satisfied at which time slot traversal takes place from *v* to *u*. Thus there can be zero or more number of halts at a node along the optimal path. In the earlier description of TDDA, the cost of halting is ignored. In this section, we consider the cost of halting of the packet at a node and we extend the TDDA algorithm to include the halting cost.

A. Halting or Waiting cost at a node

When a packet halts at a node there will be a storage cost, because the packet has to be stored in a buffer memory.

A halting packet at a node may cause congestion at that node and may incur the congestion cost. Halting cost may be imposed by the designer or regulator to discourage or encourage halt at nodes. For example a halting cost of ∞ per halt will prevent halting along the optimal path.

Cost per halt or halt rent: To calculate the total halting cost, we introduce the cost per halt parameter or the *halt rent* (*hr*). Halt rent at node *v* is the cost to be incurred (or paid) to halt for one time slot at node v. *Halt rent* may vary from node to node. Hence the *halt rent* at node *v* is represented as hr(v). which is a function of the node at which halt occurs. *hr*(*v*) and the cost of a link *cost*(*v*, *u*, *k*) are expressed in the same unit, so that they can be added to get the cumulative cost.

Considering various factors that affects the *halt rent*, we assume that the cost per halt at node *v*, that is, *hr*(*v*) is known in advance in proper units for all nodes before implementing the Extended TDDA (ETDDA). In this paper hr(v) is assumed to be independent of time. But time dependent hr(v) also can also be implemented, which will be implemented in a future paper.

B. Extended Time Dependent Dijkstra Algorithm (ETDDA)

The basic idea is to include the successive halting costs at halt nodes while calculating the total cost of a node from s to u at successive time slots. The halting cost at node *v* in time slot *T*(*k*) is designated by *halt_cost*(*v*, *k*) and is calculated as,

$$halt\_cost(v,k) = hr(v,k)*halt(v) \tag{38}$$

Here, *hr*(*v*) is the cost per halt at *v* and *halt(v)* is the number halts (in time slots) at node v. The value of *halt*(*v*) is obtained using (27) which is reproduced here.

$$halt\left(v(j)\right) = atsi\left(v(j+1)\right) - atsi\left(v(j)\right) - 1$$

When the packet traverses from node *v* to node *u*, node *v*(*j*+1) of (27) is *u* and node *v*(*j*) is simply node *v*. Hence (27) is rewritten as,

$$halt(v) = atsi(u) - atsi(v) - 1 \tag{39}$$

If the traversal (hop) from v to u is successful, then from (15), we know that,

$$atsi(u) = k \tag{40}$$

From (40) and (39),

$$halt(v) = k - atsi(v) - 1 \tag{41}$$

Equation (41) is used to calculate the number of halts at node *v*. (It could be zero also. Then it means, there is no halt at *v*.)

Inclusion of halt_cost at s: When halt occurs at source *s* itself, we use the following steps to assign the total cost of node *u* in time slot *T*(*k*)

```
     halt(s)=k-atsi(s)-1; %Eq. (41)
     halt_cost(s)=hr(s)*halt(s); %Eq (38)
     if tot_cost(u,k)>
        cost(s,u,k)+halt_cost(s);
        tot_cost(u,k:kmax)=cost(s,u,k)
                          +halt_cost(s);
       pred(u)=s;
       atsi(u)=k;
     end
```

Inclusion of halt_cost at an intermediate node v: When halt occurs at an intermediate node v, we use the following steps.

```
   for v=1:N
       halt(v)=k-atsi(v)-1; %Eq. (41)
       cost_halt(v)=hr(v)*halt(v);     %Eq
(38)
    if tot_cost(u,k)>tot_cost(v,k-1)

+cost(i,u,k)+cost_halt(v)


tot_cost(u,k:kmax)=tot_cost(i,k-1)
            +cost(i,u,k)+cost_halt(v);
        pred(u)=i;
        atsi(u)=k;
     end
  end
```

These two extensions are incorporated in TDDA to get ETDDA as follows.

```
  %Initialization in ETDDA
    for u = 1:N
      for k = 1:kmax
         if u==s
           tot_cost(u,k) = 0; %from s to s
         else
           tot_cost(u,k) = ∞;    %from s to
u
           end
        end      %end of k-loop
        prev(u) = 0; %undefined values
        halt(u) = 0; %default value (no halt)

        if u == s
          atsi(u)= 0  %to satisfy (13)
        else
          atsi(u)= ∞; %Initialization
        end   %end of if
      end   %end of u-loop
      %Initialization over
    % Assignments in time slot T(1)in TDDA
          k = 1        %first time slot T(1)
           for u =1:N
     if cost(s,u,k)= 1 %then assign
               for j=k:kmax  %to satisfy (15)



              tot_cost(u,j)            =
cost(s,u,k);
               end
               prev(u) = s;
               atsi(u) = k;
             end
           end
            % In T(1) halt(u) remains 0.
            % Therefore gets no assignment.
         %Update for k>1
       for k=2:kmax       %update time loop
        for u=1:N      %cover all nodes
         if u==s     %except s
           continue;
         end

    % Assignment through direct connection
      %Inclusion of halt_cost
        halt(s)=k-atsi(s)-1; %Eq. (41)
        halt_cost(s)=hr(s)*halt(s); %Eq (38)
        if tot_cost(u,k)>
           cost(s,u,k)+halt_cost(s);
           tot_cost(u,k:kmax)=cost(s,u,k)
                             +halt_cost(s);
          pred(u)=s;
          atsi(u)=k;
        end

    % Update through intermediate nodes
         for v = 1:N  %consider all
                    %intermediate nodes
    %conditional         update          of
tot_cost(u,k:kmax)begins
           halt(v)=k-atsi(v)-1; %Eq. (41)
           cost_halt(v)=hr(v)*halt(v); %Eq (38)
           if tot_cost(u,k)>tot_cost(v,k-1)
                  +cost(i,u,k)+cost_halt(v)
        tot_cost(u,k:kmax)=tot_cost(i,k-1)
           +cost(i,u,k)+cost_halt(v);
        prev(u)=v; %previous node of u
        atsi(u)=k; %assignment of atsi(u)
 % conditional update of
 tot_cost(u,k:kmax)ends
     end
    end   %end of v-loop
   end     %end of u-loop
  end       %end of k-loop
```

C. Time Complexity of ETDDA

The main nested loops v-loop and u-loop have N iterations each and contribute N^2 iterations. These N^2 iterations occur *kmax* times in the outer k-loop. Therefore the time complexity of ETDDA is O(N^2×kmax).

D.Special Case when halting rent is one

When the halting rent *hr*(*v*) is one, it means, halting for one time slot is one which is same as the cost of a connected link *cost*(*v*, *u*, *k*). Here we use one time slot for one hop. Therefore the cost of one hop in terms of time slot is one. Since halting rent *hr*(*v*) is also one, the total traversal time and the total travel cost are same. Therefore the ETDDA also minimizes the total travel time when *hr*(*v*) = 1 and *cost*(*v*, *u*, *k*) = 1. Thus ETDDA gives the shortest travel time when *hr*(*v*) = 1 and the shortest topological distance when *hr*(*v*) = 0. Note that, when *hr*(*v*) =0, ETDDA and TDDA are same.

E.Two types of shortest paths

Topological shortest path: The path from *s* to *d* which has total physical or topological minimum distance is called the *topological shortest path*. The path can have zero or more number of halts at nodes depending on the availability of suitable forward links. Here the halting cost is taken as zero. Because of halts at nodes, total time taken for the travel may be higher than the optimum. In this case, the number of intermediate nodes visited is minimum. Hence the number of hops are minimum and the corresponding transmission energy consumption is minimum. In ETDDA, *topological shortest path* is found using *hr*(*v*) = 0.

Least travel time path: The path from *s* to *d* which provides minimum overall travel time is called the *least travel time path*. Here the number of halts are minimum. In ETDDA, *Least travel time path* is found using *hr*(*v*) = 1. This provides minimum end to end delay.

Example 3. The basic graph and the link cost information are same as in Example 2. The source and the destination are *s* = 20 and *d* = 53. The halt rent *hr*(*v*) is taken same for all *v*'s (all the nodes). The topological shortest path is found with *hr*(*v*) = 0 and is shown in Fig. 13. The topological shortest path length = 7. The travel time = 14. Number of halts = 7.

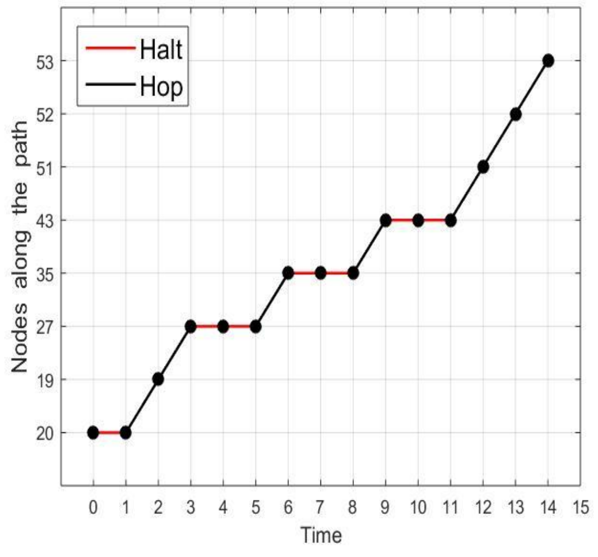

Fig. 13. Path itinerary with hr(v) = 0. s = 20 and d =

The shortest travel time path is found with *hr*(*v*) = 1 which is shown in Fig. 14. The topological shortest path length = 9. The travel time = 11. Number of halts = 2.

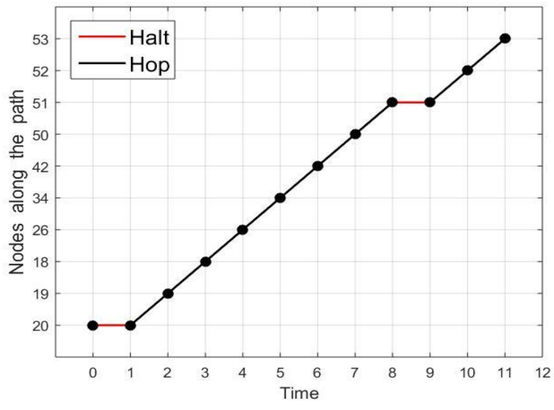

Fig. 14. Path itinerary with *hr*(*v*) = 1. s = 20 and d =

When the halt rent *hr*(*v*) increases, the number of halts gets reduced.

Example 4. Here, the number of nodes is 144 and the blocked links (links with cost = ∞) are selected randomly. The source to destination unobstructed length is taken as 13. The Halt rent *hr*(*v*) is varied from 0 to 4 and the corresponding total cost which is topological cost + halting cost, number of halts, topological length and the minimum travel time are shown in Fig. 15. In our scheme, the link cost is taken as 1 when it is connected. Therefore the topological length (from the source to the destination) and the topological cost are same. Since a packet takes one unit of time to traverse one unit of topological length (one hop), the travel time to cover the topological length is the topological length itself.

Therefore the Total travel time is the waiting time at nodes + topological length. In the present scenario link traversal for each link is taken as 1. Therefore the total travel time is the sum of the topological length + waiting time at nodes. The total cost = (topological cost + *hr*(*v*)* No. of halts). The various numerical values are shown in Table 1.

Table 1. Performance parameters for hr(v) = 0 to 4.

hr(*v*)	0	1	2	3	4
No. of Halts	5	2	2	1	1
Topological length	13	15	15	17	17
Minimum travel time	18	17	17	18	18
Total cost	13	17	19	20	21

From the values of the various parameters, we see that the topological length and the and the total cost are minimum when *hr*(*v*) = 0 and the minimum travel time is the lowest when *hr*(*v*) = 1.

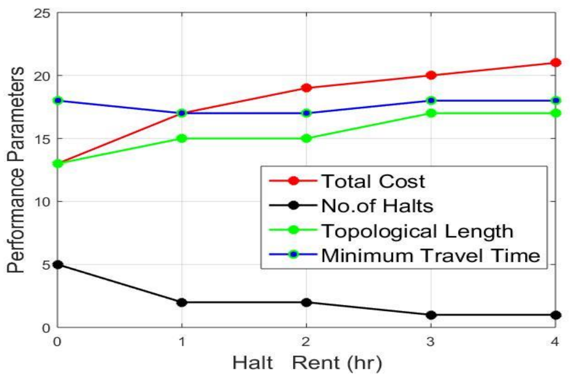

Fig. 15. Performance parameters versus *hr*(*v*)

VI. COMPARISON WITH OTHER METHODS

In the methods by Dreyfus [11], Halpern [12] and Orda and Rom [15], the main algorithm calculates the earliest arrival time and there is no provision for assigning halting cost or halt-rent. On the other hand, our method calculates the shortest topological path and the earliest arrival time simultaneously with provision for selecting suitable values for the halting cost at nodes. In our method, we can calculate the optimal halts at different nodes that make the traversal plans efficient and provides good coordination in multi-path routing.

VII. CONCLUSION

A new graph theory method of finding the shortest path in a time dependent link cost network is presented. The well-known Dijkstra algorithm is modified and extended to solve the time dependent shortest path problem. The problem is solved in the uniform discrete time domain. Our main contribution is to provide solution for both the shortest topological path and the minimum travel time problems simultaneously. Our method enumerates the optimal halts at nodes along the optimal path. By adjusting the halting cost, the designer can select the zero-halt optimal path or the topological shortest path which may use several halts as needed to achieve the shortest path criterion. Our method is well suited for unicast, multicast and broadcast communications.

REFERENCES

1. M. Amjad, M. H. Rehmani and S. Mao, "Wireless Multimedia Cognitive Radio Networks: A Comprehensive Survey," in IEEE Communications Surveys & Tutorials, vol. PP, no. 99, pp. 1-1, 2018. (Early Access article).
2. D. M. Alias and R. G. K, “Cognitive radio networks: A survey”,International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2016, pp. 1981–1986.
3. N. Bouabdallah, B. Ishibashi and R. Boutaba, "Performance of Cognitive Radio-Based Wireless Mesh Networks," IEEE Transactions on Mobile Computing, vol. 10, no. 1, 2011,pp. 122-135.
4. K. R. Chowdhury and I. F. Akyildiz, "Cognitive Wireless Mesh Networks with Dynamic Spectrum Access," in IEEE Journal on Selected Areas in Communications, vol. 26, no. 1, 2008,pp. 168-181.
5. R. B. Battula, M. S. Gaur, D. Gopalani, K. Radhika and S. Shiwani, "A dynamic medium access mechanism for cognitive radio wireless mesh networks," International Conference on Signal Processing and Communication (ICSC), Noida,2015, pp. 434-438.
6. K. M. F. Rabbi, D. B. Rawat, M. A. Ahad and T. Amin, "Analysis of multi-hop opportunistic communications in cognitive radio network," SoutheastCon 2015, Fort Lauderdale, FL,2015, pp. 1-8.
7. Akyildiz, I., Wang, X., & Wang, W, “Wireless mesh networks: A survey. Elsevier Computer Networks”, 47(4), 2005, pp 445–487.
8. Deo, N., and pang, C. Y, “Shortest path algorithms: Taxonomy and annotation.”, Networks, volume 14, Issue 2,1984, pp 275-323.
9. L. Cooke and E. Halsey, "The shortest route through a network with time dependent internodal transit times", Journal of Mathematical Analysis and Applications, 1966,pp 492-498.
10. Klafszky, E. Determination of shortest path in a network with time-dependent edge-lengths. Math. Oper. Stat, 1972,pp 255-257.
11. Dreyfus, S. E. An appraisal of some shortest path algorithms. Oper. Rex I7, 1969,pp 395-412.
12. Halpern, J. The shortest route with time dependent length of edges and limited delay possibilities in nodes. Z. Oper. Res. 21 (1977), 1977, pp117-124.
13. D. E. Kaufman, R. L. Smith (1993). "Fastest paths in time-dependent networks for intelligent-vehicle-highway systems application". IVHS Journal 1, 1-11.
14. Daniel Delling and Dorothea Wagner. Time-dependent route planning. In Robust and Online Large-Scale Optimization, volume 5868 of Lecture Notes in Computer Science,2009, pp 207–230.
15. A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. J.ACM, 37(3), 1990,pp 607–625.
16. B. Ding, J. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In Proc. EDBT, pages 205–216. ACM, 2008.
17. B. C. Dean. Continuous-time dynamic shortest path algorithms. Master’s thesis, Massachusetts Institute of Technology, 1999.
18. W. Zhan-yu and G. p. Liu, "Earliest-arrival path: A global optimized transmission for networked control systems," 2017 36th Chinese Control Conference (CCC), 2017, pp. 7909-7914.
19. Abdelfattah Idri, Mariyem Oukarfi, Azedine Boulmakoul, Karine Zeitouni and AliMasri “A new time-dependent shortest path algorithm for multimodal transportation network,” Procedia Computer Science Volume 109, 2017, pp 692-697.
20. https://ahmedhanibrahim.wordpress.com/2016/04/15/solving-time-dependent-graph-using-modified-dijkstra-algorithm/.
21. Shylashree N, “Non-crossing rectilinear shortest minimum bend paths in the presence of rectilinear obstacles”, Journal of Telecommunication and Information Technology, no. 3, 2018, pp 82-91.