01 Jan 2003

Two Distributed Algorithms for E-ticket Validation Protocols for Mobile Clients

H. Mohanty

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

T. Suman Kumar Reddy

R. K. Ghosh

Recommended Citation

H. Mohanty et al., "Two Distributed Algorithms for E-ticket Validation Protocols for Mobile Clients," *Proceedings of the IEEE International Conference on E-Commerce, 2003*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2003.
The definitive version is available at https://doi.org/10.1109/COEC.2003.1210253

Two Distributed Algorithms for E-ticket Validation protocols for Mobile Clients

T. Suman Kumar Reddy* sumanreddy_t@yahoo.com
Hrushikesha Mohanty† hmcs@uohyd.ernet.in
R.K. Ghosh‡ r.ghosh@ieee.org
Sanjay Madria§ madrias@umr.edu

Abstract

E-Ticket validation problem has relevance in mobile computing environment because of the multiple submission of a ticket that is possible due to intermittent disconnections and mobility of hosts. Here, we propose protocols that are not only sensitive to disconnection but also to location. One of the proposed protocol is variant of the distributed protocol proposed in [7] for internet users. This shows a distributed protocol for static network can be restructured for distributed computation in mobile computing environment. We have also proposed another protocol that uses hierarchical location database of mobile hosts [9].

Keywords: *Mobile Computing Environment, Distributed Computing Environment, E-Ticket, Validation*

1 Introduction

In the age of internet, the traditional ticket based business can be conducted in cyberspace by using E-Tickets. However, while using E-Tickets, typical problems like atomicity, duplication and forgery of E-Tickets are to be considered seriously. Researchers have proposed system architectures for internet-based and smart card technology based [10] E-Ticket management systems. Such a system deals with issuance, validation, transfer as well as consumption of E-Tickets. E-Ticket validation includes issues like duplication and authenticity of E-Tickets. In cyber environment, untrusted users may resort to duplicate tickets and use those at different service providers. While a system assures services to genuine E-Ticket holders, it must guard against multiple uses of E-Tickets.

We are interested in problem due to multiple uses of E-Tickets which may happen either by untrusted users or due to malfunctioning of service providing systems. In later case, user being unaware of status of its E-Ticket may submit it again for services. In either case, service providers are required to accept a ticket *at least* and *at the most* for once. In [7], a two-phase protocol for on-line validation of E-tickets is proposed. The proposed distributed protocol in phase-I, validates an E- Ticket if, currently, it is not being used by any. The conflict occurs when more than one service providers simultaneously hold the copies of the same ticket. In phase-II of the protocol, the conflict is resolved and one of the service provider accepts the ticket while others reject. For example, a user while on move may like to avail digital services like music or games by submitting E-Tickets. In this case we also need to have an E-Ticket validation protocol for mobile users. Mobile computing environment is a kind of distributed computing environment with typical problems due to disconnections and lack of enough computing power. Particularly, communication using wireless channel is not error free as it is so in fixed network. In a distributed computing environment such a disconnection is treated as non-availability of a system. So, a distributed computation in mobile computing environment should be structured properly to handle the eventuality due to disconnections. In a similar attempt, we have restructured a reported validation protocol [7] making it suitable for execution in mobile computing environment. The proposed protocol other than ensuring *at least, at most* properties also deals with *eventual* property.The protocol ensures that inspite of disconnections a mobile user can eventually make use of its E-Ticket for at least and at most once.

2 System Model

In mobile computing environment [2], mobility of systems is possible while retaining network connections intact. A host that can move while retaining its network connections is called a Mobile Host (MH). A static/fixed host that co-ordinates activities of MHs is called Mobile Support Station (MSS). A cell is a logical or geographical coverage area under an MSS inhibited by several mobile hosts. All MHs that have identified themselves with a particular MSS are considered to be local to that MSS. A MH can directly com-

†* Department of Computer & Information Sciences, University of Hyderabad.
‡Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur.
§Department of Computer Science, University of Missouri-Rolla.

Proceedings of the IEEE International Conference on E-Commerce (CEC'03)
0-7695-1969-5/03 $17.00 © 2003 IEEE

municate with a MSS and vice versa only if the MH is physically located within the cell serviced by that MSS. At any given instant of time, a MH may (logically) belong to only one cell i.e. the cell where the MH is currently located.

Host mobility manifests itself as a migration of a MH from one cell to another. The mobile host mobility is asynchronous i.e. there is no bound on the time interval between a MH leaving its current cell and entering a new one. However a MH that leaves its current cell will eventually enter some cell in the system. To locate a mobile host in the system, we use a *hierarchical location database scheme* [8][9].

MHs are often disconnected from the rest of the network [1]. A disconnected MH can neither send nor receive messages. Disconnection in a mobile computing environment is distinct from failure: disconnections are elective by nature and so, a mobile host can inform the system of an impending disconnection prior to its occurrence and execute a disconnection protocol [2].

Our system model thus consists of two distinct sets of entities: a large number of mobile hosts $\lambda_H = \{H_i\}_{i=1..n}$ and a relatively fewer, but more powerful, MSSs $\lambda_S = \{S_i\}_{i=1..m}$. The model thus consists of a static/fixed network comprising of the MSSs and the communication paths between them, and a wireless network associated with each MSS for communicating with the mobile hosts located within its cell.

MSSs trust MHs and would like to validate a service request issued by a host. In the following sections, we describe a validation problem for E-Tickets.

3 E-Ticket Problem

E-Ticket – a digital version of a real world ticket is being used in internet to avail E-Services [5][6]. The use of E-Tickets can be extended to mobile hosts to enable them to avail services at their visiting places in exchange of such tickets. The users can acquire E-Tickets by purchasing them from any MSS or from another user who previously acquired them. Let I be the ticket issuer, O be the ticket owner, S be the service promised and V be the validation status. Then, an E-Ticket is defined as signed_I(I,S,O,V), where the phrase "signed_I" means the ticket is signed by the issuer's digital signature. To use an E-Ticket, it should be send for validation. E-Ticket validation is intended to prevent duplication and to ensure authenticity and integrity. Preventing duplication avoids multiple use of an E-Ticket by the same or different users. Ensuring authenticity and integrity guarantees that E-Tickets are only accepted if they have been issued by an authorized source, and have not been tampered with. In this paper, we are interested in the validation of E-Tickets. In mobile computing environment, MSS validates the E-Tickets. The validation process, called E-Ticket validation problem, results in acceptance or rejection of the E-Tickets. Generally speaking, validation of E-Tickets addresses two concerns: first, the same E-Ticket should not be accepted more than once, which can happen for example, when users distribute copies of their E-Tickets to other users. Second, there must be situations where E-Tickets are accepted at least by one MSS i.e. not all the MSSs reject the E-Ticket. The E-Ticket problem can be defined in short by the following two properties:

(P.1) If an MSS accepts an E-Ticket T and does not crash, then no other MSS accepts T and an MSS does not accept the same E-Ticket more than once.

(P.2) Let δ(T) be the set of MSS that validate the same E-Ticket T. If not all the MSS in δ(T) crash, then there is at least one MSS in δ(T) that accepts T.

4 E-Ticket Validation Protocols

4.1 A Two-phase E-Ticket Protocol for Internet Users

A two-phase E-Ticket protocol for distributed computing environment has appeared in [1]. The protocol is designed to validate E-Tickets in two phases. The protocol works efficiently only when E-Tickets are used only once. If the E-Tickets are used more than once, the validation process becomes complex and inefficient. The protocol is executed as follows:

Phase I: Once a server S_i receives an E-Ticket T from an user, it sends T to all the servers in the system to find out whether T has been already validated by one of them. When a server S_j receives T from S_i, if T has been already received, S_j sends a negative acknowledgment to S_i; else S_j sends a positive acknowledgment to S_i. After receiving the replies from more than half of the servers in the system, and of the received messages if there are no negative acknowledgments from any server, S_i accepts T in Phase 1; otherwise, S_i enters Phase II.

Phase II: Each server S_i that is executing Phase II handles two cases: (1) if all the servers that are engaged in validating T enters Phase II, then one of them accepts T and (2) if a server that is engaged in validating T is not in Phase II (which means it accepted T in Phase I), then all the servers running Phase II rejects T.

The main drawbacks of this protocol are:

- The protocol does not explain which server will accept T, if T has been issued to more than one server at the same time.
- A server S_i has to wait until it receives a majority of the replies and enter Phase II to reject that E-Ticket, even if some other server S_j accepted the ticket T much before S_i starts validating T.

There has been studies to adaptability of existing distributed algorithms to this new environment [3][4]. In this spirit we propose to study the suitability of the above discussed two-phase E-Ticket protocol for mobile computing environment. We also try to improve upon the previously proposed algorithm by suggesting remedies to the above said drawbacks.

4.2 A Two Phase Protocol for Mobile Users

4.2.1 overview of the protocol

Let λ_S be the set of MSS and λ_H be the set of mobile hosts in the environment. A host submits its E-Ticket to its MSS and on receipt of this message the validation protocol is initiated at the MSS. Unlike in the previous algorithm [7], in the proposed algorithm a MSS does not send validation request to all the other MSSs in the network. Instead, the MSS sends the requests to the MSSs located at the cells that were previously visited by the MH. Considering the uses of the hierarchy scheme [8][9] to maintain location databases of the mobile hosts, we restrict our MSSs of interest to those who are located in the subtree rooted at the least common ancestor of all the places visited by the host. This criteria is quite rational in case of mobile computing environment as the host can only use the E-Ticket at any of the MSS it has visited earlier. The concept of finding the MSS that an MH visited is implemented in the algorithm by using a function visited_list(mobile_host). The method search(H_i) is used to get the MSS, which is currently serving H_i and the method nearest(CurrentMSS, List) returns the MSS in the given list containing MSSs that is nearest to CurrentMSS.

The *Phase I* (lines 6-89) of the E-Ticket Validation Protocol at MSS S_i (refer appendix) is initiated by the user H_i by sending a message NEWTKT with E-Ticket T to the MSS S_i.

- To validate a ticket T sent by a mobile user H_i, MSS S_i sends a message (REQUEST_ACK, T, S_i, S_j) to all the MSSs visited by H_i i.e. δ_{H_i}.
- When a MSS S_j receives a message (REQUEST_ACK, T, S_i, S_j), if T is already present in its validated ticket set then S_j sends a reject acknowledgement to S_i; If S_j has already received a message of type (REQUEST_ACK, T, S_k, S_j), S_j sends a negative acknowledgement along with S_k's id to S_i, otherwise S_j sends a positive acknowledgement to S_i.
- Upon receiving a reply message (ACK, T, S_j, S_i), if the message is a reject acknowledgement, S_i rejects T; otherwise count the replies and waits. Once S_i receives the majority of the replies, and all the messages received by S_i are positively acknowledged, S_i accepts T. If there are negative acknowledgements, S_i starts Phase II of the protocol.

The *Phase II* (lines 90-113) of the protocol is executed when more than one MSS tries to validate the same E-Ticket T i.e. the same ticket is issued by H_i from several places. In this case, the MSS that is nearer to the MSS that is currently serving H_i accepts the ticket T and the others reject it.

At the end of the protocol the MSS S_i informs the decision on the validation of the E-Ticket, T, to the mobile host, H_i, holding T. To send the decision to the mobile host (lines 114-130), S_i searches for H_i, to find out if it switched the cell in the mean while, and delivers the message to the MSS that is currently serving H_i. If H_i is in disconnected mode, S_i waits for the reconnection of H_i and delivers the message. Subsequently, we present the algorithm in detail for reference in the appendix. The Messages transacted in the algorithm follow the format (Message Type, ticket, sender, receiver).

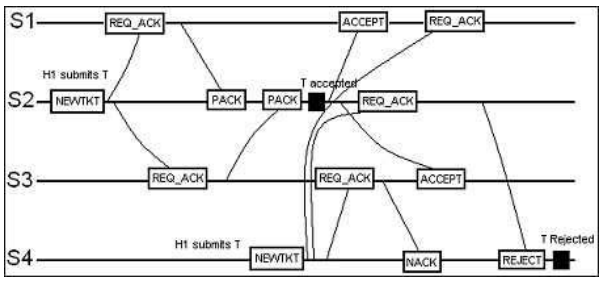

Figure 1. E-Ticket accepted in Phase I

Figures 1 and 2 depict the executions of the protocol. In figure 1, the mobile host H_1 submits a ticket T to the MSS S_2. H_1 visited S_1 and S_3 after acquiring the ticket T and before submitting the ticket T to S_2. So, S_2 sends a REQUESTACK message to S_1 and S_3. S_2 receives an ACK message with positive acknowledgement from S_1 and S_3. Therefore, S_2 accepts T. Before receiving an acceptance message from S_2, H_1 moves from its current cell to the cell served by S_4 and submits the same ticket T again. S_4 now sends the REQUESTACK to S_1, S_2 and S_3. S_4 receives a REJECT acknowledge from S_2, thus S_4 rejects T.

In figure 2, H_1 submits the ticket to S_2, switches the cell immediately and submits the same ticket to S_4. In this case a shown in the figure neither of the servers get the majority. so, both of them enter the phase II. Since, H_1 is currently closer to S_4, S_4 accepts the ticket and S_2 rejects the same.

4.2.2 Analysis of the Protocol

In this section, we present intuition behind the correctness of the protocol and the improved performance. E-Tickets can be accepted in Phase I or Phase II of the protocol. The

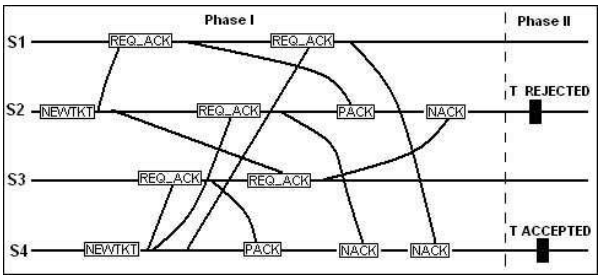

Figure 2. E-Ticket accepted in Phase II

algorithm guarantees that no two MSS will accept the same E-Ticket and an MSS does not accept the same E-Ticket more than once.

- Say a set of MSSs have been presented with ticket T by the MH H_i. S_i can accept the ticket only if it was not accepted before and S_i got the majority of the ACK messages with a positive ackstatus (lines 60-74), and so, S_j cannot get the majority of positive acknowledgments (lines 34-39) or it may even get a ACK message with a reject ackstatus(lines 27-31). So, S_j cannot accept T. So, the protocol guarantees that the E-Ticket validation and the validator is unique.

- Suppose MSSs S_i, S_j have not accepted T before and because of asynchrony in message passing and majority protocol in (line 66) both may intend to accept T. This contention is resolved at (lines 92-108) by selecting the MSS located nearer to H_i.

Furthermore, when a server S_i validates an E-Ticket T, it includes T in its vTkts, so that this T is not accepted again.

This protocol minimizes communication complexity using the mobile hosts' travel account. In Two-Phase E-Ticket validation protocol for distributed computing environment [7], the messages are sent to all the servers in the environment for acknowledgment of the ticket. In this protocol, the requests are sent only to the MSS that the MH previously visited after getting the E-Ticket T. There are three types of messages exchanged in the Phase I of the protocol.
REQUEST_ACK: Sent by the validating MSS to all the MSSs visited by the MH after acquiring the E-Ticket.
ACK: Replies sent by the MSSs to the MSS that sent the REQUEST_ACK.
ACCEPT: Intimation sent by the validating MSS to all the MSS about its acceptance of the Ticket.

During the Phase-I of the protocol if the ticket T is accepted then the following cases may arise. The message complexity entirely depends on these cases.

case 1 Mobile Host is residing in only one cell. In this case the number of messages exchanged will be zero.

case 2 Mobile Host is highly mobile and covered the entire network. In this case the number of messages exchanged will be $3(m-1)$ where m is the number of MSS in the environment.

case 3 This is an average case where the mobile host is limited to a few cells. The number of messages exchanged will be $3(\delta_{MH}-1)$ where $\delta_{MH} < m$ is the number of cells that MH visited.

If the ticket is accepted in the Phase II of the 2PE protocol for distributed computed environment [7] , which suggests that more than one server is associated in validating ticket T. Let δ_s be the number servers that are engaged. Now, in addition to the messages that are exchanged in Phase I, some messages will be exchanged among those δ_s MSSs. The number of these messages will be $(\delta_s - 1)^2$. But in our protocol, no such messages are involved in Phase II. This reduces the number of messages exchanged by a large number.

If the ticket is rejected at any part of our algorithm, the number of messages exchanged will be reduced. The rejection of tickets take place at lines 11, 55, 105 and 110. If rejection takes place at line 11, then the messages exchanged will be zero. In the other cases the *ACCEPT* message will be ignored.

Protocol ensures services for an E-Ticket holder from the nearest locality i.e. even if a MH issues an E-Ticket and switches cell, the service is provided to the MH at a place where it is currently located. The protocol also handles the prolonged disconnections by sending the validation to the MH at the place of reconnection.

If an MH issues an duplicate of an E-Ticket that has been validated previously, the protocol ensures that the algorithm is not executed completely but the ticket is rejected at the beginning of the phase I. This is achieved by storing the E-Tickets in the vTkts at each server. To optimize the size of the vTkts each E-Ticket may be given a timestamp after which the ticket automatically gets invalidated and its entry can be removed from vTkts.

4.3 Tree Based Protocol

Essentially, a validation protocol explores the possibility of multiple uses of a E-ticket by enquiring the service providers (MSSs) in the network. In the previous algorithm a network wide search has been reduced by restricting to servers at locations earlier visited by a mobile user. However, the difficulty in this strategy is due to the increasing length of the message carrying list of the locations visited by a mobile user. In order to overcome this problem we propose a scheme that uses a hierarchical distributed databases containing status of E-tickets. In order to validate a E-ticket queries are to be put to tree nodes only. It is not required

to a communicate length messages with the list of locations visited by a mobile use.

4.3.1 Overview of the Protocol

In cellular environment maintenance of locations of mobile users is an important issue as it has practical significance to locate and to forward messages to mobile users. Among the reported strategies [9] hierarchical region maintaining algorithm is of our interest. In this scheme, a geographical region is grouped into several sub-regions of different sizes and arranged in hierarchical fashion by designating a region of a given size to a particular level of a regional hierarchy.

A concept of hierarchy on a geographical region is defined in the following way. A mobile computing is a collection of cells and each such cell represents a leaf node (level 0) of a tree. In the next level (level 1) of hierarchy some regions are conceptualized where each region is a collection of some cells and is represented by a tree node of level 1. So, at i^{th} level a region is a collection of some trees of level (i-1). So, the root node represents the total area spanning over the environment. From this abstraction, we understand that a node in a tree represents certain area of the environment. At each node there is a server containing the identities of mobile users and E-tickets submitted by these users.

A mobile user residing in a cell S_3 (in fig. 3) when uses its E-ticket the server at the cell stores the information and the same is transmitted upward to ancestors upto root node. If the same user on changing its cell to S_6 produces the same E-ticket to the server at S_6 then the server can enquire status of this ticket to least common ancestor (LCA) of S_3 and S_6. In the worst case (e.g. when the mobile user switches from S_2 to S_3) status of a e-ticket will be certainly available at the root vertex. A formal presentation of the algorithm is given in Appendix B. An informal discussion of the proposed E-ticket protocol using hierarchical database is as follows.

- The algorithm is initiated with a NEWTKT message from the client to a server S_i (line 5). If S_i received the ticket before or has been validated before, S_i rejects the ticket (lines 8-12). Otherwise, it sends a REQUEST_ACK message to its immediate ancestor. And initializes a stack for storing the server addresses to trace the route of the ticket (lines 13-20).

- Once a server S_j receives a REQUEST_ACK message from S_i, S_j checks for the ticket in its vTkts. If T is in vTkts of S_j, it sends a negative acknowledgment with a ACK message along the route (lines 24-29). Otherwise, the request is passed on to its parent (lines 37-41). In the mean while, if the request reaches the root and root does not have T in its vTkts, then a positive acknowledgment is send through a ACK message. The vTkts of the root is updated with T (lines 30-36).

- If a node S_k receives a ACK message and if it is not the requestor, it updates its vTkts with T and passes on the message to its child that is in the route to the actual requestor (lines 60-64). If S_k is the actual request holder, based on the ACK message's ackStatus it accepts or rejects the tickets (lines 47-58).

- At the end of the protocol S_i, that accepted or rejected the ticket, informs the decision on the validation of the E-ticket, T, to the host, H_i, holding T. To send the decision to the host (lines 65-81), S_i searches for H_i, to find out if it switched the cell in the mean while, and delivers the message to the MSS that is currently serving H_i. If H_i is in disconnected mode, S_i waits for the reconnection of H_i and delivers the message.

4.3.2 Analysis of the protocol

In this section, we present intuition behind the correctness of the protocol. This protocol guarantees that the E-ticket is accepted *exactly once*.

Suppose, a client H_i submits the same ticket T, to a set of servers. If S_i accepts that ticket, an entry for that ticket is made at the root. Any other server S_j, which contacts the root will get a negative acknowledgement and has to reject the ticket. As mentioned in the previous section, if both the servers request the server at root at the same time, the request with the greater time stamp will be accepted and the other is rejected.

For example, consider a tree as shown in figure 3. The tree is constructed of eight servers based on their respective work loads. Say, the server S_1 may be having less number of clients than the servers S_4 and S_7.

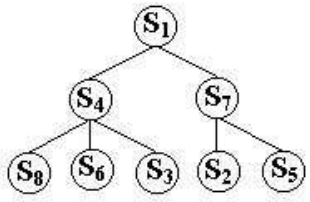

Figure 3. A tree constructed using eight server

Suppose, a client submits a ticket T, at S_4 and S_5. Due to message delay and other factors, say S_1, the root, is contacted first by S_4. So, S_1 gives an positive acknowledgement to S_5 and negative acknowledgement to S_4. If both the requests reach S_1 at the same time, since the client submitted the ticket to S_4 before S_5 and so the time stamp on the request of S_5 will be larger than S_4, S_1 gives a positive acknowledgement to S_5 and negative acknowledgement to S_4. Whom ever may accept the ticket, it is their duty to

search for the client who produced the ticket and provide the service to the client.

In this protocol, there are two types of messages that are transacted in the system. They are acknowledgement request messages from client that will be moving up the tree and the acknowledgement message that comes down the tree. At the worst case, where the Server that gets the ticket is at the highest level, the number of messages will be $2(h-1)$, where h is the height of the tree. In the best case, where the server that gets the ticket and the root being the same, the number of messages is 0. In an average case, where the server that gets the ticket is at a level l, the number of messages will be $2(h-l-1)$, where h is the height of the tree.

5 Comparison of the protocols

Our comparison assumes executions without failures, and the most common case where the same E-ticket is used once by the users. We compare the protocols based on (a) their resilience, and (b) the latency and (c) the number of messages exchanged to validate an E-ticket.

In the two-phase E-ticket protocol for the internet users, E-tickets are accepted in phase 1 after two communications steps: the initial acknowledgement request sent to all the servers by the server that receives the E-ticket, and the reply message sent by each server. This amounts to a latency of $L = 2\theta$, where θ is the maximum message delay, and $2(n-1)$ messages. To accept or reject an E-ticket, the protocol requires that a majority of the servers to reply. So, the resilience becomes $f < n/2$.

In the two-phase E-ticket protocol for the mobile users, E-tickets are also accepted in phase 1 after two communications steps: the initial acknowledgement request sent to all the servers, that the client visited previously, by the server that receives the E-ticket, and the reply message sent by each server. This amounts to a latency of $L < 2\theta$, since maximum message delay will be less than θ due to less number of servers visited. The total messages transacted will be $2(\delta_s - 1) + (n-1)$ messages where δ_s is the number of servers visited by the client. To accept or reject an E-ticket, the protocol requires that a majority of the servers to reply. So, the resilience here also is $f < n/2$.

In the tree based protocol, the latency amounts to $L << 2\theta$, since the maximum message delay will be the time taken to pass the request from a leaf node to the root and the time taken to pass the acknowledgement from root to a leaf node. So, the maximum number of messages are 2(h-1) where h is the height of the tree. Similarly, to accept or reject an E-ticket, the protocol requires that at least the servers upto the root should give response. So, the resilience in this case becomes $f < h$. Table 1 compares the cost of the three protocols.

E-ticket Protocols	**2PE Internet Users**	**2PE Mobile Users**	**Tree Protocol**
Distributed Environment	yes	no	yes
Mobile Environment	no	yes	yes
Resilience	$f < n/2$	$f < n/2$	$f < h$
Latency	2θ	$< 2\theta$	$<< 2\theta$
Messages	$2(n-1)$	$2(\delta_s - 1) + (n-1)$	$2(h-1)$

Table 1. Comparison of the protocols

6 Conclusions

In this paper we study E-Ticket validation problem and propose a validation algorithm that is suitable to run in mobile computing environment. The algorithm transacts less messages than the similar algorithm [7] proposed for the internet environment. The algorithm is sensitive to disconnection as well as location. In case of host disconnection, the algorithm waits to communicate validation result on reconnection of the host while major computational steps of the algorithm is performed on static network. On validation of a ticket, the algorithm cares to provide service at the cell nearest to the current location of the host. Here, in addition to proposing an algorithm for E-ticket validation for mobile computing environment, we show the possibility of restructuring of a distributed algorithm for this new computing paradigm.

References

[1] James J. Kistler and M. Satyanarayanan. "Disconnected Operation In The Coda File System," ACM Transactions on Computer Systems, Vol. 10, No. 1, February 1992.

[2] Tomasz Imielinski and B. R. Badrinath. "Data Management for Mobile Computing," Sigmod Record, Vol. 22, No. 1, March 1993.

[3] B. R. Badrinath, Arup Acharya and Tomasz Imielinski. "Impact of Mobility on Distributed Computations," ACM Operating Systems Review, Vol. 27, No. 2, April 1993.

[4] B. R. Badrinath, Arup Acharya and Tomasz Imielinski. "Structuring Distributed Algorithms for Mobile Hosts," 14^{th} International Conference on Distributed Computing Systems, June 1994.

[5] N. Asokan, P. A. Janson, M. Steiner and M. Waidner. "The State of art in the electronic payment systems," Computer, Vol. 30, No. 9, pg. 28-35, September 1997.

[6] B. Patel and J. Crowcroft. "Ticket Based Service Access for the Mobile User," The proceedings of the third annual ACM/IEEE international conference on Mobile Computing and Networking, Budapest, Hungary, September 26-30, 1997.

[7] Fernando Pedone. "A Two-Phase Highly-Available Protocol for Online Validation of E-Tickets," HP Laboratory Report, September 2000.

[8] Evaggelia Pitoura and Ioannis Fudos. "Distributed Location Databases for Tracking Highly Mobile Objects," The Computer Journal, Vol. 44, No. 2,2001.

[9] Evaggelia Pitoura and George Samaras. "Locating Objects in Mobile Computing," IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No. 4, July/August 2001.

[10] W.I. Siu and Z.S. Guo. "Smart card based Electronic Ticket Management System (ETMS)," http://www.sftw.umac.mo/∼utako/research

APPENDIX

A Algorithm for the Structured Two Phase Protocol

Algorithm to be executed at each MSS S_i

1. begin
2. rTkts $\leftarrow \phi$
3. vTkts $\leftarrow \phi$
4. aTkts $\leftarrow \phi$
5. end

6. **Event:** on Receive **(NEWTKT, T, H_i, S_i)**
7. **Action:**
8. begin
9. if ((T $\in$ rTkts) $\vee$ (T $\in$ vTkts)) then
10. begin
11. T.status $\leftarrow$ REJECT
12. send_to_MH (T, S_i, H_i)
13. end
14. else
15. begin
16. rTkts $\leftarrow$ rTkts $\cup$ {T}
17. $\delta_{H_i} \leftarrow$ Visited_List(H_i)
18. $\forall S_k \in \delta_{H_i}$, send (REQUEST_ACK, T, S_i, S_k)
19. rep_count$^T \leftarrow 0$
20. pack_count$^T \leftarrow 0$
21. validators$^T \leftarrow \phi$
22. end
23. end

24. **Event:** on Receive **(REQUEST_ACK, T, S_j, S_i)**
25. **Action:**
26. begin
27. if (T $\in$ vTkts) then
28. begin
29. T.ackStatus $\leftarrow$ REJECT
30. send (ACK, T, S_i, S_j)
31. end
32. else
33. begin
34. if($\exists S_k$,(S_k, T) $\in$ aTkts) then
35. begin
36. T.ackStatus $\leftarrow$ NACK
37. T.currentValidator $\leftarrow S_k$
38. send (ACK, T, S_i, S_j)
39. end
40. else
41. begin
42. aTkts $\leftarrow$ aTkts $\cup \{(S_j, T)\}$
43. T.ackStatus $\leftarrow$ PACK
44. send (ACK, T, S_i, S_j)
45. end
46. end
47. end

48. **Event:** on Receive **(ACK, T, S_j, S_i)**
49. **Action:**
50. begin
51. if(T $\notin$ vTkts) then
52. begin
53. if(T.ackStatus = REJECT) then
54. begin
55. T.status $\leftarrow$ REJECT
56. vTkts $\leftarrow$ vTkts $\cup$ {T}
57. send_to_MH (T, S_i, H_i)
58. end
59. else
60. begin
61. rep_count$^T \leftarrow$ rep_count^T + 1
62. if(T.ackStatus = PACK) then
63. pack_count$^T \leftarrow$ pack_count^T + 1
64. else
65. validators$^T \leftarrow$ validators$^T \cup$ {T.currentValidator}
66. if(rep_count$^T \geq \lceil$cardinality$(\delta_{H_i})/2\rceil$) then
67. begin
68. if(rep_count^T = pack_count^T) then
69. begin
70. T.status $\leftarrow$ ACCEPT
71. vTkts $\leftarrow$ vTkts $\cup$ {T}
72. send_to_MH (T, S_i, H_i)
73. $\forall S_k \in \lambda_S$, send (ACCEPT, T, S_i, S_k)
74. end
75. else
76. begin
77. start PhaseII
78. end
79. end
80. end
81. end
82. end

83. **Event:** on Receive **(ACCEPT, T, S_j, S_i)**
84. **Action:**
85. begin
86. vTkts $\leftarrow$ vTkts $\cup$ {T }
87. if ($\exists S_k$, (S_k, T) $\in$ aTkts) then
88. aTkts $\leftarrow$ aTkts – $\{(S_k, T)\}$
89. end

Phase II

```
90. begin
91.     if( T ∉ vTkts ) then
92.         begin
93.             currentMSS ← search(H_i)
94.             if( Nearest(validators^T ∪ {S_i}, currentMSS) = S_i) then
95.                 begin
96.                     T.status ← ACCEPT
97.                     send_to_MH(T, S_i, H_i)
98.                     vTkts ← vTkts ∪ {T}
99.                     ∀ S_k ∈ λ_S, send (ACCEPT, T, S_i, S_k)
100.                end
101.            else
102.                begin
103.                    T.status ← REJECT
104.                    vTkts ← vTkts ∪ {T}
105.                    send_to_MH(T, S_i, H_i)
106.                end
107.        end
108.    else
109.        begin
110.            T.status ← REJECT
111.            send_to_MH(T, S_i, H_i)
112.        end
113. end

114. send_to_MH (T, S_i, H_i)
115. begin
116.    if ((isUnder(H_i, S_i) = true) ∧ (isAlive (H_i) = true)) then
117.        begin
118.            deliver(T, S_i, H_i)
119.        end
120.    else if ((isUnder(H_i,S_i) = true) ∧ (isAlive (H_i) = false)) then
121.        begin
122.            wait(Δt)
123.            send_to_MH ( T, S_i, H_i)
124.        end
125.    else if ( isUnder(H_i, S_i) = false ) then
126.        begin
127.            currentMSS ← search(H_i)
128.            deliver_to_MSS( T, H_i, S_i, currentMSS)
129.        end
130. end
```

B Algorithm for the Tree based Protocol

Algorithm to be executed at each MSS S_i

```
1. begin
2.    rTkts^i ← φ
3.    vTkts^i ← φ
4. end

5. Event: on Receive (NEWTKT, T, H_i, S_i)
6. Action:
7. begin
8.    if ((T ∈ rTkts^i) ∨ (T ∈ vTkts^i))
9.        begin
10.           T.status ← REJECT
11.           send_to_MH(T, S_i, T.owner)
12.       end
13.   else
14.       begin
15.           rTkts^i ← rTkts^i ∪ {T}
16.           route_i^T ← φ
17.           route_i^T .push(S_i)
18.           send (REQUEST_ACK, T, route_i^T, S_i, S_i.parent)
19.       end
20. end

21. Event: on Receive (REQUEST_ACK, T, route_j^T, S_k, S_i)
22. Action:
23. begin
24.   if (T ∈ vTkts^i)
25.       begin
26.           T.ackStatus ← NACK
27.           Child ← route_j^T .pop()
28.           send (ACK,T,route_j^T, S_i, Child)
29.       end
30.   else if (S_i.parent = φ)
31.       begin
32.           T.ackStatus ← PACK
33.           vTkts^i ← vTkts^i ∪ {T}
34.           Child ← route_j^T .pop()
35.           send (ACK,T,route_j^T, S_i, Child)
36.       end
37.   else
38.       begin
39.           route_j^T .push(S_i)
40.           send (REQUEST_ACK, T, route_j^T, S_i, S_i.parent)
41.       end
42. end

43. Event: on Receive (ACK, T, route_j^T, S_k, S_i)
44. Action:
45. begin
46.   vTkts^i ← vTkts^i ∪ {T}
47.   if (route_j^T .isEmpty() = TRUE)
48.       begin
49.           if (T.ackStatus = NACK)
50.               begin
51.                   T.status ← REJECT
52.               end
53.           else
54.               begin
55.                   T.status ← ACCEPT
56.               end
57.           send_to_MH(T, S_i, T.owner)
58.       end
59.   else
60.       begin
61.           Child ← route_j^T .pop()
62.           send (ACK,T,route_j^T, S_i, Child)
63.       end
64. end

65. send_to_MH (T, S_i, H_i)
66. begin
67.   if ((isUnder(H_i, S_i) = true) ∧ (isAlive (H_i) = true)) then
68.       begin
69.           deliver(T, S_i, H_i)
70.       end
71.   else if ((isUnder(H_i,S_i) = true) ∧ (isAlive (H_i) = false)) then
72.       begin
73.           wait(Δt)
74.           send_to_MH ( T, S_i, H_i)
75.       end
76.   else if ( isUnder(H_i, S_i) = false ) then
77.       begin
78.           currentMSS ← search(H_i)
79.           deliver_to_MSS( T, H_i, S_i, currentMSS)
80.       end
81. end
```