Journal of Network and Information Security
9 (2) 2021, 30-41
http://www.publishingindia.com/jnis

SMADS - An ITS in a Wide-Area Public Transportation Network

Shushan Zhao[1*] and Wes C. Chiang[2]
[1]Department of Computer Electronics and Graphics Technology, Central Connecticut State University, New Britain, CT, USA. Email: shushanz@ccsu.edu
[2]Department of Management and Education, University of Pittsburgh, Bradford, PA, USA.
*Corresponding Author

Abstract: In this paper, we propose an untraditional Intelligent Transportation System (ITS) that deploys in a wide-area public transportation network. Based on emerging digitalized supply chain service (including logistics), peer-to-peer/ad-hoc network, blockchain and Internet of Things (IoT) technologies, our proposed SMADS system has these new features and advantages compared to traditional transportation systems: self-organizing, multi-relayed, across-strangers, decentralized, and sharing-economy-based. We identify two essential requirements for this network system — availability and reliability. To meet these requirements, we compare this network system with existing similar ones and pinpoint two major technical challenges — connectivity/routing maintenance and trust management. We apply latest development in technologies in above-mentioned areas to design solutions to address these challenges. The resulted system can operate on top of existing multiple separate segments of transportation links to form a wide-area transportation network, which enlarges services with low cost. Additionally, the notion has potential application in a wide range of various services. We hope to bring this to readers' attention for further discussion and improvement.

Keywords: Blockchain, Connectivity, ITS, Network, Routing.

I. Introduction

City logistics and urban mobility are essential part of modern life. An Intelligent Transportation System (ITS) applies advanced hardware, software, and communication technologies, as well as planning, operation, and control methods, to transportation, whereby improves city logistics and urban mobility, including the transportation of people and freight. Traditionally, according to the scope of the system, ITS is classified into two broad classes: Commercial Vehicle Operations (CVO) for system-wide, regional, national, or continental applications and Advanced Fleet Management Systems (AFMS) dedicated to the operations of a particular (group of) firm(s) [10]. ITS integrates advanced electronics, sensors, information technology and communication to improve the convenience and security of transportation systems [14].

In traditional logistics industry, one firm transports passengers or goods all the way from source to destination; or multiple firms that have signed cooperation contract cooperate to accomplish the task. In this way, some logistics firms provide nation-wide service, and some provide only local service. We have been thinking about the question that motivated us to this research: Can these local logistics firms, or even interested individuals, cooperate to build a logistics network that provides nation-wide logistics service? A few years ago, we could not figure out a solution — there is no control center to coordinate them, they do not know how to find one willing to cooperate from one area to another, and they do not trust each other. Thanks to latest development in technologies from ad-hoc/peer-to-peer networks, blockchain, and IoT, this is possible now. The major contribution of this paper is the notion of a novel ITS that builds a nation-wide network on top of independent existing logistics networks or segments, which runs across strangers in different cities and unknown to each other.

There are some traditional service providers or operators in passenger or goods transportation, e.g. Greyhound Coach and Fedex/UPS. Recently Uber and Lyft are well known taxi service providers; LaZooz and Arcade City [1, 2] provide decentralized ride sharing services. The differences from these similar services as well as challenges for our novel SMADS system are as follows, and summarized in Table I:

- Traditional passenger or goods transportation service providers employ a large number of full-time or part-time employees and provide a variety of benefits, and always have issues such as union negotiation and employee strike etc. The novel network is based on sharing economy. There are no real employees, and costs of labor and fuel etc. apportioned to the service are significantly lowered down.
- Online taxi service requests are usually inside one city or several neighboring cities, and only one driver is

involved. SMADS could span multiple cities, and thus multiple service providers are involved. In this respect, service availability, routing efficiency, and cooperation and trust issues ensue.

- Traditional logistics firms operate in multiple cities, but they have dedicated drivers, vehicles, and routes. They do not have concerns on service availability, routing efficiency, or cooperation and trust issues.
- Online taxi service providers, such as Uber and Lyft, are centralized. A centralized system requires higher investment in hardware, software and maintenance. What's more, a centralized service is subject to Denial of Service (Dos) and Distributed Denial of Service (DDoS) attacks. SMADS is decentralized and does not have these concerns.

TABLE I: COMPARISON ON FEATURES OF TRADITIONAL SERVICE PROVIDERS AND SMADS

Features	Greyhound/Coach	Fedex/UPS	Uber/Lyft	LaZooz/Arcade City	SMADS
Sharing economy	No	No	Possible	Possible	Yes
Decentralized	No	No	No	Possible	Yes
Self-organizing	No	No	No	Possible	Yes
Multiple relayed	No	No	No	No	Yes
Across strangers	No	No	No	No	Yes

Because of these differences and challenges, we cannot use existing solutions to build such an ITS system. We propose a novel ITS named SMADS (Self-organizing, Multi-relayed, Across-strangers, Decentralized, and Sharing-economy-based).

This type of logistics networks has advantages over traditional logistics service: From the service providers' (i.e. drivers') standpoint, joining a nation-wide logistic network brings more customers and profit, while they do not need to travel to several thousands of miles away spanning tropical regions and freezing world. From the customers' standpoint, they have more choices to send parcels to or travel to a far-away place, paying much less than what they have to pay currently.

The rest of this paper is organized as follows: Section II reviews related work in the area of ITS in literature. Section III shows an overview of the framework of such a novel SMADS logistics network. Section IV presents our solution to service availability and connectivity issue. Section V explains the solution to trust and cooperation issue. Section VI demonstrates simulation results and performance evaluation. Alternative ideas and future work are discussed in Section VII. Section VIII concludes the paper.

II. LITERATURE REVIEW AND RELATED WORK

There is an extensive literature on various aspects of ITS we are studying in this work.

Routing the traffic in various situations in a transportation network is the most studied topic in ITS. Li *et al*. [13] point out that in ITS studies, methodology innovation is to some extent lagging behind vehicle infrastructure and information and communication technologies, and represents a great challenge for industry and academia. They propose a new integrated planning problem for intelligent food logistics systems.

Myung *et al.* [19] consider the problem of how to send goods with an additional decision for bundling from suppliers to customers at minimum cost --- so-called freight transportation network problem with bundling option. They propose an efficient heuristic that uses a network flow algorithm, which finds near-optimal solutions even for fairly large problems within a reasonable time.

Inspired by [16, 17, 18] etc., Chen *et al.* [15] propose a deviation path algorithm called KSP-LPA* for exactly finding k shortest simple paths in a road network between a given origin and a destination. Their contribution is to speed-up deviation path calculation by reusing results of shortest path tree generated at the previous search, thus avoiding the large computational overhead of the M-P algorithm incurred because the entire shortest path tree needs to be updated during each calculation of the deviation path. Their algorithm is claimed to achieve the optimal solution of finding k shortest simple paths, with optimality having been proved.

As for multi-prodiver or multi-segment cooperative transportation, Crainica *et al.* [10] have considered combinatorial auctions in which participants are allowed to bid directly on attractive bundles. They also propose a research direction of the so-called advisors based on enhanced Advanced Fleet Management Systems to assist carriers when participating to auctions.

De and Giri [22] propose a closed-loop supply chain which consists of multi-manufacturer, multi-depot, multi-distribution/collection center for multi-product delivery, and integrate transportation location routing network.

Kuang *et al.* [20], based on works of [11, 12, 21] etc., find logistical transport routes with optimal cost and time, among routes that are priced and congestible, in a modeling framework of a multi-segment transportation system of either serial or

parallel structure of transport provider competition. Their work shows that parallel duopoly competition among transportation providers is preferred over serial duopoly, regardless of the latency parameters of the system, and there is a conflict of interest between the supply side of the market and social welfare for low latency parameter values.

All these works consider transportation in a city-wide local-area network or a firm-wide private network, and do not consider the wide-area public network where the network is neither pre-configured nor static, and where the passenger or goods traffic needs to flow through an untrustworthy stranger's link. An ITS in a wide - are public transportation network is a novel topic and the particular contribution of our work. A preliminary work from us for mail service is presented in [23].

III. Overview of the Framework

SMADS system is designed to be decentralized and self-organizing, because of the high cost and security concerns of traditional centralized system mentioned in Section I.

To be self-organizing, the system needs a service portal website. From "Register to be a Service Provider" page, a driver can register as a service agent, and download a service app. After registering and installing the app, the driver becomes an official service agent.

A customer can request logistics service from "Request Service" page of the service portal website. Whenever a request is submitted, a service agent node can run a connectivity detection and route - finding algorithm that searches the drivers information database and returns service availability and route information if available.

To be decentralized, SMADS system stores the service agent information and the transaction information in all participating nodes. Blockchain technology is employed in generation and maintenance of the decentralized database. To help generate and verify integrity of blocks of the distributed database (This is called mining in blockchain terminology), qualified agents can earn rewards. The following components are needed:

- Registration: A driver can register as a service agent on the service portal website, and get a unique account ID assigned. To ensure security of persons and properties, some background check, such as identity verification, address/phone number verification, etc., needs to be conducted. A driver registers his/her service areas, and can choose to be a local-area agent, long-distance agent, or long-distance & local-area agent, depending on their provided source and destination areas. After registration, the new service areas data is broadcast to all participating nodes in the network through the app they have installed.
- Service Request Submission & Acceptance: When a customer needs a logistics service, he/she submits a service request on the service portal website or app at home, or do that in a service agent kiosk. The system determines the route and fee to charge.

  The website accesses to a centralized database. To alleviate the burden of the website, a customer's app can be installed. The app has local database synchronized with the system. The app has same functionalities as the website, such as deter-mining the route and fee, but can provide a discount to encourage loyalty. Service confirmation and tracking information for items are provided to the customer.
- Item Relay/Tracking and Payment: The service agents' app determines route and sends notification messages to involved agents en route. Reception point agent receives payments from the customer. If there are other agents en route, he/she exchanges part of received payment to SMADS internal digital currency, deposits it to his/her own account, and transfer to them. Smart contract calculates and transfers proper amount of payment to the next agent when handing over the item. For example, the reception agent receives $20, and transfers the item and $16 payment to the second agent. The second agent transfers the item and $10 payment to the third agent. The third agent transfers the item and $5 payment to the fourth agent who hands the item to the receiver and earns $5 payment. A transaction is generated and written to the distributed transaction database when an item is handed over and a payment is made. A transaction includes timestamp, previous agent ID, next agent ID, and the tracking number of the item etc. The customer can track the item with the tracking number through the service portal or the service app.

In the above process, there are mainly two technical difficulties: how to determine a route from the sender to the recipient, and how to ensure cooperation among multiple agents who do not know each other. We will look at them further in the next 2 sections.

IV. Network Connectivity and Routing Management

Being decentralized and self-organizing, SMADS system has the weakness of coverage at the early stage of the system. It should start from some cities where there are many drivers and many customers. Then as more and more drivers are earning money from being service agent of the system, more and more new drivers will be joining the network and the service coverage will be growing larger and larger.

When registering, the driver inputs his/her source and destination service areas. A service area is indicated by a postal code. Immediately contiguous postal codes can be combined

using regular expressions, e.g. "M1*" indicate all postal codes starting with "M1" (the marked triangle area in the map in Fig. 1). Fig. 1 shows the postal codes of downtown Toronto ON Canada. Suppose Joe lives in L5W and works in M1*. He inputs source area "M1*", destination area "L5W", and vice versa. This implies that he can serve the areas: from M1* to M1*, from L5W to L5W, from M1* to L5W, and from L5W to M1*. Two links (from M1* to L5W, and from L5W to M1*) can be added to the network. The service app broadcasts links data to all participating nodes with the app installed in the network.

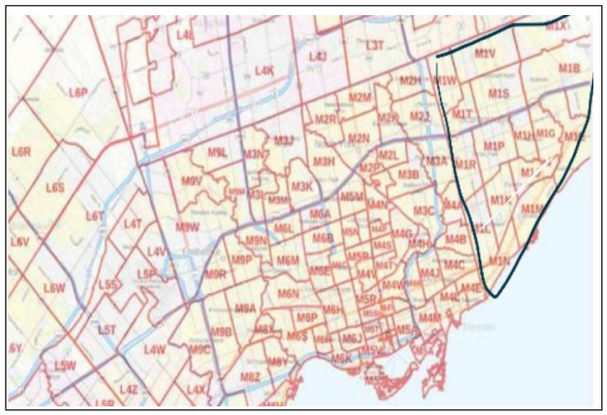

Fig. 1: The Postal Codes of Downtown Toronto ON Canada

A. Testing Connectivity

To determine connectivity of the network and then a route inside it, we model the network to a graph with multiple edges and vertices. Each vertex of the graph represents a node in the network and a service area in real world; each edge of the graph represents a connection in the network and a link between two areas in real world. If this graph is fully connected, it becomes a tree. In most cases, especially at early stage of the system, the graph is not fully connected, but has multiple connected components (each component being a tree). Connectivity management in graph theory determines number of connected components and vertices included in each connected component.

To the best of our knowledge, there is no existing algorithm in graph theory or networking to solve our Connectivity Management problem. A related problem and solution is Minimum Spanning Tree (MST). A minimum spanning tree or minimum weight spanning tree is "a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight" [3, 4]. Classical Kruskal's MST algorithm finds MST in a connected tree. Because of following reasons, we cannot use MST algorithm for our Connectivity Management problem directly:

Algorithm GenerateConnectivityTable

Input: A set of available links.

Output: An updated vector of connected tree, an updated hashmap containing nodes and their tree numbers.

1: Initialize a global vector of connected trees: Vector<Set<Area>>trees;

2: Initialize a global hashmap to look for tree number of a node: HashMap<Area, Integer>treeNumOfArea;

3: for all Area a do

4: add an entry (a, MAX_INT) in treeNumOfArea to indicate that its tree number is not determined;

5: end for

6: int currentTreeNum ← 0;

7: for all link <area1, area2>in linkSet do

8: call *ProcessALink* with link <area1, area2>

9: end for

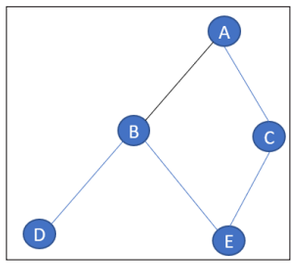

Fig. 2: The Shortest Path between Two Nodes Might not be in the MST

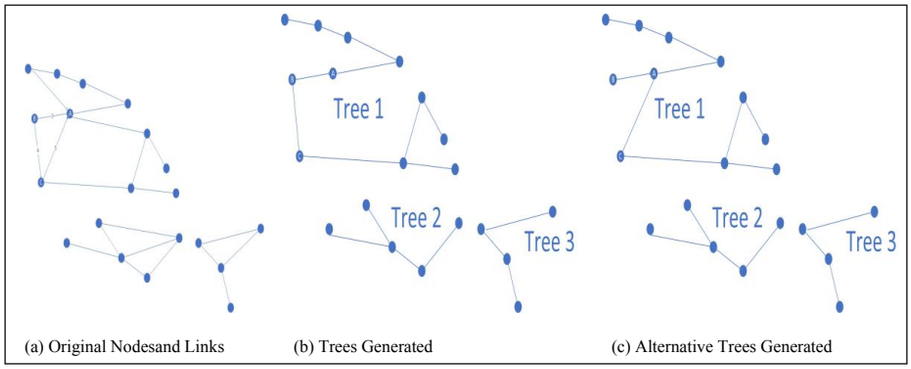

Fig. 3: An Example of Results of Algorithm *GenerateConnectivityTable*

- MST works in an already connected tree, and outputs a simplified tree. Our graph might not be a connected tree, and outputs one or multiple trees.
- MST takes into consideration path weights. Our Connectivity Management problem does not care about weights.

Please note that the shortest path between two nodes might not be in the MST, and the shortest path between nodes has nothing to do the MST either. For example, in Fig. 2, the MST is D-B-A-C-E, and Link B-E is not in the MST, while shortest path between B-C and shortest path between D-E go through B-E. When looking for a best route, we should ignore the MST.

Algorithm GenerateConnectivityTable

Input: A link from source node area1 to destination node area2, a reference to a vector of connected tree trees, a reference to a hashmap containing nodes and their tree numbers *treeNumOfArea;*

Output: An updated vector of connected tree, an updated hashmap containing nodes and their tr
Input: a link from source node area1 to destination node area2, a reference to a vector of connected tree *trees*, a reference to a hashmap containing nodes and their tree numbers *treeNumOfArea*;

Output: An updated vector of connected tree, an updated hashmap containing nodes and their tree numbers;

```
1: add an entry (area1, MAX_INT) in treeNumOfArea to indicate that its tree number is not determined;
2: add an entry (area2, MAX_INT) in treeNumOfArea to indicate that its tree number is not determined;
3: int currentTreeNum ← 0;
4: if area1 != area2 then
5:      {
6: area1_treeNum ← treeNumOfArea.get(area1);            //Look up area1's tree number intreeNumOfArea

7: area2_treeNum ← treeNumOfArea.get(area2);            //Look up area2's tree number intreeNumOfArea

8: if area1_treeNum == MAX_INT && area2_treeNum == MAX_INT then  .//area1 and area2 are not in a tree
9:          {
10:          SethNodei tree ← new SethNodei();                                          // create a new tree
11:          tree.add(area1);
12:          tree.add(area2);                                                  //add the two nodes to the new tree
13: trees.insertElementAt(tree, currentTreeNum);                  //insert tree to trees at indexcurrentTreeNum

14:          treeNumOfArea.put(area1, currentTreeNum);
15:          treeNumOfArea.put(area2, currentTreeNum);
16:          currentTreeNum++;
17:          }
18:     else if area1_treeNum == MAX_INT then                                       //area1 is not in a tree
19:          {
20:        area2_treeNum ← treeNumOfArea.Find(area2);
21:          trees.ElementAt(area2_treeNum).add(area1);                          //add area1 to area2's tree
22:          treeNumOfArea.put(area1, area2_treeNum);
23:          }
24:     else if area2_treeNum == MAX_INT then                                       //area2 is not in a tree
25:          {
26:        area1_treeNum ← treeNumOfArea.Find(area1);
27:          trees.ElementAt(area1_treeNum).add(area2);                          //add area1 to area2's tree
28:        treeNumOfArea.put(area2, area1_treeNum);
29:          }
30:     else                                                  //area1 and area2 are in the same or separate tree(s)
31:          {
32:        area1_treeNum ← treeNumOfArea.get(area1);
33:        area2_treeNum ← treeNumOfArea.get(area2);
34:         if area1_treeNum == area2_treeNum then
35:              continue;                                                      //already in same tree, skip this link
36:            else .          //Otherwise, harea1, area2i is connecting area1's and area2's trees, combine the two trees
37:              {
38:             for all Area a in trees.ElementAt(area2_treeNum) do
39:                  {                                                  //move all areas in area2's tree to area1's tree
40:                 add a to trees.ElementAt(area1_treeNum);
41:                 treeNumOfArea.put(area2, area1_treeNum);
42:                  }
43:              end for
44:             release trees.ElementAt(area2_treeNum);
45:              trees.remove(area2_treeNum);                                  //remove area2's tree from trees
46:              currentTreeNum–;
47:              }
48:          end if
49:          }
50:     end if
51:     }
52: end if
```

Algorithm CheckIfRouteExists

Input: sourceArea, destArea

Output: *true* if there is a route, *false* if there is no route between input areas

```
 1: if sourceArea.contains(destArea) then
 2:     return true;
 3: end if
 4: if trees.size>1 then                    //there are more than one trees in the network
 5:     {
 6:area1_treeNum ← treeNumOfArea.get(sourceArea);
 7:area2_treeNum ← treeNumOfArea.get(destArea);
 8: if area1_treeNum 6= area2_treeNum then
 9:         return false;                                                    //no route
10:       end if
11:     }
12: end if
13: return true;
```

Algorithm CheckIfRouteExists

Input: sourceArea, destArea

Output: *true* if there is a route, *false* if there is no route between input areas

```
 1: if sourceArea.contains(destArea) then
 2:     return true;
 3: end if
 4: if trees.size>1 then                    //there are more than one trees in the network
 5:     {
 6:area1_treeNum ← treeNumOfArea.get(sourceArea);
 7:area2_treeNum ← treeNumOfArea.get(destArea);
 8: if area1_treeNum 6= area2_treeNum then
 9:         return false;                                                    //no route
10:       end if
11:     }
12: end if
13: return true;
```

```
Algorithm   AddANewLink
    Input: sourceArea, destArea
    Output: new link needed to connect two trees containing sourceArea and destArea

 1: if !CheckIfRouteExists(sourceArea, destArea)) then
 2:     {
 3: long_int treeDistance ← MAX_Distance;
 4:    Link newLink;
 5:area1_treeNum ← treeNumOfArea.get(sourceArea);
 6:area2_treeNum ← treeNumOfArea.get(destArea);
 7: sourceAreas ← all areas in area1_treeNum;
 8: destAreas ← all areas in area2_treeNum;
 9:   for all Area a in sourceAreas do
10:       for all Area b in destAreas do
11:           Calculate d ← distance(a,b);
12:            if d <treeDistance then
13:                 {
14:                treeDistance ← d;
15:               newLink ← new Link(a,b);
16:                 }
17:             end if
18:         end for
19:     end for   //calculate and compare distances of all pairs between the two trees
20:    return newLink;
21:     }
22: end if
```

As mentioned above, in SMADS, there might be multiple separated trees instead of one connected tree. The goal is to generate a network-wide view of connected nodes, based on connections available in the network, and then be able to find routes between nodes. For connectivity management purpose, we do not really need to generate the minimum spanning tree, and only want to get all connected trees. So we can ignore edge weight and do not need to sort the edges before running Kruskal's MST algorithm. This makes significant difference in time complexity: our connectivity algorithm *GenerateConnectivityTable* is incremental. As the name suggests, *ProcessALink* algorithm processes the new link when a new link is added to the network with $O(|E|)$ time complexity ($|E|$ denotes number of edges in the graph). Kruskal's MST algorithm, need to check all links and sort them each time a new link is added, the time complexity is $O(|E|^2)$.

Each node runs *GenerateConnectivityTable* algorithm just once at first time to generate a connectivity table, and later runs *ProcessALink* algorithm when a new link is added to update the connectivity table. Inside the table, each area has a tree number stored in HashMap *treeNumOfArea*. Vector *trees* contains bunch of trees. In each tree, a set of nodes are connected. Size of the vector indicates total number of trees. Since we do not sort the links first in the algorithm, links are processed in random order, and different links could be selected in final trees. But the node set in each tree is always the same, and just this information is what we care about exactly. For example, running the algorithm against the graph indicated in Fig. 3(a) could generate three trees indicated in Fig. 3(b) (if Link (A, B) and (B, C) are processed first), or three trees indicated in Fig. 3(c) (if Link (A, B) and (A, C) are processed first).

When a new link is added, the app sends an update message to all agent nodes. Each node runs the algorithm to process it. If the new link connects two trees, they merge into one tree.

After a customer submits a service request, the service app receives the request from *sourceArea* to *destArea*, checks if there is a route between them to serve the request. The *CheckIfRouteExists* algorithm provides this functionality. If a route exists for the requested service, the service app determines a service agent in the source area, in a nearest-first or highest-rated-first fashion, or in a round-robin fashion, or with some other algorithm. Then the customer hands in the item to the suggested service agent physically.

B. Determining a Route

With a connectivity table that each service agent node has from algorithm *ProcessALink*, algorithm *CheckIfRouteExists* can determine if two nodes are in the same tree, which means if there is a route from *sourceArea* to *destArea*. Each service agent node also has connectivity database containing all

available links among service areas (neighbors of a node in the network). With this information, we propose a Fewest-stops Path algorithm *FindARoute* to find a route from *sourceArea* to *destArea*if there is one.

In graph and network theory, Dijkstra's Shortest Path algorithm (3) finds a path from one node to another. In SMADS system, for higher security and less overhead, we only consider the number of intermediate nodes, and do not take into account path cost or weight for now. In addition, Dijkstra's Shortest Path is mostly for pro-active routing. SMADS is essentially an ad-hoc network where the links among all areas are not stable and requests come only intermittently, we tend to use on-demand and reactive routing algorithms.

Algorithm *FindARoute* works in this way: while it traverses the tree in Breadth First Search (BFS), the route from the source node to a neighboring node (representing neighboring area) of the tree is stored into a stack. The stacks (representing routes) are added to the BFS queue. The first time the destination node is visited, the stacks contain a route to the node with the fewest intermediate nodes. The algorithm terminates at that moment normally. It is easy to see that the algorithm has time complexity of $O(|E|^2)$.

C. Adding New Links and Connecting Trees

In case two trees are not connected (there is no route to the destination), we might want to connect the two trees with human interference. We design an algorithm *AddANewLink* to suggest a new link connecting the two trees. To this end, it is natural that the new link should be of shortest distance between the two trees. Distance between two areas can be simply calculated using their postal codes, for example: $|H8N-M2K|=|H-M|-36^2+|8-2|-36+|N-K| = 6480+216+3 = 6699$. Obviously, this algorithm has time complexity of $O(|E|^2)$.

The system suggests a new link added to connect the two trees. Service agents and customer can now go through an auction and bid process to determine the provider and price of the new link. With the new link, the *FindRoute* algorithm finds a route between the two trees. Since this new link is the only link between the two trees, the new link must be included in the route. This feature is especially useful at the early stage of the system, or when the service destined is an out-of-the-way area. If multiple requests are brought up for the same link, the cost can be shared among them and the profit for the agent adding the link grows higher. This is the way how sharing economy works and makes profit.

V. Trust and Cooperation Ensurance

SMADS system is self-organizing into a network with people unknown to each other, who form a so-called Decentralized Autonomous Organization (DAO). Trust and cooperation among these unknown people are essential to successful operation of the system. Blockchain technology finds its niche in this situation and has made Bitcoin and Ethereum networks possible and successful, which sets up a paradigm for the type of systems in question. We apply blockchain technology to ensure trust and cooperation among unknown nodes.

A. Private Permissioned Blockchain and Mirrored Data

For SMADS system, we propose to build a private permissioned blockchain, because on the one hand, it has higher processing speed in a commercial context; and on the other hand, it has ability to hide transaction data from the public.

A user gets an account and installs the service app on a computer/tablet/smartphone, after completing registration on the service portal website. The service app then generates blockchain ledger and synchronizes it up among all appnodes.

Users of the app are divided into two types with different privileges: service providers and customers. Write access is granted only to service provider users that are to provide service in their registered areas and be identified trustworthy through a screening process. Screening process requires strict identity verification. Customer users have the option to register an account after going through a simple and fast registration process, and install the app. Then they can send a service request and have reading permission to track items in the system. They can always do these from the service portal website of course. Advantages of becoming a registered user and installing the app are local response speed and possible discount on service charge.

Each transaction needs to be signed with the private key of the service agent or private keys of two service agents involved in a handover. A signature can be verified using the public key of the service agent which is unique and implicit in the account name.

A typical series of transactions generated in a typical process flow of an item is like this: "Amy received Item #A001" (signed by Amy) $\Rightarrow$ "Amy forwarded Item #A001 to Joe" (signed by Amy and Joe) $\Rightarrow$ "Joe delivered Item #A001 to recipient Leo" (proof of delivery, signed by Joe and Leo).

Transaction data is stored in the blockchain ledger, including information of: checking in an item, checking out an item, adding a link, removing a link, updating a link, etc. The app creates and maintains several local database files derived from the transaction data in the blockchain. When a new block of transaction data is added to the blockchain, local database is updated on the fly. If there is a change in global view of connectivity, it is updated immediately. Local files are a mirrored mutable version of the global immutable version of the blockchain ledger. Removals and updates are allowed in local files, while they are just added as new transactions without actual modification.

Blockchain-mirrored local databases have the following advantages over blockchain:

- They support real cumulative Create/Read/Update/Delete operations while the blockchain only supports independent Read/Write operations.
- Data access and system response are much faster, user experience is better, because the operations are local.
- Number of records and volume of data size are smaller because Update/Delete operations are enabled.

However, the blockchain ledger is mandatory in the system because it is redundancy that ensures data consistence and cooperation among untrustworthy partners.

B. Service Reward and Incentive

The app broadcasts transactions as they are generated to all nodes in the network. As it is in a blockchain, at intervals, a block containing these transactions is generated and added to the distributed ledger. The SMADS system provides incentive rewards in a form of network-wide currency ---SMADS-coin --- to promote the service provider users to build the blocks competitively and to protect the blockchain from attackers, in a way similar to Bitcoin blockchain [5]. However, SMADS-coin is designed to mainly provide the transaction means for service agents to transfer service fee for relayed service to succeeding agents en route.

Of course, a service agent can also earn SMADS-coin by mining, especially for one that does not get many service requests, can help build and verify blocks in the blockchain. This helps balance income difference between different areas. A promotional amount of SMADS-coin could be offered to each registered service agent at registration time.

C. Proof-of-Credit

Traditional blockchains use Proof-of-Work (PoW) to ensure integrity of the blockchain. The idea is to deter attempts to manipulate the history of transaction data in blockchain with prohibitively high cost of the computation capacity requirement for most users. The drawback is that the cost or the energy consumption to build the blockchain is too high for honest users — the majority of the network, as well.

In SMADS system, the trust level between users is higher than Bitcoin for following reasons:

- Items need to be handed face to face, and many online frauds can be eliminated.
- It is based on a private permissioned blockchain, and only screened users who went through a strict on-boarding process have right to build blocks.
- Transaction initiators pay to succeeding service agents. There is no point to forge a transaction. Each hand-over needs to be signed by both users and is stored in the blockchain, and each agent en route has access to previous trans-action of the item.It is impossible to input an incorrect (shorter-distance) destination to under-pay others.

Because of these facts, we proposed a proof-of-credit (PoC) mechanism combined with Proof-of-work (PoW) to ensure integrity of blocks but avoid drawbacks of PoW: The difficulty level of PoW is set in a range (PoW_{min} to PoW_{max}). The system starts with PoW mode with PoW_{max}. Later, as agents earn credits from logistics service transactions, they can use credits in block mining to lower down difficulty level of PoW until PoW_{min} is reached. This mechanism encourages more trustworthy users to participate in blockchain maintenance service --- the more trustworthy; the less proof of work is needed. Algorithm *DetermineDifficultyLevel* shows how the difficulty level for a user to build a block is determined.

Algorithm DetermineDifficultyLevel

```
1: ......
2: difficultyLevel ← LEVEL_MAX
3: while user.credit>= 10 && difficultyLevel >LEVEL_MIN
   do
4:         {
5:         user.credit← user.credit -10;
6:         difficultyLevel ← difficultyLevel -1;
7:         }
8: end while
9: ......
```

VI. Simulation and Evaluation

We simulated a logistics service network with an Ethereum private blockchain. Basic account operation and logistics service operation transactions are propagated and stored on chain. An off-chain database containing link information is duplicated locally on each participating node. In this design, connectivity management is a totally local process. We implemented a Java application to manage a logistics service network with multiple areas most of which are connected with links. We simulated 16/25/36/64/81/100/144 links, each of which connects two random areas. We ran the *GenerateConnectivityTable* and *FindRoute* algorithms with above numbers of links respectively in our program and ran it on a desktop with Intel i7-3700 CPU@3.4GHz/64GB RAM and Windows Server 2016/JDK 9.0. The time performance was recorded and summarized in Fig. 4.

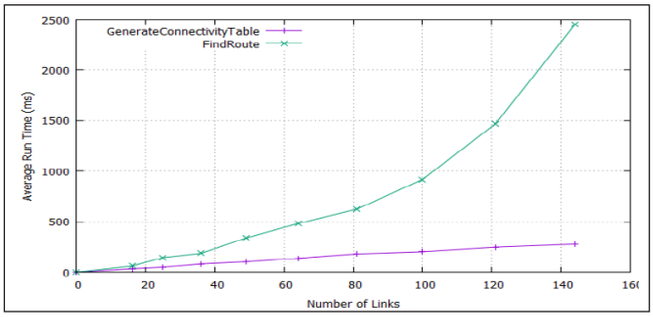

Fig. 4: Time Performance of Algorithm *GenerateConnectivityTable* and *FindRoute*

Note that for simplicity, irrelevant database retrieval operations are omitted, and links are simulated randomly on the fly. From the results we see that for up to 144 links in the network, time performance is good. Time complexity of *GenerateConnectivityTable* algorithm is approximately linear to total link number, and for *FindRoute* algorithm, it is approximately quadratic to total link number. Scalability is acceptable when total number of links increases.

To simulate blockchain operations of the network, we installed Truffle Framework on Windows Server 2016, and set up an Ethereum private chain. We designed User Interface (UI) webpages in HTML/Javascript/CSS, developed basic service smart contracts in Solidity and deployed them on blockchain. We used web3.js library to interact between webpages and Ethereum nodes on top of HTTP connections. Through the webpages, local operations, e.g. connectivity management, implemented in Java Jar files and on-chain smart contract operations, e.g. node and link addition/removal, can exchange data. In this way, connectivity data is synchronized up between local nodes and the blockchain. In the experiment, we simulated 4/8/12/16/20/24 full nodes that generate and propagate *AddAnAgentNode*/*AddALink*/*RemoveALink*/ *SendAnItem*/ *ReceiveAnItem* transactions, store entire blockchain data locally, and also mine blocks in the chain. We counted maximum number of transactions processed over a period of 5 minutes with different number of chain nodes, calculated throughput, namely number of trans-actions processed per second, and present them in Fig. 5(a). We measured latency time with different number of chain nodes, and present them in Fig. 5(b).

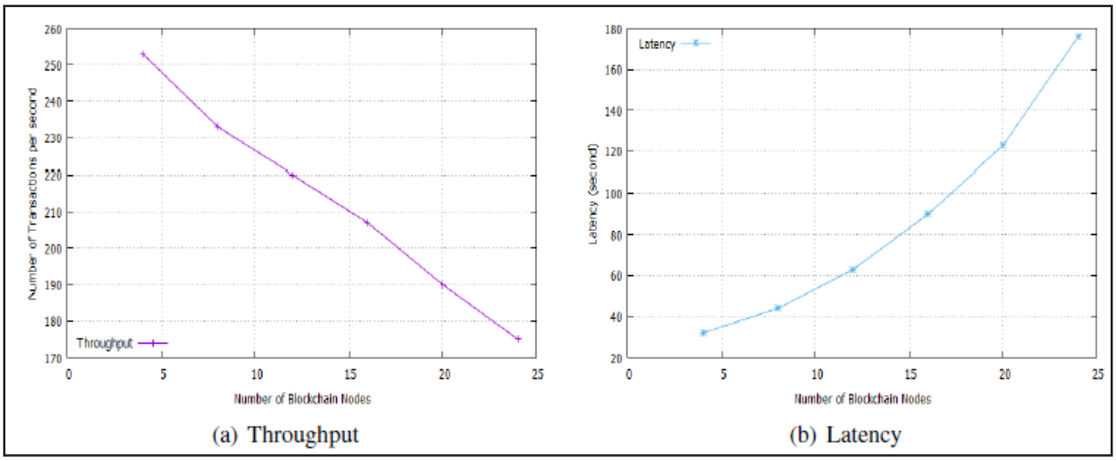

Fig. 5: Time Performance of the Blockchain Operations

The purpose of this experiment is to study scalability pattern of throughput and latency when number of chain nodes increases. Restricted by resources available, we could not simulate a large number of nodes in the chain. From the results, we can see that throughput decreases and latency increases as number of nodes increases, but they degrade in an acceptable and reasonable pattern. Please note that the actual amount of transactions or time does not make much significance, as throughput and latency are dependent on hardware/software resources and configurations of each chain node. We believe in real distributed blockchain environment, actual performance could be better, and scalability pattern should remain the same as we see here.

VII. Discussions and Future Work

We deliberately leave out some technical details or options to simplify the system framework in the above description. We will discuss them in this section.

- Item Tracking and Handover: We can use QR codes for item tracking and handover. A QR code tag is printed and attached to the item when it is accepted and input to SMADS system at the reception agent. The customer can scan the saved code in the smartphone app and track the item. The app reads the tag and triggers a smart contract transaction when the item is handed over to the next

agent. The smart contract transfers SMADS-coin from the former to the latter, and generates a transaction broadcast throughout the network. The transaction is written to the blockchain in the next interval.

- Routes Determination: In Subsection IV-B, if we run until the BFS queue is empty, store each stack of nodes (i.e. each route) into an additional ArrayListMultimap with reachable node as key, we can actually create a complete route table from the source area, including multiple routes from source to destination. We can assign each link a distance or cost or time attribute for route determination, we can then compare difference on these criteria between different routes, and find route with minimum distance or minimum cost or shortest time, instead of fewest stops.
- More on Proof-of-Credit: The essence of Proof-of-credit is basically cumulative Proof-of-work, as credit is earned from previous work.

We can manage credits in a way to encourage trust and cooperation of users in the system, for example, allowing a user to earn credit by building blocks, deducting credits from a user for certain misbehaviors, and suspending a user when the credit reaches a low threshold.

As future work, we can use a recommender system that employs variant service review/evaluation methods, such as the ones mentioned in [6, 7, 8, 9], to determine an agent's score. With the score, the credit can be adjusted, to combine proof of credit with customers' rating or review.

VIII. Conclusion

In light of new development in IoT, blockchain, ad-hoc/peer-to-peer network technologies, and inspired by various applications brought by them in real life, we come up with a notion of ITS that leads to SMADS (Self-organizing, Multi-relayed, Ad-hoc, Decentralized, and Sharing-economy-based) logistics networks. In this work, we propose the business model, design heuristic algorithms, pinpoint and address major technical issues in building such a system.

The idea of SMADS system can be applied to many services: A Uber$^+$/Lyft$^+$ transportation network can be built in which a passenger can travel from one city to another city with multiple drivers and vehicles, each serving only in a single city. A Fedex$^+$/UPS$^+$ parcel service network can be built in which parcels go through multiple segments of routes to the destination with multiple commuters with actual cost near to zero.

The far-reaching contributions of this work are: a novel idea to organize unrelated publicly accessible service segments or small networks into a larger logistics network, and solutions for availability and reliability issues in the network. The service network is not limited to logistics service, but has wide application in electronic commerce and Internet economy.

References

[1] "New Application for Ride Sharing". Accessed Jan. 2021. [Online]. Available: http://lazooz.org/

[2] "Arcade City: The Future of Ridesharing is Decentralized". Accessed: Jan. 2021. [Online]. Available: https://fee.org/articles/arcade-city-the-fu ture-of-ridesha ring-is-decentralized/

[3] T. C. Hu, "The maximum capacity route problem," *Operations Research*, vol. 9, no. 6, pp. 898-900, 1961.

[4] F. Chin, and D. Houck, "Algorithms for updating minimal spanning trees," *Journal of Computer and System Sciences*, vol. 16, no. 3, pp. 333-344, 1978.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] Z. Ji, H. Pi, W. Wei, B. Xiong, M. Woźniak, and R. Damasevicius, "Recommendation based on review texts and social communities: A hybrid model," *IEEE Access*, vol. 7, pp. 40416-40427, 2019.

[7] R. K. Behera, S. K. Rath, S. Misra, R. Damasevicius, and R. Maskeliunas, "Large scale community detection using a small world model," *Applied Sciences*, vol. 7, no. 11, pp. 1-18, 2017, Art. no. 1173.

[8] R. Colomo-Palacios, F. J. García-Peñalvo, V. Stantchev, and S. Misrad, "Towards a social and context-aware mobile recommendation system for tourism," *Pervasive and Mobile Computing*, vol. 38, part. 2, pp. 505-515, 2017.

[9] Rabbani, and Aslam, "Internet service selection using service association factor (SAF)," *Information Technology and Control*, vol. 48, no. 1, pp. 104-114, 2019.

[10] T. G. Crainica, M. Gendreaub, and J.-Y. Potvin, "Intelligent freight-transportation systems: Assessment and the contribution of operations research," *Transportation Research Part C: Emerging Technologies*, vol. 17, no. 6, pp. 541-557, 2009.

[11] M. Rahimi, A. Baboli, and Y. Rekik, "Sustainable inventory routing problem for perishable products by considering reverse logistic," *International Federation of Automatic Control*, vol. 49, no. 12, pp. 949-954, 2016.

[12] C. Cheng, P. Yang, M. Qi, and L.-M. Rousseau, "Modeling a green inventory routing problem with a heterogeneous fleet," *Transportation Research Part E: Logistics and Transportation Review*, vol. 97, pp. 97-112, Jan. 2017.

[13] Y. Li, F. Chu, C. Feng, C. Chu, and M. Zhou, "Integrated production inventory routing planning for intelligent

food logistics systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 3, pp. 867-878, 2019.

[14] H. Anisi, and H. Abdullah, "Efficient data reporting in intelligent transportation systems," *Networks and Spatial Economics*, vol. 16, no. 2, pp. 623-642, 2015.

[15] B. Y. Chen, X.-W. Chen, H.-P. Chen, and W. H. K. Lam, "Efficient algorithm for finding k shortest paths based on re-optimization technique," *Transportation Research Part E Logistics and Transportation Review*, vol. 133, pp. 1-13, 2020.

[16] E. Q. V. Martins, and M. M. B. Pascoal, "A new implementation of Yen's ranking loopless paths algorithm," *4OR*, vol. 1, no. 2, pp. 121-133, 2003.

[17] L. Yang, and X. Zhou, "Optimizing on-time arrival probability and percentile travel time for elementary path finding in time-dependent transportation networks: Linear mixed integer programming reformulations," *Transportation Research Part B: Methodological*, vol. 96, pp. 68-91, 2017.

[18] H. Yu, Z. Fang, F. Alan, T. Murray, H. Peng, and J. Chen, "Impact of oil price fluctuations on tanker maritime network structure and traffic flow changes," *Applied Energy*, vol. 237, pp. 390-403, 2019.

[19] Y.-S. Myung, and Y.-M. Yu, "Freight transportation network model with bundling option," *Transportation Research Part E: Logistics and Transportation Review*, vol. 133, pp. 1-11, 2020, Art. no. 101827.

[20] Z. Kuang, Z. Lian, J. W. Lien, and J. Zheng, "Serial and parallel duopoly competition in multi-segment transportation routes," *Transportation Research Part E: Logistics and Transportation Review*, vol. 133, pp. 1-23, 2020, Art. no. 101821.

[21] L. Wei, J. Zhang, R. Dai, and G. Zhu, "Green flexible vs. inflexible capacity strategies for duopoly," *Transportation Research Part E: Logistics and Transportation Review*, vol. 122, pp. 247-267, Feb. 2019.

[22] M. De, and B. C. Giri, "Modelling a closed-loop supply chain with a heterogeneous fleet under carbon emission reduction policy," *Transportation Research Part E Logistics and Transportation Review*, vol. 133, pp. 1-24, 2020, Art. no. 101813.

[23] S. Zhao, and K. Wang, "A framework for a decentralized and self-organized mail service network," *Proceedings of the 14th International Conference on Future Networks and Communications (FNC)*, 2019, pp. 327-334.