Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Quick Consensus Through Early Disposal of Faulty Processes

Mamata Dalui, Bidesh Chakraborty and Biplab K Sikdar

Department of Computer Sc and Tech, Bengal Engineering and Sc University, WB, India 711103
mamata_dalui@rediffmail.com, bidesh.me@gmail.com, biplab@cs.becs.ac.in

Abstract— **This work reports an efficient solution for reaching agreement (consensus) among the processes of a distributed system. The better efficiency is achieved through early disposal of faulty processes while approaching for a consensus. The introduced network partitioning scheme further facilitates the progress by reducing the message exchange overhead. Simulation results establish that the proposed solution significantly reduces the message exchange complexity, in comparison to the schemes reported so far, simultaneously ensuring the fault-tolerance ability of a system as that of the known best results.**

Keywords: Reaching agreement, Byzantine agreement, Consensus, Early stopping.

I. Introduction

In a distributed system, it is often needed to reach a common decision (consensus) among the processes. The participating processes can compete or cooperate among themselves for such consensus. This results in voluminous information exchanges [1], [4], [5]. A failure prone system further increases the message exchanges to disregard the participation of faulty processes in decision making.

A number of solutions [1], [3], [4], so far been reported, for reaching agreement with the target to reduce message exchanges as well as improved fault tolerance. However, none of these can achieve optimality in terms of both the fault tolerance through bare minimum message exchanges and quick termination of the agreement procedure. Further, a very few of these have taken care of simultaneous existence of the dormant and crash faults with reservations.

The above scenario motivates us to study the issues related to an agreement protocol and then to propose a solution that can overcome the limitations of existing schemes reported in [1], [2], [3], [4], [5].

The proposed solution is set to achieve better efficiency through early disposal of faulty processes while trying to reach an agreement (consensus). The consensus is achieved after two rounds instead of arbitrary number of rounds required in the SMBTC [5]. This ensures minimum message exchanges among the processes without sacrificing the fault tolerance. Further, a network partitioning scheme is introduced to handle large networks [6]. The reported experimental result establishes effectiveness of the proposed solution, in terms of message exchange overhead, fault tolerance as well as the time required to reach a consensus. The next section provides the topics relevant for our current work.

II. The Mortal Byzantine

In Byzantine agreement, one process (called the initiator) floats a value that is to be agreed upon. All the processes then communicate (exchange messages) and the non-faulty processes agree on the same value. If the initiator is non-faulty, all non-faulty processes agree on the initiator's proposed value. On the other hand, in consensus, each of the participating processes has its own initial value. The processes exchange those values among themselves and finally all the non-faulty processes agree on the same value.

The algorithm proposed in [1] solves the Byzantine agreement problem for n $\geq$ 3t+1 processes, where n is the number of processes and t can be the at most tolerable faults. The performance of this algorithm is measured in terms of message exchange rounds. It is shown in [2] that, with n $\geq$ 3t+1 processes, any non-authenticated Byzantine agreement protocol requires at least t+1 rounds in worst case. In [3], it is min{f+2, t+1}, if there are f $\leq$ t faulty processes in the system.

The Synchronous Mortal Byzantine Tolerant Consensus (SMBTC) scheme [5] achieves a bound n> 2t with the assumption that the faulty processes crash within a finite time. That is, in a system of 5 processes (P_1, P_2,... , P_5), at most 2 (P_3 and P_5) can be faulty. To reach an agreement, a process (P_1) initiates the process of consensus by sending its proposed value and decision value to all the participating processes. Then it passes through some phases each consisting of two major steps called rounds.

In the first round of a phase, each process sends its proposed and decision values to all. If any non-faulty process (P_2) identifies that the message from a process (P_5) is lost, the non-faulty process (P_2) detects it (P_5)

This research work is supported by the Sponsored Cellular Automata Research Projects, Bengal Engineering and Science University, Shibpur, West Bengal, India - 711103.

as faulty. In the second round, each process sends the records it is having, at the end of first round, to all others. A non-faulty process checks the received information from processes considered non-faulty and tries to decide.

If a faulty process (P_3) is not yet crashed (i.e. to be detected) and sends conflicting values, the non-faulty process (P_2) can not decide and a new phase is started.

The agreement process ends when all the non-faulty processes decide on the same value.

The SMBTC is optimal in terms of fault tolerance, but not in terms of number of message exchanges. For a moderately large network, say with 50 processes, this may require $\simeq$ 58333 message exchanges. We have proposed a scheme (*agreement-at-partition*) in [7] that reduces the number of message exchanges. However, the SMBTC and *agreement-at-partition* require arbitrary number of rounds for a consensus. The solution proposed in the current work resolves the issue through early disposal of faulty processes as well as partitioning of the network while tries to reach a common agreement. Further, we consider simultaneous existence of both the dormant and crash faults. This maximizes the fault tolerance capability of a system.

III. The Proposed Scheme

In a large network, reaching a common decision (consensus) among the processes, requires huge message exchange overhead. Due to presence of faulty processes, a consensus scheme may take several rounds (set of steps) [4], [5] to ensure that the decision taken by the non-faulty processes is correct. An early identification of faulty processes can speed up the agreement process. In the current work, we develop a scheme that solves consensus in exactly two rounds (quick-consensus) through minimum message exchanges among the participating processes. The proposed scheme can also take care of the system even when there are dormant faults and crash faults (malicious faults). A partitioning scheme is introduced with the target to further reduction in message exchange overhead while reaching an agreement. It results in low congestion and high network throughput and, thereby, avoids broadcast storm in a large distributed system.

The following subsection reports the proposed quick consensus scheme. The introduction of partitioning in this process is described in Subsection III-B.

A. Quick consensus

The proposed consensus scheme, exploiting the early disposal of faulty processes from decision making, passes through two rounds. In first round, all the processes exchange their own initial values. At the start of second round every process (P_i) knows the initial values of all the n processes. This is stored in a vector (say V_i). These V_is vectors formed at the processes (P_is), at the end of first round, are shown in Fig.1(a).

Each (P_i) of the processes then sends its vector (V_i) to all the processes. At this point every process (P_i) is having an array of vectors (2-d array) containing initial value of the process (P_i), sent by the process P_i as well as the other processes. These 2-d arrays (initial-value-sets) of all the processes are presented in Fig.1 (b).

In a fault-free network, the 2-d arrays (initial-value-sets) constructed at process sites are the same. In a system with faulty/non-faulty processes, a non-faulty process (P_i) can identify the dormant faulty processes as well as the crash faulty processes by investigating each row and column of its initial-value-set$_i$. The different entries (1/0) in column j signify that the process P_j sends different initial values to different processes i.e. P_j is a dormant faulty process. If all the entries in row k are not equal to 0 or 1 (say, M), then it signifies that P_k doesn't send any value i.e P_k is a crash faulty process. In Fig.1 (b) P_3 is a dormant faulty process and P_5 is a crash faulty process. Any non-faulty process (say, P_1) detects all the dormant and/or crash faulty processes (i.e. P_3 and P_5) at the end of second round and eliminates the values sent by the faulty process(es) by discarding/ignoring the corresponding row(s) and column(s) (i.e. 3rd row & column and 5th row & column) from its initial-value-set (initial-value-set$_1$). During the process, a non-faulty process (P_1) keeps count (k) of the faulty processes it detects.

If a non-faulty process (P_1) sees that the number of valid rows in its initial-value-set (initial-value-set$_1$) is greater than the number of faulty processes (k), then it (P_1) decides on the majority of 0s or 1s in the row that corresponds to the process (P_1).

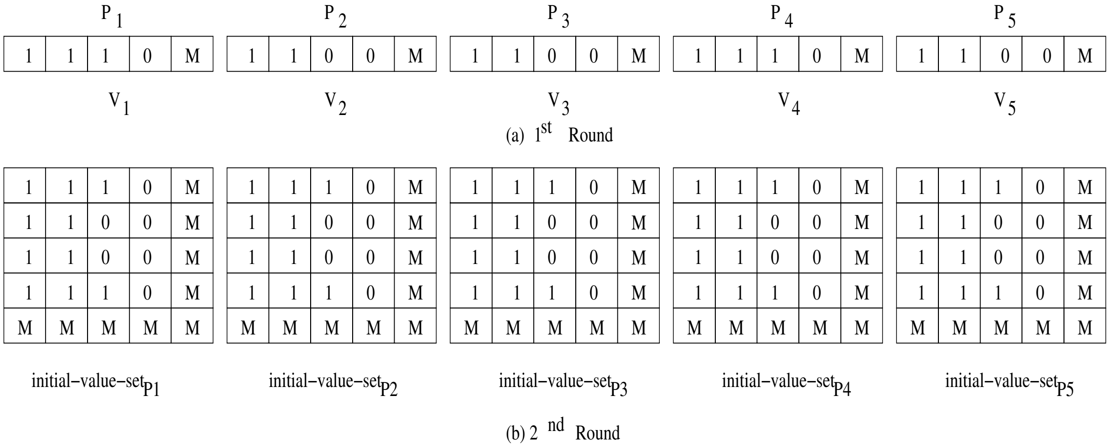
The solution to reach a consensus among the processes is described next. The following assumptions are taken while devising the solution:

1. Links among the processes/nodes are reliable and these are not introducing any delay.
2. F_d is the total number of dormant faulty processes. A dormant faulty process can send different values to different processes.
3. F_c is the total number of crash faulty processes. Such fault occurs when a process is permanently crashed and a faulty process crashes within a finite time interval.
4. A faulty process does not alter the values it received from others.
5. T_f=k=F_d+F_c is the total number of faulty processes.

Algorithm 1: quick-consensus

(i) $PL_i \equiv$ array of process ids (process-list) stored at each process i

(ii) $n \times n$ initial-value-set$_i$ array (n = number of



P_1	1	1	1	0	M
P_2	1	1	0	0	M
P_3	1	1	0	0	M
P_4	1	1	1	0	M
P_5	1	1	0	0	M

V_1 V_2 V_3 V_4 V_5

(a) 1st Round

initial−value−set$_{P1}$				
1	1	1	0	M
1	1	0	0	M
1	1	0	0	M
1	1	1	0	M
M	M	M	M	M

initial−value−set$_{P2}$				
1	1	1	0	M
1	1	0	0	M
1	1	0	0	M
1	1	1	0	M
M	M	M	M	M

initial−value−set$_{P3}$				
1	1	1	0	M
1	1	0	0	M
1	1	0	0	M
1	1	1	0	M
M	M	M	M	M

initial−value−set$_{P4}$				
1	1	1	0	M
1	1	0	0	M
1	1	0	0	M
1	1	1	0	M
M	M	M	M	M

initial−value−set$_{P5}$				
1	1	1	0	M
1	1	0	0	M
1	1	0	0	M
1	1	1	0	M
M	M	M	M	M

(b) 2 nd Round

Fig. 1. Initial value sets

processes) stored at each process i

(iii) decision-value$_i$ stored at each process i

Input: processes, initial-values and PL for all processes.

Output: decision-value.

```
1. initialize initial-value-set_i[i][i] = 0/1 ∀i,
   initialize NFL_i= |PL_i| ∀i=1 to n
2. first round: each process sends its initial-value to all.
   if a process p doesn't receive message from P
   then updates initial-value-set_p[p][P] = M in p's arrays. PL_p[P]=M
3. second round: each process i sends the initial-value-set_i[i] to all
   if process p doesn't receive message from P
   then initial-value-set_p[P][i = 1 to n] = M and PL_p[P]=M
   for j=1 to n
   if column j of initial-value-set_p not contains all 0s or all 1s (ignoring entry M)
   then report P_j is faulty and set all entries of row j and column j as M
   end for loop
   NFL_p=NFL_p-k // k= number of rows with all M
   if NFL_p > k, then
   for p=1 to n
   ignore row p if all its columns are M
   else decision_p= majority of 0/1 in row p of initial-value-set_p
   return decision-value=decision_p
   end for loop
```

The proof of correctness of Algorithm1 follows from [5].

B. Improvement through partitioning

A network of n processors/processes is partitioned into a number, say g, of groups. A process of a group (G$_i$) is selected as the co-ordinator/leader and initiates the process of consensus (Algorithm 1) within G$_i$. The rounds of message exchanges to reach an agreement among the members of G$_i$ is called local round. The agreement reached is called *local agreement*. Once the local rounds for all the groups are completed, one process (leader) from each group G$_i$ then participates, considering the weighted local decision value x_i, in a process to reach the final (global) agreement. The weighted decision value is represented as x_i= (d,w), where d (decision value) $\in \{0, 1\}$ and w = number of non-faulty processes in the group G_i.

At the initialization phase of network partitioning, a randomly selected process P_r (initiator) broadcasts an initialization message. After receiving it, each process of the system initializes a counter to '1' and starts incrementing. The P_r then further broadcasts g tokens. Each token can be received by one and only one process. The processes holding the tokens are the leaders. Each leader logically forms a group of n/g-1 processes.

Algorithm 2: global-agreement

Input: leaders of the groups and their weighted local-decision-value.

Output: global-decision-value.

```
1. if for any k, $1 < k < g$ (g is the number of groups),
   local round for group $G_k$ is finished
   leader P of $G_k$ sets a random timer RT[P] and starts decrementing it
```

2. if P doesn't receive any advertisement from an initiator of global round and RT[P] = 0
   then P sends local-decision$_P$ and weight (i.e. no. of non-faulty processes in G$_k$) to all the leaders, else go to step 2
3. if a leader Q $\in$ G$_i$ receives initialization message from P then Q resets RT[Q]=0
4. P initiates *Algorithm 1* (quick-consensus) considering the set of leaders as a group and decision value computed is the global-decision-value
5. the leader L of each group conveys global-decision value to all processes belonging to its group G_l
6. return global-decision-value

In global round (Algorithm 2), if weighted local-decision-value is sent by the faulty leader of group G, there may be a chance of mishap. However, the proposed scheme can mask off such faults as the faulty leader is detected in the local round. Once the faulty leader is detected, a new leader is selected from G following a cellular automata based election algorithm reported in [8]. The new leader participates in global round by sending a reply to the initiation message [7].

IV. Performance Evaluation

This section reports performance evaluation of our proposed scheme in terms of number of message exchanges to reach a consensus. The analytical results are shown in the following subsection.

A. Analytical results

The number of message exchanges required in the *quick-consensus* scheme (Algorithm 1 & 2) is

$$m = \frac{2n^2}{g} + 2g^2 + (n - g - T_f) \qquad (1)$$

where the 1st ($\frac{2n^2}{g}$) and 2nd ($2g^2$) terms represent the number of message exchanges required in local and global rounds respectively. The 3rd term $(n - g - T_f)$ appears due to the fact that after completion of global round, the leaders of each group informs the global decision value to all the non-faulty processes belonging to that group. Therefore,

$\frac{dm}{dg} = \frac{-2n^2}{g^2} + 4g - 1$

and $\frac{d^2m}{dg^2} = \frac{4n^2}{g^3} + 4 > 0$ (since g and n, both positive).

For an optimum m, $\frac{dm}{dg} = \frac{-2n^2}{g^2} + 4g - 1 = 0$

i.e., $g^3 = \frac{g^2}{4} + \frac{n^2}{2}$ i.e., $g = \frac{1}{4} + \frac{n^2}{2g^2}$

That is,

$$g \simeq \sqrt[3]{\frac{1}{2}n^2} \qquad (2)$$

B. Simulation results

The performance of proposed solution scheme is compared with the SMBTC [5] and *agreement-at-partition* [7], in terms of message exchange overhead, to reach an agreement.

Table I shows the number of message exchanges required by the SMBTC and the proposed scheme (quick-consensus) without partitioning of network. The first column represents the number of participating processes (n). The second column represents the number of tolerable faulty processes. Since the faulty processes crash randomly, we have taken three sets of observations for different crash times. This is shown in column 3 and the message exchanges in column 4. The average of these three, for an n, is provided in column 5. The last column reports the message exchanges required by the proposed scheme quick-consensus. Fig.2 displays the performance comparison (in terms of message exchanges) of these two schemes.

The comparison results of quick-consensus with partitioning (global-agreement) and *agreement-at-partition* are shown in Table II. The columns A represent the number of processes in the system. The columns B (number of tolerable faulty processes (fPr)), C (number of message exchanges) & D (average message exchanges) show the results of SMBTC [5]. The columns E-I are for the *agreement-at-partition* scheme of [7] and our proposed (quick-consensus) scheme with partitioning. We have taken three sets of data for random crash times (column C and G) of faulty processes. The average of these three is reported in the columns D and H. The number of groups (g) and number of tolerable faulty processes (fPr) in *agreement-at-partition* and the proposed scheme (quick-consensus with partitioning) is shown in the columns E and F.

The results shown in the tables point to the fact that the proposed solution maximizes the fault tolerance capability of a system. The schemes (without partitioning and with partitioning) achieve optimality in terms of fault tolerance as that in SMBTC [5] and *agreement-at-partition* [7] and significantly reduce the message exchange overhead (Fig3). Both the SMBTC and *agreement-at-partition* take arbitrary number of rounds to reach an agreement depending on the crash times of the faulty processes. The proposed schemes, on the other hand, can dispose off the faulty processes from consideration and completes the consensus process only in two rounds.

The impact of number of partitions is illustrated in Fig.4. It directs that the number of message exchanges reduces with the increase in number of partitions. This continues up to a threshold limit. If the number of partitions is close to the number of processes, the partitioning can't reduce the message exchange overhead. This can be better explained from the analytical results. From the expression of m (Equation 1), we can observe that, for g << n, the 1st term prevails and the

TABLE I
PERFORMANCE EVALUATION I

No. of Proc	No. of faulty	SMBTC [5]			quickcon
		Obs. for diff. crash times	# Msg	Avg Msg	# Msg
15	7	1	2250	2250	450
		2	2250		
		3	2250		
20	9	1	4800	4000	800
		2	4000		
		3	3200		
25	12	1	7500	6667	1250
		2	6250		
		3	6250		
50	24	1	65000	58333	5000
		2	65000		
		3	45000		

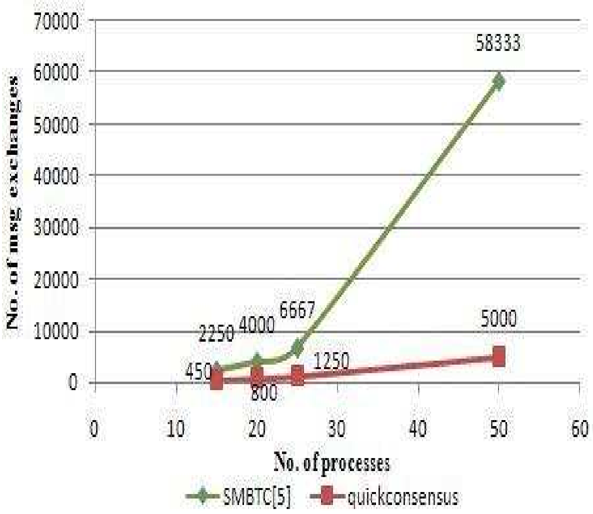


Fig. 2. Performance Comparison I

contribution of 2^{nd} term is negligible. If we go on increasing the g, contribution of the 1^{st} term decreases but the 2^{nd} term becomes more prominent and the 3^{rd} term decreases. So, there must exist a minima in the curve for m (since $\frac{d^2m}{dg^2} > 0$).

From Table II, it can be observed that the simulation results conform to the analytical results. Graph shown in Fig.4 displays that for 50 processes if the number of groups is 11, the number of message exchanges is minimum (664). The expression for g, shown in Equation 2, also conforms with that value. That is,

$$g = \sqrt[3]{\frac{1}{2}50^2} \simeq 10.7696 \simeq 11.$$

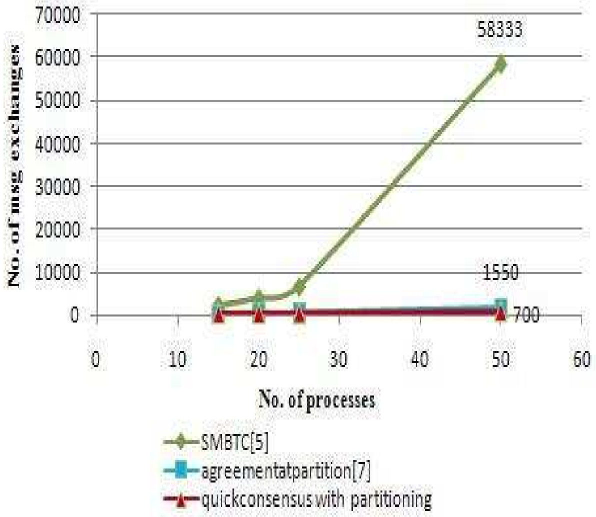


Fig. 3. Performance Comparison II

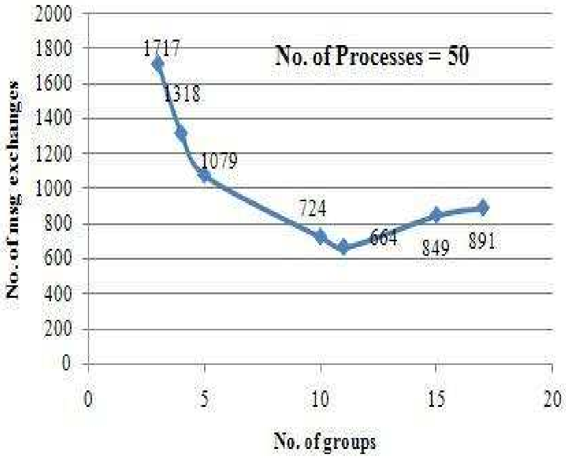


Fig. 4. Performance comparison III

V. Conclusion

This work addresses the issue of reaching agreement (consensus) in a distributed system. A scheme has been proposed to ensure early disposal of faulty processes that leads to a quick solution. A network partitioning scheme is also introduced for further reduction in message exchange overhead. The simulation results establish that the proposed scheme can drastically reduce the message complexity in comparison to the state-of-the-art solutions.

TABLE II

Performance Evaluation II

	SMBTC [5]					scheme[7]		quickcon
# Proc A	# fPr B	# Msg C	Avg Msg D	# Gr E	# fPr F	# Msg G	Avg Msg H	(partition) #Msg I
15	7	2250	2250	3	7	268	341	173
		2250				418		
		2250				336		
	4	1800	2100		4	368	368	176
		2250				468		
		2250				268		
	3	1350	1500		3	268	285	177
		1500				318		
		900				268		
	2	1350	1350		2	318	268	178
		900				218		
		1800				268		
20	9	4800	4000	3	8	1090	1054	295
						1090		
						980		
		4000		4	9	532	559	239
						664		
						482		
		3200		5	7	306	350	224
						388		
						356		
	6	4000	3467	3	6	890	733	297
						818		
						490		
		1600		4	6	482	565	242
						582		
						632		
		4800		5	6	274	301	219
						324		
						306		
	3	3200	3467	3	3	1090	823	300
						546		
						834		
		3200		4	3	332	399	245
						432		
						432		
		4000		5	3	370	338	222
						338		
						306		
	0	800	800	3	0	290	290	303
				4	0	232	232	248
				5	0	210	210	225
25	12	7500	6667	3	11	1000	1085	451
						982		
						1272		
		6250		4	10	790	664	369
						516		
						686		
		6250		5	12	500	567	316
						600		
						600		

	SMBTC [5]					scheme[7]		quickcon
# Proc A	# fPr B	# Msg C	Avg Msg D	# Gr E	# fPr F	# Msg G	Avg Msg H	(partition) #Msg I
25	8	6250	5000	3	8	982	1053	454
						1016		
						1162		
		3750		4	8	588	655	371
						686		
						692		
		5000		5	8	500	500	312
						500		
						500		
	4	7500	7917	3	11	1000	914	458
						1016		
						726		
		7500		4	10	614	585	375
						620		
						522		
		8750		5	12	450	417	316
						400		
						400		
	0	1250	1250	3	0	436	436	462
				4	0	346	346	379
				5	0	300	300	320
50	24	65000	58333	4	22	4376	4563	1312
						4512		
						4800		
		65000		5	22	2900	2700	1073
						2700		
						2500		
		45000		10	24	1500	1550	716
						1550		
						1600		
	16	7500	10417	4	16	4768	4552	1318
						4376		
						4512		
		10000		5	16	2650	2817	1079
						3100		
						2700		
		13750		10	16	1400	1300	724
						1300		
						1200		
	8	16250	13750	4	8	3696	4022	1324
						4192		
						3904		
		11250		5	8	2650	2650	1087
						3050		
						2250		
		13750		10	8	1100	967	732
						850		
						950		
	0	2500	2500	4	0	1288	1288	1334
				5	0	1050	1050	1095
				10	0	700	700	740

References

[1] Lamport L., Pease M., Shostak R., *'Reaching Agreement in the presence of Faults,'* Journal of ACM, Vol.27, pp. 228-234, Apr 1980.

[2] M. Fischer, N. Lynch, *'A lower Bound for the Time to Assure Interactive Consistency,'*Information Processing Letters, Vol. 14, No. 4, pp 183-186, June 1982.

[3] D. Dolev, H. R. Strong, *'Authenticated Algorithms for Byzantine Agreement,'* SIAM Journal on Computing, Vol. 12, No. 4, pp. 656-666, Nov. 1983.

[4] A. W. Krings, T. Feyer, *'The Byzantine Agreement Problem: Optimal Early Stopping,'* in Proceedings of the 32nd Hawaii International Conference on System Science, 1999.

[5] J. Widder, G. Gridling, B. Weiss, *'Synchronous Consensus with Mortal Byzantines,'* in Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable systems and Networks, pp. 102-112, 2007.

[6] T. Chiang, H. Tsai, Y. Huang, *'A Partition Network Model for Ad Hoc Networks'*, IEEE Wireless and Mobile computing conf, Networking and Comm, Vol.3, pp.467-472, 2005.

[7] M. Dalui, B Chakraborty, B. K. Sikdar, *'An Efficient Scheme For Quick Consensus In Partitioned Adhoc-Network'*, Submitted in 34th IEEE Conference on Local Computer Networks, Switzerland, 2009.

[8] Kalyan Mahata, Meghnath Saha, and Sukanta Das, *'Cellular Automata Based Coordinator Selection Scheme in Distributed System'*, accepted in CSC'09, USA, 2009.