A Novel Binary Lock based Collision Avoidance Algorithm for Transportation Networks

Sankhadeep Chatterjee, Debarshi Banerjee

1Department of Computer Science & Engineering

2Hooghly Engineering & Technology College
Vivekananda Road, Pipulpati, Hooghly-712103, West Bengal, India

ABSTRACT

A novel collision avoidance algorithm is proposed to resolve the problem of collision among independent entities in transportation networks. The algorithm aims in scheduling multiple entities in a network in such a way that eliminates every possibility of collision between any two entities in the network. To achieve the desired goal the algorithm takes help of Binary locks which has found enormous and successful application in the field of Database management systems. Considering each node of the network as a resource and asserting a Binary lock for each of them the algorithm ensures concurrency control for each node in the network thereby eliminating any possibility of collisions what so ever. A smart prediction method has been employed to prevent any kind of indefinite scheduling in the network. Decentralising the intelligence over the network the algorithm avoids any possibility of collisions before it can really happen. A detail analysis and experimental results have been provided in support of its ingenuity.

Keywords: Collision Avoidance, Transportation Network, Signal Set, Priority scheduling, Binary Lock

1.INTRODUCTION

Transportation Engineering aims in planning, designing, and managing different facilities in different modes of transportation to achieve a safe, efficient, flexible and economical movement of different entities in a transportation network. It finds an extensive application mainly in air, highway, railway, and waterway transportations. The matter of utmost concern is to achieve safety in all of these different transportation networks to provide an immaculate service in respective systems. Collision avoidance has become an imperative part of modern transportation systems in connection to achieve maximum safety in both traditional and intelligent transport systems. An immense application of this rapidly growing technology in automobile safety system in reducing the severity of accidents being popularly known as pre-crash system, forward collision warning systems or collision mitigating system has been reported [1], in railway safety and signalling for all railroads [2], in large vessel at sea that are out of the range of shore-based system by using AIS, in networking with CSMA and several other fields. In order to solve collision problem in IEEE 802.15.4 LR-WPANs, a novel channel scheduling scheme was proposed [3]. The proposed scheme which is an adaptive method with fast recovery, aimed at minimizing the possibilities of beacon collision by efficiently utilizing multiple available channels in a hybrid manner combining proactive and reactive methods. To emphasize various issues in the field for further research related to collision avoidance surveys are being carried out on the co-existence between IEEE 802.11 (2.4 GHz Wireless Network) and IEEE 802.15.4-based wireless network [4-5]. Studies have also been carried out on different CSMA/CA IEEE 802.11 based implementations. The study has revealed the access protocol performance in terms of available throughput, access delay and packet dropping. A comparative study on the performance of IEEE 802.3 based CSMA/CA vs. CSMA/CD used in wired LANs has also been report ed [6].Using vibrating sensors and Zigbee technology an efficient method of avoiding train collisions has been reported [7]. Wireless communication technology has been used to transfer data or signals over a part of the entire communication network. The safety is ensured in terms of saving lives and property through wireless implementation of sensor network. Collision avoidance system is also implemented at various nodes viz., server side node, train side node, track side node and station side node. Feasibility studies have also been carried out on two methods of crack detection in rail networks and avoidance of collision between the rails [8]. The collision avoidance algorithm being developed in context to state estimation introduces the concept of force field and warning function in the roundabout [9]. Vehicles, by selecting the safety operation mode, can be avoid the conflict area and pass the merging points by using collision avoidance algorithm. A hybrid algorithm based supervisor is helping in the synthesis of a least restrictive controller has been reported in
literature [10] for collision avoidance in case of multiple vehicle at an intersection. Research is also reported on intelligent collision avoidance and adaptive cruise control. Recent advancements and research trends in collision avoidance based warning system and automation of vehicle longitudinal/ lateral control tasks has been reported [11]. Research is focussed in order to implement automation in different levels of transportation system with specific emphasise on vehicle level automation. In commercial airlines collision avoidance is of prime concern due to occurrence of a series of mid-air collisions in recent years. Collision avoidance systems for both manned and unmanned aircraft must reliably prevent collision with minimising alerts. An on board next-generation collision avoidance system aiming to provide a higher degree of safety without interfering with normal, safe operations has been proposed [12]. Research has also been focused in finding robust collision avoidance optimization to modelling errors [13]. Moreover research is being carried out in reducing the complexity of computing the optimal strategy where only some of the problem dimensions are controllable [14]. Such an approach finds application in aircraft collision avoidance when the system recommends manoeuvres to an imperfect pilot. A modification over existing TCAS (Traffic Alert Collision Avoidance System) logic has been proposed [15]. The approach involves leveraging recent advances in computation to automatically derive optimized collision avoidance logic directly from encounter models and performance metrics with an outline on anticipated impact on development, safety and operation. Collision avoidance finds its image in the Automatic Identification System (AIS) which is basically an automatic tracking system used on ships and by VTS (Vessel Traffic Session), originally developed for identifying and tracking of vessels by electronically exchanging data with other nearby ships, AIS base stations and satellites. The AIS system finds its application in avoiding collision among large vessels at sea that are out of the range of shore based systems. The proposal for the establishment of a regional AIS application specific message register has been reported in literature [16]. An efficient collision avoidance algorithm has been proposed [17] for transportation networks based on Dynamic reference set method. The analysis of the algorithm has revealed a possible application of Binary Lock mechanism which is widely popular in the fields of Database management systems. Locks have been reported to be well under different loads of the system [18]. Consistency of the system is found to be guaranteed while using Locks [19]. Thus the wide popularity and immense, diversified application of collision avoidance prompted the authors to propose a new efficient binary lock based collision avoidance algorithm using dynamic Reference Set method in resolving the collision among independent network entities. The proposed algorithm is aimed at preventing collision before it can really happen by decentralising the intelligence. The implementation of the proposed algorithm in near future will assure preventing of any type of collision at any node at any instance between any two entities.

2. PROPOSED METHODOLOGY

2.1 IMPLEMENTATION OF SIGNAL SET

The collision avoidance algorithm schedules multiple entities willing to travel from a given ‘Source’ node to another ‘Destination’ node. A Signal set ($S_{\text{Destination}}$) is a set of nodes of a network such that any node $n_i$ of $S_{\text{Destination}}$ may be needed exclusively by any entity $E_j$ at any time $T_k$ to travel from a given source node to a given destination node. The proposed algorithm uses the smart Set (Signal Set) of nodes ($S_{\text{Destination}}$) which is a subset of the set of all nodes (N) of the network. Before selecting next node for each entity $E_j$ at any time $T_k$ the algorithm refers the Signal Set which in turn provides knowledge to the algorithm regarding the status of the node i.e. whether that node of interest can be used in scheduling or not. The following algorithm generates the Signal Set for any given Destination of any Network.

![Fig. 1: A typical Network (N) before Signal_Generator is being applied](image-url)
Algorithm 1: Signal_Generator (Destination)
\[
//G[1:n][1:n] is the graph depicting the Network (N). SignalSet[1:n] holds the status of Signal Set (S_{Destination}).
//visited[1:n] is an array which holds the visiting status of the node and all elements are initialized to false, //visited[i] set to true if algorithm has already visited node i.
\]
1. push (Destination)
2. visited[Destination]←1
3. do until Stack is empty
   4. x←pop()
   5. SignalSet[x] ←1
   6. Repeat until i = | N |
      7. Check if (G[i][x] ≠ 0 and visited[i] = 0) then
         8. push(i)
         9. visited[i] = 1
      10. end if
      11. next i
   12.wend

Consider a typical Network shown in Fig. 1 whose set of nodes N = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. The Source is set as node ‘1’ and destination set as node ‘8’. Fig. 1 is depicts the Network before being processed with Signal_Generator. Fig. 8 depicts the same Network after being processed with Signal_Generator. The nodes which can be used by any entity E_j at any time T_k are denoted by Green and nodes which must be avoided are kept in red. After execution Signal Set = {1, 2, 3, 4, 5, 6, 8}.

2.2 IMPLEMENTATION OF BINARY LOCK

A lock is typically a variable, associated with one node, describes the status of operation that can be done with it. Binary Lock is a stricter version of Lock which can have only two states ‘0’ or ‘1’. If the value of the Lock associated with one node is ‘1’ then it can’t be accessed, otherwise if it is ‘0’ then it can be accessed if it is requested. At any time T_k, any entity E_i can request to start travelling for any node which is currently connected with its present position, the request can only be granted if and only if the value of the Lock associated with the node it want to travel to is ‘0’. If it is found that the value is ‘1’ then some other entity is still acquiring it and the request can’t be granted. Once an entity get access to one node it changes the value of the Lock to ‘1’ and when it leaves it resets the value. The following algorithm describes the procedure of requesting.

Algorithm 2: LOCK(i)
\[
// Lock[1:n] is an array holding values of Binary Locks associated with the nodes. Lock[i] = 1 means
// node ‘i’ is already acquired by an entity. Lock[i] = 0 means it is free. It returns True if request can
// be granted else it returns False.
\]
1. if Lock[i] = 0 then
2.   set Lock[i] = 1
3.   return (true)
4. else
5.   return (false)
6. end if
7. end

The next algorithm shows how an entity unlocks one node.

Algorithm 3: UNLOCK(i)
\[
1. set Lock[i] = 0
2. end
\]

It may be possible that at any time T_k, more than one entity can request for same node. In such cases the entity priority is used to take the decision that which entity’s request will be granted. Fig 4. depicts the status of a typical network between two time instances any time T_k and T_{k+1}. It depicts different cases of Binary Locks, for example entity E_5, E_6 is currently holding source node, while both are requesting for node ‘2’ which is currently being held by E_4. To resolve the situation priorities must be used to schedule the entities further. But E_5 is holding ‘3’ and requesting for ‘4’ which
can easily be resolved. If \( E_1 \) leaves node ‘5’ then as per priority \( E_2 \) will get chance to access node ‘5’ while \( E_4 \) has to wait. Fig 2. to Fig 7. depict status of different locks associated with different nodes of a typical network at different time instances. The network consists of five nodes and entities are aimed to travel from NODE[1] to NODE[5]; the value in each reflects the status of the lock of that corresponding node. For instance at \( T_0 \) NODE[1], NODE[2] and NODE[4] is currently being held by some entities. The next transition can take place from NODE[2] to NODE[3] and NODE[4] to NODE[5]. Each possible transition is shown in blue edges. First of all the entity at any node requests to acquire a lock on the next node it wants to move on. If the status of its lock is ‘0’ then depending on priorities one of the requesting entities will lock the node. After acquiring the lock it will release all locks which it already has to ensure a collision free environment. For example at \( T_1 \) the entity at NODE[2] has reached NODE[3] and entity of NODE[4] has reached NODE[5], thereby unlocked NODE[2]. According to priority entity of node NODE[1] acquired lock of NODE[2] and travelled to it. In the subsequent figures the next transitions has been depicted. Fig 9 depicts a Wait-For graph at time instance ‘3’ of the network depicted in Fig 1.

2.3 PROPOSED COLLISION AVOIDANCE ALGORITHM

The proposed Collision Avoidance algorithm uses the concept of Signal Set and Dynamic Reference Set to schedule multiple entities in a Network avoiding any type of collision at any time instance. Before the algorithm runs, the Signal_Set generates the Signal set first for given Source and Destination. After having the Signal Set
the Collision Avoidance algorithm starts scheduling entities for each time instance \((T_k)\) until all the entities reach the Destination node. One fundamental criterion about entities is that the entities are scheduled in descending order of priorities i.e. for any \(\alpha < \beta\), priority of \(E_\alpha\) is greater than priority of \(E_\beta\). The Network is taken as a non-weighted simple graph where each entity takes same amount of time to travel through any edge.

Algorithm 4: Collision Avoidance

\[
// G[1:n][1:n] \text{ is the graph depicting the Network (N). flag is initialized to 1. num}_e \text{ is the number of entities and }\num_v = |N|. Present\_Position[i] \text{ holds the node acquired by entity i. Signal\_Set[]} \text{ implements Signal Set } // (S_{Destination}). t \text{ is initialized to 0. It holds the time instance } (T_k).
\]

1. until flag = 1 do
2. set flag = 0
3. repeat for e = 1 to num_e
4. if Present\_Position[e] = Destination then
5. set flag = 1
6. repeat for j = 0 to num_v
7. if G[Present\_Position[e]][j] = 1 and LOCK(j)
8. and visited[e][j] = 0 and Signal\_Set[j] = 1 then
9. UNLOCK(Present\_Position[e])
10. set visited[e][j] = 1
11. Present\_Position[e] = j
12. break
13. end if
14. next j
15. end if
16. set UNLOCK(Destination) = 0
17. next e
18. set t = t + 1
19. if flag = 1 then
20. print Present\_Position
21. end if
22. wend

Fig 8. Network of Fig. 1 after Signal\_Generator is being applied
3. RESULTS & DISCUSSION

The proposed Collision Avoidance algorithm schedules multiple entities in a network keeping track of its priority constraints. Hence it is expected that for any \( \alpha < \beta \), \( E_\alpha \) will reach the destination before \( E_\beta \). Consider there are \( \lambda \) many entities to be scheduled. For all \( \alpha < \lambda \), priority of \( E_\alpha \) is less than \( E_\lambda \). Hence \( E_\lambda \) is expected to be the last entity to reach destination. It can be possible to estimate what would be the maximum time taken by one entity \( E_j \) (\( j \leq \lambda \)) to reach the destination. To estimate the time first we divide the whole network into several layers depending on the Source and Destination node. One Layer (\( L_\iota \)) is a set of nodes where each node \( n_i \in L_\iota \) and \( n_i \in S_{Destination} \) and each \( n_i \) is almost at same distance from Source, except Source and Destination provided that for any \( \alpha \) and \( \gamma \) there will be no connection between \( n_\alpha \) and \( n_\gamma \). Distance indicates the time to travel from destination to that node. Fig 10 depicts a typical Layering of Network depicted in Fig. 1. The layering of the Network may result in multiple Layers. For instance \( L_1 = \{1, 2, 3\} \) and these nodes can be reached in equal time from Source. Generally one Layer may contain \( \omega \) number of nodes where \( \omega < | S_{Destination} | \) must hold, because the authors are concerned about the intermediate nodes through which entities will travel from Source and destination. Hence Source and Destination are not included in any Layer. In general, each entity travels from one Layer to another Layer to reach the Destination. To get chance to enter one Layer at least one node of that Layer must be empty. For \( \varepsilon \) no of Layers of a Network, every entity will take \( (\varepsilon + 1) \) amount of time (Actual Cost) if it doesn’t have to wait at any node. But generally if the number of entities having greater priorities is at least one then the entity of our interest must have to wait (Redundant Cost) until those higher priority entities leave the layer unless next Layer accepts sufficient amount of entities at any time instance. Hence total time (Required Time) taken by an entity \( E_j \) to travel from Source to Destination is Required Time = Actual Cost + Redundant Cost. Each Layer has one Intake Capacity and Release Capacity. Intake Capacity of Layer (\( I(L_\iota) \)) is the number of incoming paths of that Layer and Release Capacity (\( R(L_\iota) \)) is the number of outgoing paths of the Layer. There can be multiple entities in a Layer at any instance of time \( T_i \) where number of entities must be less than or equal to the Layer Capacity (\( C(L_\iota) \)) which is the cardinality of the set \( L_\iota \). A Source or Destination node can be thought of as a Layer having infinite Intake Capacity or Release Capacity. An entity can only be released from a Layer \( L_\iota \) iff all the entities having higher priorities have already been released from the Layer \( L_\iota \) and have been accepted by the next Layer. Thus if \( I(L_\iota) > R(L_\iota) \) for any Layer \( L_\iota \), then \( I(L_\iota) = R(L_\iota) \) number of entities have to wait for at least one time instance in \( L_\iota \). It is thus evident that if \( I(L_\iota) \geq R(L_\iota) \) for any Layer \( L_\iota \), no entity has to wait in \( L_\iota \). The waiting time of an entity \( E_j \) can be calculated by observing the Intake-Release Status table (IRS table) of the Layers through which it has travelled. IRS table for any entity \( E_j \) depicts the detailed movement of the entity through the network by indicating whether the entity is waiting at any time instance \( T_i \) or moving normally. For example Table 1 depicts the movement status of entities by using the number of Intakes and Releases of entities. Intake is the number of entities which have been accepted by any Layer \( L_\iota \) at time instance \( T_i \) and Release denotes the number of entities which are leaving any Layer \( L_\iota \) at time instance \( T_i \). The letter ‘W’ denotes that any entity \( E_j \) is waiting in any Layer \( L_\iota \) at time instance \( T_i \) and ‘T’ denotes it is travelling. Thus by counting the number of ‘W’s in all the ISR tables for any entity \( E_j \) total waiting time (Redundant Cost) of that entity can be calculated. For instance in Table 1 entity \( E_3 \) is waiting at time ‘1’ and at time instance ‘2’ it gets a chance to move on to the next Layer and as it starts travelling. ‘T’ is indicated at time instance ‘2’ at both ISRs of Source and \( L_2 \). Further
study reveals that total waiting time of entity $E_5$ is 2. Time requirement analysis draws a vital conclusion about the maximum Required Time of an entity $E_j$. The results can be used to assign entities before being scheduled. Fig 11 depicts a graphical representation of scheduling of 9 different entities in the Network depicted in Fig. 1. A plot of node vs. time instance has been shown in the graph where the time instance starts from ‘1’ and at each time instance it reflects the status of each entity along with corresponding nodes. The graph reveals how the entities are travelling through the network along with their waiting status.

### Table 1. IRS of Source for $E_5$

<table>
<thead>
<tr>
<th>Entity Status</th>
<th>Time</th>
<th>Intake</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>1</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>-</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 2. IRS of $L_j$ for $E_5$

<table>
<thead>
<tr>
<th>Entity Status</th>
<th>Time</th>
<th>Intake</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 3. IRS of $L_2$ for $E_5$

<table>
<thead>
<tr>
<th>Entity Status</th>
<th>Time</th>
<th>Intake</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. IRS of Destination for $E_5$

<table>
<thead>
<tr>
<th>Entity Status</th>
<th>Time</th>
<th>Intake</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

For example at time instance ‘2’ $E_1$ is at node ‘2’, $E_2$ is at node ‘3’, $E_3$ is at node ‘4’, rest of the entities are still at Source (node ‘1’). Watching the graph from left to right for a particular node will reflect how it is acquired by different entities at different time instances. For instance node ‘3’ is acquired by $E_2$ is at time instance ‘2’, by $E_3$ is at time instance ‘3’and ‘4’, followed by $E_6$ is at time instance ‘5’ and ‘6’. The graphical representation is one of the utmost analyses in support of the assurance of preventing any type of collision at any node at any time instance between any two entities by the proposed Collision Avoidance algorithm.
4. CONCLUSION

In this paper the authors have proposed a novel Collision Avoidance algorithm using Binary Lock mechanism which aims in scheduling independent entities in a network, preventing any possibility of collision. The problem of collision avoidance seeks high importance in different real life problems. Keeping it in mind, the algorithm is designed to be as generalized as possible to encompass the large quantity of different collision avoidance problems in transportation networks. The proposed methodology is elaborate and shows how different independent entities can travel without making any collision. The present work will be immensely helpful for the researchers in future application to specific domains as per need.

References

[2.] http://www.argeniarailwaytech.com


Fig 11. Graphical representation of movements of entities in the Network depicted in Fig. 1 having 9 entities. Actual Cost and Redundant Cost can be measured from this representation to compute Required Time of travelling of any Entity.


AUTHOR

Sankhadeep Chatterjee is pursuing B.Tech Course in Computer Science & Engineering from Hooghly Engineering & Technology College, West Bengal, India. His areas of interest are computer network, design and analysis of algorithms, artificial intelligence and optimization techniques. He has published papers related to the field of computer science in International Journals & Conferences.

Debarshi Banerjee is pursuing B.Tech Course in Computer Science & Engineering from Hooghly Engineering & Technology College, West Bengal, India. His areas of interest are computer networking, optimization techniques, logical analysis and data analysis. He has been a part of several research projects related to the field of computer science. He has appeared in many national and International competitive coding competitions.