Averaging Consensus over Erasure Channels via Local Synchronization

Saber Salehkaleybar
Dep. of Electrical Engineering
Sharif University of Technology
Tehran, Iran
Email: saber_saleh@ee.sharif.edu

S. Jamaloddin Golestani
Dep. of Electrical Engineering
Sharif University of Technology
Tehran, Iran
Email:golestani@ieee.org

***Abstract*—Averaging consensus on the values of nodes in a network is a principal problem in distributed computation. In the presence of erasure channels, conventional averaging consensus algorithms may not converge to the average value if packets are erased in arbitrary order. In this paper, we propose a "Pseudo-Synchronous Averaging Consensus" (PSAC) algorithm to guarantee averaging consensus over erasure channels by employing tagged packets. We show that the PSAC algorithm has a simple structure and it can work with just two tags "0" and "1". In asynchronous networks, the PSAC algorithm is a synchronizer in the sense that it keeps the updates of various nodes in step with each other. By exploiting the broadcast nature of wireless links in complete graphs, the PSAC algorithm obtains the exact average value with minimum number of transmissions, in the asynchronous setting.**

I. Introduction

Distributed averaging consensus is a fundamental problem in network distributed computations; moreover, it can be used as a key component in designing more complex signal processing and distributed control algorithms [1], [2]. In this problem, each node has an initial value and the goal is to reach consensus on the average of initial values by using a robust distributed algorithm. In the literature, there are several works in designing distributed algorithms for averaging consensus [3], [4]. However, most of these works assume that the links between nodes are reliable or erased symmetrically[1] which may not be the case in practical scenarios.

In [4], authors proposed an algorithm that achieves averaging consensus in an erasure network, provided that the network remains balanced after the occurrence of erasures. In [5] and [6], two synchronous algorithms have been proposed to solve averaging consensus problem under asymmetric erasures. The algorithm in [5], has two types of standard and corrective iterations. In each iteration, each node transmits a packet for a number of times. To ensure convergence to the average value, the number of retransmissions of a packet should be carefully adjusted, in accordance with the network topology. In [6], nodes exchange surplus variables in addition to updated values. During algorithm execution, the surplus variables monotonically increase with time, making it necessary to reset such variables through a sophisticated scheme [6]. Recently in [7], tree codes have been employed to guarantee averaging consensus in the case of asymmetric erasures. In this work, nodes use maximum likelihood decoder and solve linear equations recursively, in order to decode sent data with an exponentially decaying probability of error.

[1]Links are erased symmetrically if a packet is erased from node i to node j at time t, the same happens for the packet transmitted from node j in the reverse direction. In the asymmetric case, the packet from node i to node j and the one in the reverse direction are erased independently from each other.

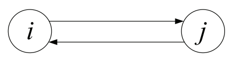

Fig. 1. A simple network with two nodes and two erasure channels

To show what difficulties may arise in the case of erasure channels, consider the simple network in Fig. 1 with two nodes i and j, and two erasure channels $i \to j$ and $j \to i$. Assume that the two nodes want to reach consensus on the average of their initial values by passing packets over the erasure channels. Note that packets may be erased in any order.

One simple algorithm is that each node sends its packet periodically to the other node until it is received. Since the two nodes should reach consensus, each node needs to know that its packet has been received successfully. Otherwise, one node might stop sending packets after it has received other node's value, while all of its transmissions to other node have been erased. To resolve this problem, each node can use feedback messages to inform the other node that its value has been received. Therefore, when node j receives the initial value of node i, it periodically sends feedback messages to node i to inform that its initial value has been received. Node i does the same thing after receiving the initial value of node j. However, the problem remains unsolved. Since it is not clear how each node knows its feedback message is received. One solution is to use another feedback message. Continuing in this way, we should use feedback messages to inform about the reception of other messages and we get stuck in a loop. In fact, it can be proved that two nodes cannot reach consensus on the average value over erasure channels in finite transmissions by any algorithm. The proof is similar to that of the coordinated attack problem [8] and it is given in the appendix.

Although nodes cannot reach consensus on the average value over erasure channels in finite transmissions, they can get close to it by updating their values, iteratively. Our approach here is to use "tagged packets" in an iterative algorithm. Let us explain how tagged packets can be used to converge to the average value in the simple network depicted in Fig. 1. Assume that each node sends its current value, v, with two tag numbers, tag_1 and tag_2 in the packet (v, tag_1, tag_2). The first tag shows the number of iterations that the node has updated its value. The second tag represents the maximum tag_1 which has been seen in the packets received from the other node. At the beginning of the algorithm, two nodes send their initial values with $tag_1 = 0$ and $tag_2 = -1$ to the other node repetitively. When a node receives a packet, it checks the tag_2 included in the packet to see whether it is equal to its own current sending tag_1. If so, it updates its value (here, it averages its value with the other node's value with the same tag_1). Then, it sends the new packet (v, tag_1+1, tag_1) to the other node repetitively. It can be seen that two nodes converges to the average value after first iteration and stay at this point in subsequent iterations.

In this paper, we develop the idea described above to general networks and show that the algorithm converges to the average value in the presence of asymmetric erasures. As we will see later, in the proposed algorithm, each node knows which packets to expect in each iteration, and it updates its value once it has received all these packets from its neighbours. This is the main idea of the "local synchronization" [9]. In the execution of the proposed algorithm, nodes update their values for the same iteration even if their updates occur asynchronously in time. Due to this property, we call the proposed algorithm as "Pseudo-Synchronous Averaging Consensus" (PSAC) algorithm. The main contributions of the paper are:

- The PSAC algorithm guarantees averaging consensus over erasure channels in the asymmetric case. We show that the PSAC algorithm has a simple structure and it can work with just two tags "0" and "1". In addition, by increasing the rate of clocks' ticking at nodes, we can speed up the algorithm while keeping the number of transmissions constant.
- Nodes update values of the same iteration even in the asynchronous setting. Therefore, the PSAC algorithm is a kind of synchronizer for averaging consensus problem in asynchronous networks (for the formal definition of the synchronizers, please refer to [8], ch. 16).
- The PSAC algorithm exploits the broadcast nature of wireless medium. We show that this property makes the PSAC algorithm converge to the exact average value in at most $2n$ transmissions in complete graphs over reliable channels (n is the number of nodes). This is the most efficient algorithm in terms of number of transmissions for the case that nodes transmit asynchronously.

The remainder of this paper is organized as follows: In Section II, we first describe the PSAC algorithm over reliable channels and then extend it to the case of erasure channels. We analyse the performance of the PSAC algorithm at the end of this section. In Section III, the simulation results are provided. Finally, we conclude with Section IV.

II. Pseudo-synchronous Averaging Consensus Algorithm

We consider a wireless network with n nodes. The topology of the network is represented by a graph, $G = (V, E)$, with the vertex set $V = \{1, ..., n\}$, and the edge set $E \subseteq V \times V$, such that $(i, j) \in E$ if and only if nodes i and j can communicate directly. We assume that if there exists a link from node i to node j, then there is a link in the reverse direction, but they are erased independently from each other. The set of neighbours of node i, $\mathcal{N}_i$, is $\mathcal{N}_i = \{j : (i, j) \in E\}$. Each node i has an initial value $x_i(0)$, and the goal is to compute the average of initial values, $x_{ave} = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$, by a distributed algorithm. Each node updates its value based on received packets from its neighbours.

Nodes send their packets asynchronously in the network. To model the asynchronous transmissions, we assume that each node has a clock which ticks independently from others at the times of a rate-λ Poisson process [3], [9]. When node i's clock ticks, the node decides to send a packet or wait for another tick according to the history of its transmitted and received packets.

In the following, we first describe the PSAC algorithm over reliable channels and then extend it to the case of erasure channels.

A. PSAC algorithm over reliable channels

In the PSAC algorithm, nodes locally broadcast tagged packets to their neighbours. A tagged packet contains a value and a tag number. Let $(v, r)_i$ denote the packet carrying the value v and the tag number r sent by node i. We define the state of node i, $state_i$, as the maximum tag number that it has sent or it is ready to send. When node i receives a packet from node j with the tag number r, it sets the indicator value $I_i(r, j)$ from zero to one. Each node also keeps a value and updates it in time, iteratively based on values received from its neighbours. We denote the value of node i at $state_i = r$ by $x_i(r)$.

Each time node i receives a new packet, it checks whether it has received from all its neighbours a packet with the tag number equal to its current state, (i.e., whether $I_i(state_i, j) = 1$ for all $j \in \mathcal{N}_i$). If so, it will update its value, increment $state_i$, and wait for its next clock's tick to transmit the packet with the new tag number $r = state_i$, to its neighbour nodes. The updating rule of node i's value at the state r is given by the following equation:

$$x_i(r+1) = w_{ii}x_i(r) + \sum_{j\in\mathcal{N}_i} w_{ij}x_j(r) \qquad (1)$$

where $0 \leq w_{ij} \leq 1$ for $j \in \mathcal{N}_i \cup \{i\}$, and it is equal to zero if $(i, j) \notin E$. For initializing the algorithm, each node i, at time $t = 0$, sets $state_i = 0$, and $I_i(-1, j) = 1$, for all $j \in \mathcal{N}_i$, and sets its initial value, $x_i(0)$. Therefore, when node i's clock ticks for the first time, it will send the packet $(x_i(0), 0)$ to its neighbours. Algorithm 1 shows the steps taken by node i and its neighbour nodes, each time its clock ticks. Here, the

Algorithm 1 PSAC algorithm over reliable channels

```
1:  if $trans_i = 1$ then
2:      node $i$ transmits packet $(x_i(state_i), state_i)$;
3:      for $j = \mathcal{N}_i$ do
4:          if $(I_j(state_j, k) = 1$ for all $k \in \mathcal{N}_j)$ & $trans_j = 0$ then
5:              node $j$ updates its value by Eq. (1);
6:              node $j$ increases $state_j$ by one;
7:              node $j$ sets $trans_j = 1$;
8:          end if
9:      end for
10:     node $i$ sets $trans_i = 0$;
11: end if
```

parameter $trans_i$, when equals to one, indicates that node i has a packet to send.

Let $\vec{x}(r)$ be the vector $[x_i(r)]$ and W be a matrix with entries $[w_{ij}]_{n\times n}$. The update expression in Eq. (1), can be stated in matrix form as:

$$\vec{x}(r+1) = W\vec{x}(r) \tag{2}$$

Here, we implicitly assumed that all nodes will eventually reach state r, for all $r > 0$. One may wonder whether this assumption is indeed valid, or to the contrary, the algorithm could reach a sticking point where each node in the network has to wait for new updates form its neighbours, before it can proceed and update its own state; thereby, nothing changing at all in the network, afterwards. To answer this question, let us first define the system state at time t:

$$state_{sys}(t) = \min_i state_i(t). \tag{3}$$

where $state_i(t)$ is the state of node i at time t. The next proposition establishes that the system state cannot stop, indefinitely, at any one value, and will continue to increase.

Proposition 1: Consider any interval $(t, t+T)$ of duration T. The probability that the system state remains unchanged during this interval approaches zero, as T goes to infinity.

Proof. Let $N_{min} \triangleq \{i | i \in V, state_i(t) = state_{sys}(t)\}$. According to Eq. (3), $state_j(t) \geq state_{sys}(t)$, for all $j \in V$. Therefore, each node $i \in N_{min}$, at time t, has already received a packet with the tag number $r = state_{sys}(t)$ from each neighbour node j. It follows that at time t, $I_i(r, j) = 1$, $\forall j \in \mathcal{N}_i$. We conclude that the states of nodes in the set N_{min} will be incremented when clocks of all of them tick for the first time after time t. Once all nodes in the set N_{min}, increment their states, the system state will be incremented, as well. Therefore, we have:

$$\begin{aligned}&\text{Prob.}[state_{sys}(t+T) = state_{sys}(t)] = \\ &= \text{Prob.}[\text{For some node } i \in N_{min}, \text{there is no clock tick during } (t, t+T)] \\ &= 1 - (1 - e^{-\lambda T})^{|N_{min}|}\end{aligned} \tag{4}$$

where $|N_{min}|$ is the number if nodes in the set N_{min}. We conclude that:

$$\lim_{T\to\infty} \text{Prob.}[state_{sys}(t+T) = state_{sys}(t)] \longrightarrow 0. \tag{5}$$

□

It follows from the above proposition that the tag numbers passed between nodes, increase unboundedly with time. To have a more practical algorithm, we show next how it can be executed just by "0" and "1" tags.

B. Implementing PSAC with "0" and "1" tags over Reliable Channels

In this part, we explain how to run PSAC algorithm with "0" and "1" tags. In other words, nodes just tag their packets with "0" and "1" and they can still run the PSAC algorithm and get the same results when they use integer numbers for tagging packets. The following proposition gives us a clue to achieve this point.

Proposition 2: The difference between the states of two neighbour nodes cannot be more than one at any time.

Proof. By contradiction. Assume that the difference between the states of two neighbour nodes i and j is greater than one and $state_j > state_i$. Since $state_j - state_i > 1$, then node i eventually has sent the packet with the tag number $state_j - 1$ to node j. Otherwise, node j cannot update its value and it cannot be in $state_j$. However, if the node i has sent this packet, then its state should be at least $state_j - 1$ and this yields a contradiction. □

According to Proposition 2, the difference between the states of any pair of neighbour nodes never exceeds one. Hence, nodes can just send the residual of tag numbers with respect to (or modulo) 2.

When the system state is equal to r, we can write:

$$\vec{x}(r) = W^r \vec{x}(0) \tag{6}$$

Let W be doubly stochastic (i.e., $W\vec{1} = \vec{1}$ and $\vec{1}^t W = \vec{1}^t$ where $\vec{1}$ is the n-dimensional column vector of ones) and $\rho(W - \vec{1}\vec{1}^t/n) < 1$, then it can be shown that $W^r \longrightarrow \vec{1}\vec{1}^t/n$ and consequently $(x(r) \longrightarrow x_{ave}\vec{1})$ [3]. If the doubly stochastic matrix W is aperiodic and irreducible [10], then its eigenvalues $\{\lambda_i\}$ are:

$$-1 \leq \lambda_n \leq \cdots \leq \lambda_1 = 1 \tag{7}$$

where the eigen-vector corresponding to the eigen-value of one is $\vec{1}$. Now, let the relative error at the system state r, $e(r)$, be defined as:

$$e(r) = \frac{||\vec{x}(r) - x_{ave}\vec{1}||_2}{||\vec{x}(0) - x_{ave}\vec{1}||_2} \tag{8}$$

It can be easily seen that $e(r) \leq \max\{\lambda_2, |\lambda_n|\}^r$ [11]. Hence, $\max\{\lambda_2, |\lambda_n|\}$ governs the convergence rate of the algorithm. In addition, the optimal matrix $W^\star$ with the minimum $\max\{\lambda_2, |\lambda_n|\}$ can be obtained by solving a semi-definite programming problem [11]. We can also consider some heuristic matrices such as the maximum degree matrix, W_{maxdeg} [11]:

$$w_{ij} = \begin{cases} 1/d_{max} & \text{if } (i,j) \in E \text{ and } i \neq j, \\ 1 - d_i/d_{max} & \text{if } i = j, \\ 0 & \text{if } (i,j) \notin E, \end{cases} \tag{9}$$

Algorithm 2 PSAC algorithm over erasure channels

```
1: node $i$ sends packet $(x_i(state_i), state_i, \{j|I_i(state_i, j) = 1\})$;
2: for $j = \mathcal{N}_i$ do
3:     if $(I_j(state_j, k) = 1$ & $I_k(state_j, j) = 1$ for all $k \in \mathcal{N}_j)$ then
4:         if $update_j(state_j) = 0$ then
5:             node $j$ updates its value by Eq. (1);
6:             node $j$ increases $state_j$ by one;
7:             node $j$ sets $update_j(state_j) = 1$;
8:         end if
9:     end if
10: end for
```

where d_i is the number of neighbour nodes of node i and $d_{max} = \max_{i \in V} d_i$. Note that each node i can simply obtain the i-th row of matrix W_{maxdeg}, based on the number of its neighbours. Hence, it can update its value according to Eq. (1). Besides, d_{max} in Eq. (9) may be replaced by any upper bound on it; Therefore, nodes can simply run the PSAC algorithm using some upper bound on d_{max} and they need not know the exact value of it.

In the next part, we extend the PSAC algorithm to the case where links are not reliable.

C. PSAC over erasure channels

In this subsection, we extend the PSAC algorithm for the case that links are not reliable. We assume that when node i transmits a packet to its neighbours, any subset of nodes in $\mathcal{N}_i$ may receive the packet. However, it is assumed that after each transmission by node i, the packet is successfully decoded at each node $j \in \mathcal{N}_i$ with a positive probability.

In the case of erasure channels, first note that if we run the PSAC algorithm without any modification, one node might receive packets from all its neighbours with tag "r" and then proceed to send the packet with tag "$r + 1$". In this case, some of the neighbour nodes may not receive the packet with tag "r" due to erasure and remain at state r. To overcome this problem, node i must ensure that its packet with tag "r" is received by all its neighbours and it also receives packets with tag "r" from them. To do so, nodes include a list in their tagged packets. The list sent by node i shows which one of the packets with tag "r" from the nodes in $\mathcal{N}_i$ are successfully decoded at node i. In other words, the list includes those nodes $j \in \mathcal{N}_i$, for which $I_i(r, j) = 1$.

To put it another way, node i sends the packet with tag "r" at each clock tick until it infers its packet has been successfully decoded at its neighbour nodes by looking up the parameters $I_j(r, i)$ in the packets sent by node j, for all $j \in \mathcal{N}_i$. In fact, by sending the list, node j notifies each neighbour whether or not a packet with tag "r" has been received from it. Algorithm 2 shows the steps taken by node i and its neighbour nodes, each time its clock ticks. Next proposition claims that Proposition 1 holds true in the presence of erasure channels.

Proposition 3: In the PSAC algorithm over erasure channels, Proposition 1 still holds true.

Proof. As we discussed above, each node ensures that its packet with tag "r" is received by all nodes in $\mathcal{N}_i$ using the list mechanism. Therefore, when node i updates its value, all neighbour nodes already have the value $x_i(r)$. Since each node's clock ticks at the times of a rate-λ Poisson process and there is a positive probability in successful transmission between any two neighbour nodes, the packets with tag "r" will be received w.p. 1 as T goes to infinity. Now, by the same arguments in the proof of Proposition 1, we can show that all nodes in the set N_{min}, will eventually increment their states. Consequently, the system state will be incremented and the proof is complete. $\square$

In the same way in Proposition 2, we can prove that the PSAC over erasure channels, can run with "0" and "1" tags. The proof is left to the reader.

D. Performance Analysis of PSAC algorithm

One of the key performance metrics in designing averaging consensus algorithms is the minimum number of transmissions required to have the relative error less than a given threshold ϵ, i.e. $e(r) \leq \epsilon$. We denote the minimum number of transmissions to have $e(r) \leq \epsilon$ by $C_{rel}(\epsilon)$ and $C_{er}(\epsilon)$ for the cases of reliable and erasure channels, respectively. In PSAC algorithm over reliable channels, in each iteration of the algorithm, nodes transmit exactly n packets. Hence, we have $C_{rel}(\epsilon) = \frac{n \log(\epsilon^{-1})}{\log(\max\{\lambda_2, |\lambda_n|\}^{-1})}$ according to Eq. (8) and the subsequent paragraph. Now, let nodes' clocks tick with the rate λ'_1 and the time intervals needed to complete r iterations be $T_{rel.}(r, \lambda'_1)$ and $T_{er.}(r, \lambda'_1)$ in the cases of reliable and erasure channels, respectively. Therefore, we can speed up the PSAC algorithm in time for both cases with ratio λ'_2/λ'_1 if nodes' clocks tick with the rate λ'_2.

Studying $C_{er}(\epsilon)$ is more complicated because of erased packets in the network. Here, we give a naive analysis. Assume that packets are erased on each link with probability p. So, each node transmits $\frac{1}{1-p}$ packets on average until its value is received by all neighbour nodes. Therefore, $\frac{\alpha}{1-p} \times n$ packets are transmitted approximately in each iteration where α is a constant factor to count extra packets sent due to running the algorithm in the asynchronous setting. Hence, we have: $C_{er}(\epsilon) \approx \frac{\alpha \times n \times r_{min}}{1-p}$ where r_{min} is the minimum number of iterations to have the relative error less than the threshold ϵ.

At last, we analyse the performance of the PSAC algorithm in complete graphs for the asynchronous setting. Since each node receives packets from other nodes directly, it will have all initial values after the first iteration. If nodes use W_{maxdeg} in Eq. (9) with the upper bound $d_{max} + 1$ for the parameter d_{max}, they will converge to the exact average value after the first iteration (note that each node knows d_{max} in complete graph). Therefore, we conclude that the number of transmissions for obtaining the exact average value for the PSAC algorithm is at most $2n$ in complete graphs when nodes transmit asynchronously over reliable channels. This number of packet transmissions is considerably less than the ones for other related works [2], [3] which converge to the average value in the limit.

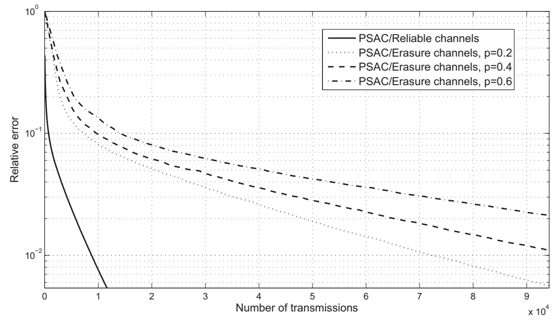

Fig. 2. Comparison of the performance of PSAC algorithm for different packet erasure probabilities.

III. Numerical Simulation

In this section, we study the effect of erasure channels on the performance of PSAC algorithm through simulations. We run the PSAC algorithm in 100 instances of network with random geometric graph topology and average the results. The number of nodes is 100 and each node's clock ticks according to a Poisson process with rate $\lambda = 1$.

Fig. 2 shows the relative error versus number of transmissions for the PSAC algorithm over reliable and erasure channels. The probability of packets being erased on each link is set to $p = [0.2, 0.4, 0.6]$. As it can be seen, in the presence of erasure links, the convergence rate of the algorithm decreases. Furthermore, the PSAC algorithm has better performance for lower value of the parameter p which is an obvious trend according to our discussion in previous section. In addition, we compute the slope of lines fitted to the results in Fig. 2. For $p = [0.2, 0.4, 0.6]$, the lines' slopes are -1.18e-5, -8.9e-6, and -6.2e-6, respectively. From previous section, we know that the ratio of two lines' slopes is approximately $\frac{1-p_1}{1-p_2}$ where p_1 and p_2 are the probabilities of packets being erased in the two fitted lines, respectively. Simulation results verify our analysis: For instance, for $p_1 = 0.2$ and $p_2 = 0.4$, the ratio of two lines' slopes is (-1.18e-5)/(-8.9e-6)=1.326 and $\frac{1-p_1}{1-p_2} = 1.33$.

IV. Conclusion

In this paper, we proposed the PSAC algorithm to guarantee averaging consensus over erasure channels. The PSAC algorithm has a simple structure and it works with just two tag numbers. Hence, it is suitable to be implemented in wireless sensor networks where sensor nodes have limited computational capability. Furthermore, we can speed up the algorithm through an increase in the rate of clocks at nodes while keeping the number of transmissions constant. In addition, we studied the effect of erasure channels on the convergence rate of the algorithm. We showed that the convergence rate is linearly dependent on $(1-p)$ where p is the links' erasure probability.

V. Acknowledgement

This work was partially supported by Iranian National Elite Foundation.

References

[1] J. Tsitsiklis, "Problems in Decentralized Decision Making and Computation," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, 1984.

[2] A. Dimakis, A. Sarwate, and M.Wainwright, "Geographic gossip: Efficient aggregation for sensor networks," Int. Conf. Inf. Proc. Sensor Networks (IPSN), Nashville, Apr. 2006.

[3] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah,"Randomized gossip algorithms, " *IEEE Transactions on Information Theory*, vol. 52, no. 6, June 2006.

[4] R. Olfati-Saber, J.A. Fax, and R.M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, jan. 2007.

[5] Y. Chen, R. Tron, A. Terzis, R. Vidal, "Corrective consensus: Converging to the exact average," *49th IEEE Conference on Decision and Control (CDC)*, pp.1221-1228, Dec. 2010.

[6] N.H. Vaidya, C. N. Hadjicostis, C.N. and A.D. Dominguez-Garcia, "Robust average consensus over packet dropping links: Analysis via coefficients of ergodicity," *IEEE 51st Annual Conference on Decision and Control (CDC)*, pp.2761-2766, Dec. 2012.

[7] R. T. Sukhavasi and B. Hassibi, "Tree Codes Improve Convergence Rate of Consensus Over Erasure Channels," available at arXiv:1204.0301v1 [math.OC], Apr. 2012.

[8] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann, 1996.

[9] D. Bertsekas, and J. Tsitsiklis, *Parallel and distributed computation*, Prentice Hall Inc., 1989.

[10] D. Shah, *Gossip algorithms*, vol. 3, no. 1, Now Pub., 2009.

[11] S. Boyd, P. Diaconis, and L. Xiao, "Fastest Mixing Markov Chain on a Graph," *SIAM Review*, vol. 46, no. 4, pp. 667-689, 2004.

VI. Appendix

Proposition 4: Nodes in the network given in Fig. 1, cannot reach averaging consensus by any algorithm in finite number of transmissions.

Proof. Without loss of generality, assume that nodes send their packets in consecutive rounds. If a node has no new data, it will send dummy packet. We define the execution of the network as the sequence: $S_0, M_0, N_0, S_1, M_1, N_1, \cdots$, where S_r is the $[state_i, state_j]$ after r rounds and M_r and N_r represent the packets that are sent and received by nodes at round r, respectively. If α and α' are two executions of the network, we say that α is indistinguishable from α' with respect to node i, if node i has the same sequences of states, sending packets, and receiving packets in α and α' and we denote it by $\alpha \overset{i}{\sim} \alpha'$. Now, suppose there exits an algorithm that solves the problem. We show that this yields a contradiction. Using this algorithm, nodes reach averaging consensus after a finite number of transmissions, say after r rounds. Let α_1 be the execution of the algorithm that nodes reach consensus. Now, let α_2 be the same as α_1 except that the last message from node i is erased. It can be seen that $\alpha_1 \overset{i}{\sim} \alpha_2$, and therefore nodes also decide on the average value in α_2. By alternately removing the last packet from nodes i and j, we reach to an execution α, which two nodes reach consensus on the average value without sending any packet and this yields a contradiction. □