Communication-efficient Distributed Average Consensus Algorithm for Packet-switched Networks

Muhammad Talha Paracha[1] and Waheed Bajwa[2]

***Abstract*— Computation and communication efficient average consensus algorithms are highly desirable for Internet of Things applications. Previous work on average consensus algorithms focuses more on their theoretical aspects (e.g convergence guarantees) and less on their practical aspects (e.g communication cost, power consumption). In this paper, we analyze an algorithm for distributed average consensus which is robust to network adversities found in packet-switched networks, and find a communication overhead in it. We propose a TCP-based modification to the original UDP-based algorithm and hypothesize it to be more efficient in terms of bytes sent and received, contrary to the conventional wisdom associated with TCP-based algorithms. We implement these two algorithms on a test bed of Raspberry Pi computers connected wirelessly in an ad hoc manner, and gather empirical results which support our hypothesis. We believe the resulting algorithm is very suitable for real-world usage and that our insights will help the community build even better algorithms for distributed average consensus in terms of communication cost.**

I. INTRODUCTION

Consider a set of nodes connected to each other in an ad hoc manner. Each node stores a value which, for example, could be a sensor measurement. Imagine we want every node to estimate the average of all values stored across the network so, for example, to increase the accuracy in measurement. This task, formally termed as "average consensus" in the literature, is being studied for more than two decades. A vanilla approach of connecting all nodes with a centralized location might not be possible because of the lack of infrastructure inherent in ad hoc networks. Other naive ways, such as nodes flooding the whole network with their initial values, will not scale well in terms of computation and communication cost.

Distributed average consensus algorithms, on the other hand, provide a sophisticated approach in which each node calculates the estimated average value by communicating with its first-hop neighbors every once in a while. The algorithms asymptotically converge to the exact average and their convergence guarantees have been studied with various network adversities such as packet loss, communication link loss, node failure/recovery, channel noise, quantized data etc.

The importance of average consensus algorithms can be realized by looking at the various applications in which they are used such as distributed clock synchronization [1], co-ordination of autonomous robots for flight [25], load balancing [4], vehicle formation control [3] and sensor fusion [2]. Furthermore, these algorithms form building blocks of many advanced algorithms for distributed optimization such as Cloud-K SVD [5], distributed Maximum Likelihood Estimation [7] and distributed Kalman Filtering [6]. The distributed consensus algorithms in ad hoc networks are even used to mathematically model how species like fireflies are able to achieve global coordination via local interactions [8].

The motivation behind our work was to analyze the practical aspects, like communication cost and power consumption, of theoretically robust distributed average consensus algorithms by implementing them on a test bed of real-world Internet of Things (IoT) devices. This paper has two contributions (1) We propose a distributed average consensus algorithm which, despite being robust to all network adversities found in packet-switched networks, is also communication-efficient in terms of bytes sent and received. (2) We generate useful insights on why algorithms using lightweight connectionless UDP protocol are not always communication-efficient as compared to their variants with heavy connection-oriented TCP protocol.

This paper is structured as follows; In Section II, we provide a review of various distributed average consensus algorithms present in the literature, and conclude with the one most suitable for packet-switched networks. In Section III, we formally discuss the task of distributed average consensus and analyze the aforementioned algorithm in detail. In Section IV, we propose a TCP-based modification to that algorithm and hypothesize it to be more communication efficient as compared to its original UDP-based implementation. In Section V, we evaluate the two algorithms on a test bed of Raspberry Pi computers connected wirelessly in an ad hoc manner and emulated with packet loss / delay. Finally in Section VI, we conclude with a summary of our work and the plans for future.

II. BACKGROUND

A. Types of algorithms

The various distributed average consensus algorithms present in literature can be classified in many ways. Here we discuss the classifications most suitable for digital packet-switched wireless ad hoc networks:

1) Synchronous vs Asynchronous Iterations: Algorithms with synchronous iterations assume that all nodes in the network perform the consensus iterations in a synchronized fashion. Due to this harmony, the node with slowest communication in each iteration creates a bottleneck for the whole network. As nodes in real-world wireless networks

[1]M. T. Paracha is with the School of Electrical Engineering & Computer Science, National University of Sciences & Technology, ISB, Pakistan 14besemparacha at seecs.edu.pk

[2]W. Bajwa is with the Faculty of Electrical and Computer Engineering, Rutgers University, NJ, USA waheed.bajwa at rutgers.edu

can face arbitrary delays in communication, such algorithms are undesirable.

2) Reliable vs Unreliable Communication: Algorithms with reliable communication assume that a message sent by a node to any of its neighbors is guaranteed to eventually reach its destination. This assumption typically translates to using a reliable communication protocol like TCP during implementation. Since TCP is a connection-based protocol with a bigger packet header as compared to a connectionless protocol like UDP, such algorithms are conventionally considered to be more computation and communication heavy, and are thus undesirable (we later show in our work that this is not always true).

3) Fixed vs Switching Topologies: Algorithms which work on fixed topologies assume that a communication link between any two neighbors in the network will never fail, and hence the network topology would remain static during all consensus iterations. As nodes frequently move in mobile ad hoc networks commonly found in real-world scenarios, such algorithms are undesirable.

B. Literature review

Xiao et al. studied various linear distributed average consensus algorithms in a synchronized environment with reliable communication and fixed network topology [9]. They provided different heuristics based on the Laplacian of the associated undirected graph for fast convergence of algorithms. We find that many future works use their proposed algorithms as a base and modify it to handle network adversities such as packet loss, packet delay and communication link loss. In fact, Xiao et al. extended their own work to handle switching network topologies using Metropolis weights [10]. The resulting algorithm works as long as the infinitely occurring communication graphs are jointly connected, which is a reasonable assumption as the estimated average value is required to be the exact value of all nodes in the network.

Chin et al. extended Xiao et al.'s work to deal with severe packet losses due to unreliable communication [11]. Apart from the standard iterations performed in the base algorithm, their algorithm also performs "corrective" iterations on regular intervals, so to eliminate the errors in estimated average values accumulated on any node due to packet losses. They prove the error-free convergence of their algorithm, and use simulations to show the behavior of their algorithm when corrective iterations are performed less-vs-more frequently. They initially assumed constant probability of packet loss in both directions on any communication link, but later removed this assumption and still proved the algorithm convergence in [12]. They also extended their work to converge faster using the accelerated consensus technique [13]. Nonetheless, their algorithm is limited by a synchronized method of communication i.e. after every corrective iteration, all nodes behave as if no packet was lost. Forcing all nodes to do a corrective iteration at the same time is not possible in an asynchronous mode of communication.

Kar et al. studied the distributed average consensus problem with challenges arising in analog communication such as channel noise [14] and quantized data [15]. Both their algorithms work on switching topologies, but are unable to deal with packet loss and/or delay.

Mehyar et al. devised an asynchronous algorithm using simple messaging-passing schemes which is able to handle dynamic topologies but assumes a reliable mode of communication [16]. In fact that is why the authors emulate their algorithm on a real-world TCP/IP network and not a UDP/IP network.

Wu et al. studied average consensus with both delays and packet losses in communication [17]. But they treat packet losses as absence of communication links i.e. packet losses either simultaneously exist on both directions or they do not. Thus they avoid discussing cases of unreliable communication in which packet losses are asymmetric and can arbitrarily happen on any side of communication, as is the case when implementing an algorithm in UDP. Zhang et al. dealt with both communication delay and asymmetric packet loss [18]. But they assume that the delay is not larger than a predefined sampling period. As a result, their resulting algorithm is pseudo-asynchronous because it still suffers from the main weakness in synchronized communication i.e. the node with slowest communication becoming a bottleneck for the whole network.

Fagnani et al. modeled the problem as a digraph in order to deal with switching topologies and packet losses simultaneously by just compensating for random link failures [20]. However, the strategies proposed do not always lead to the exact average. Hadjicostis et al. also modeled the problem as a digraph and provided a robust algorithm for converging to the true average despite packet losses and switching topologies; the limitation being that their algorithm is synchronous [19]. Although one variant of their algorithm handles the problem of unknown out-degrees at each node, a problem rarely discussed in other works.

Khosravi et al. studied the issues in wireless sensor networks [21] but they only talk about packet delay and communication link failure, and avoid any discussion on packet loss. Recently they proposed an asynchronous gossip-based algorithm for distributed averaging [22], but it only works on fixed strongly connected topologies.

Eyal et al. [23] built a robust live monitoring system for dynamic sensor networks. Their underlying average consensus algorithm is asynchronous, handles node failures/recoveries along with packet losses in communication, and also works with switching network topologies. Yet they assume that the messages which are not lost on any communication link, arrive in a FIFO order. It is not clear how critical this assumption is to the convergence proof of their algorithm. Since the UDP specification does not say anything about the message order, this algorithm can only be implemented with a protocol like TCP which guarantees in-order delivery [24].

Kriegleder et al. proposed an asynchronous implementation to control a distributed embedded system for flight

which converges to error-free average, does not assume reliable communication and handles dynamic topologies [25]. Furthermore, it supports dynamic consensus i.e. ability of the network to converge to new average if any of the nodes update its stored value. Due to all these characteristics, we believe this is the most suitable implementation for practical usage in packet-switched wireless networks of all the existing algorithms in literature.

III. ANALYSIS OF KRIEGLEDER ET AL'S IMPLEMENTATION

We start by formally defining the problem statement. Then we describe the standard distributed average consensus algorithm which provides the basis to Kriegleder et al's implementation. Finally, we analyze their implementation and point out a communication overhead in it.

A. Problem Formulation

We model a multi-agent network as an undirected connected graph G = {N , E} consisting of a set of nodes N along with a set of edges E. Each edge {i, j} ∈ E represents a bidirectional communication link. The set of neighbors belonging to node i is denoted by N_i = {j | {i,j} ∈ E}. Each node i has a state $x_i(t)$ at time t ≥ 0 and a stored value z_i. The states are initialized such that $x_i(0) = z_i$ where $z_i \in$ R for the node i. Now we want that ∀ i ∈ N:

$$\lim_{t\to\infty} x_i(t) = \frac{1}{N}\sum_{n=1}^{N} z_i$$

B. Standard Average Consensus

Standard distributed average consensus algorithm [9] assumes reliable communication between neighbors and runs in a synchronous mode. Its iterations can be summarized as a combination of two steps: 1) an agent i communicates its state information to all its neighbors j ∈ N_i 2) all agents update their state through a linear combination of their own state and the state information of neighbors received in the previous step. The procedure can be mathematically written as:

$$x_i(t+1) = x_i(t) + \alpha \sum_{j\in N_i} (x_j(t) - x_i(t)) \qquad (1)$$

So in other words, each agent update its state using the disagreement of states with all its neighbors, scaled by a factor of α. Thus the convergence rate of this algorithm depends on the scaling factor used. Convergence is guaranteed as long as the following constraint is met:

$$0 < \alpha < \frac{1}{d_{max}}$$

where d_{max} denotes the maximum degree among all nodes in the network. The constraint can be fulfilled by realizing an upper bound on the maximum possible neighbors for any node.

C. Kriegleder et al's Contribution

Kriegleder et al noticed that the procedure in (1) can be written as:

$$x_i(t+1) = z_i + \alpha \sum_{j\in N_i} \delta_{ij}(t+1)$$

$$\delta_{ij}(t+1) = \delta_{ij}(t) + x_j(t) - x_i(t)$$

where $\delta_{ij}(t)$ represents the cumulative disagreement of node i with its neighbor j until time t.

In this way, the algorithm becomes explicitly dependent on z_i which makes it capable of dynamic consensus i.e. ability of the network to converge to new average if any of the nodes update its stored value. Furthermore, the algorithm can now also handle switching topologies by setting the appropriate δ_{ij} to zero whenever a node leaves / joins the network. Although, this does assume that all nodes are capable of detecting any change in topology due to their immediate neighbors.

They also observed that whenever any change $\triangle_{ij}$ = x_j - x_i is made to δ_{ij} at node i, it is compensated by the opposite change $\triangle_{ji}$ = - $\triangle_{ij}$ to δ_{ji} at node j. This is a necessary condition to keep the sum of states across the whole network constant, which, in-turn, is essential if the consensus iterations are required to converge to the exact average. By exploiting this observation along with a directed communication scheme, the standard algorithm was made asynchronous.

Accordingly, all nodes are given unique numerical identifiers (ID) in the Kriegleder et al's algorithm. Let a node with lower ID start an interaction with any of its neighbors with higher ID by sending its state value. The neighbor then uses this information to find the disagreement in states, updates it own state value on the basis of this disagreement, and finally responds back by sending the disagreement calculated earlier. This makes the node which started the interaction, update its state value too on the basis of the information received. In this way, one loop of interaction is closed in which both the nodes have changed their values asynchronously, while keeping the sum constant.

The message sending procedure at each node is run at periodic intervals so to keep the interactions going on and to also compensate for packet losses (if any). On each interval, nodes communicate with all of their neighbors sending either their state or the disagreement information depending on the relative IDs. All data sent get marked with sequence numbers to ensure that a node does not use a given neighbor's information multiple times within the same loop of interaction. The sequence numbers are initialized and updated carefully to make sure the algorithm works as expected. This ultimately makes the algorithm implementable in UDP. We encourage the readers to look at their paper [25] for complete implementation details.

D. Communication Overhead

We observe that there are many cases in which the algorithm would end-up wasting network bandwidth. Consider a

scenario in which there are just two nodes A and B connected to each other. Now as the average consensus algorithm is running:

- If the timeout set for A's message sending procedure is such that the packet it sends to B arrive with delay greater than the timeout, A will end up sending same information to B. And finding an ideal value for timeout is non-trivial because of the arbitrary delays present in real-world networks.
- If the packets sent by A are getting lost continuously, but A is still able to receive B's packets, all those received packets from B will contain same information as B has not heard from A in a while and has consequently not updated its state either.

Because of sequence numbers appended in data sent / received, a node can not use same information from its neighbor multiples times. Hence in both of the above cases, the nodes have no option but to simply ignore all sets of packets with same information.

Algorithm 1 Message Sending Procedure for agent i.

1: j: Neighbor ID
2: x_i: State of i
3: $\triangle_{ij}$: Current disagreement between i and j

4: **procedure** MSGSENDER(j,x_i, $\triangle_{ij}$)
5: **if** $j > i$ **then**
6: $DATA \leftarrow x_i$
7: **else**
8: $DATA \leftarrow \triangle_{ij}$
9: SEND *DATA*

IV. PROPOSED MODIFICATION

On a closer analysis, it can be seen that the techniques used by the original UDP-based algorithm for compensating packet losses and delays are a subset of the ones used by TCP i.e. retransmissions and sequence numbers on packets sent. Thus, we propose to simply use a TCP-based implementation. We hypothesize that the resulting algorithm will perform at least as good as the original algorithm in ideal network conditions, but will be communication-efficient in terms of bytes sent / received in the presence of packet losses and delays typical in any real-world network. The hypothesis seems intuitive, as TCP was actually built for the task of efficient communication in the presence of network adversities.

The resulting implementation can be seen as a combination of two functions described in Algorithm 1 and 2. These look very similar to the ones in original implementation, with the only differences being removal of timeout and sequence numbers handling part.

As we do not run the message sending procedure at regular intervals but instead place a call to it inside the message handling procedure, we need a way to kick-start the algorithm. We do so by making every node run Algorithm 1 w.r.t all of their neighbors with greater IDs. This starts the algorithm and makes all nodes in the network communicate with each other in an asynchronous fashion, while asymptotically converging to the exact average.

As far as the aforementioned two cases are concerned, the TCP-based implementation would avoid them by 1) Varying the timeout to efficiently tackle network delay 2) Ensuring that for all pairs of neighbors, only one of them is in the process of communicating state / disagreement at any instant.

Algorithm 2 Message Handling Procedure for agent i; executed upon reception of a message.

1: z_i: Initial value of i
2: x_i: State of i
3: δ_{ij}: Current disagreement between i and j
4: α: Scaling factor

5: j: Neighbor ID
6: *DATA*: Data received from j
7: **procedure** MSGHANDLER(j,*DATA*)
8: **if** $j < i$ **then**
9: $\triangle_{ij} \leftarrow$ *DATA* - x_i
10: **else**
11: $\triangle_{ij} \leftarrow$ -*DATA*
12: $\delta_{ij} \leftarrow \delta_{ij} + \triangle_{ij}$
13: $x_i \leftarrow z_i + \alpha \sum_{j \in N_i} \delta_{ij}$
14: MsgSender(j,x_i, $\triangle_{ij}$)

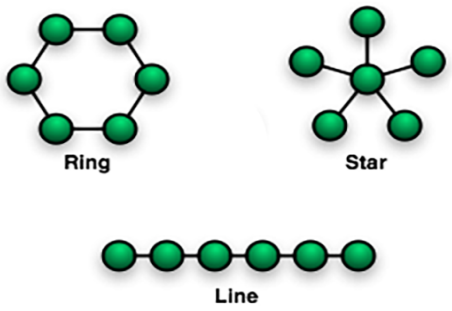

Fig. 1. Different network topologies used in our experiments.

V. EXPERIMENTAL EVALUATION

A. Implementation Details

We built a test bed of five Raspberry Pi 2 Model B computers running Raspbian OS and connected via 802.11 wireless ad hoc network. All of them were equipped with a USB wireless antenna which used the rtl8192cu wireless driver. The nodes were within wireless range to communicate with each other. Different topologies, as illustrated in the figure 1, were emulated by allowing communication only between the perspective neighbors in any topology.

Consensus was done on a vector of length hundred. The information provided to each node included its neighbors,

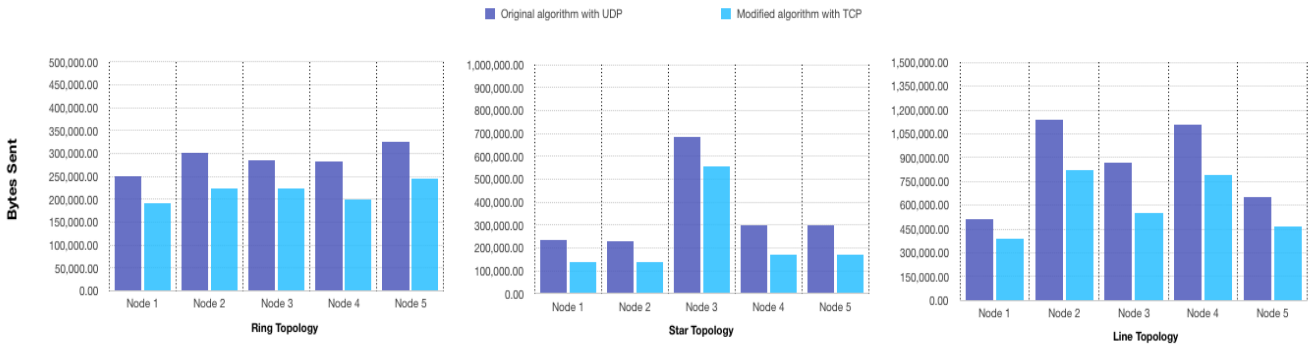

Fig. 2. Amount of bytes sent during the consensus iterations of the two algorithms, with different topologies. The measurement at each node was done until the euclidean error between estimated average and ground truth fell below 1e-5.

initial vector and the ground truth vector i.e. exact average of all initial vectors across the network. Values in the initial vectors were randomly sampled between one and hundred.

We used Python programming language to implement the original and modified algorithms. The selection of Python was based on its prototypical nature and popularity in IoT applications. The differences between two implementations were the usage of UDP in original-vs-TCP in modified and some minor changes mentioned in the previous section. A constant timeout value, required by the original algorithm only, was set to 0.2 seconds which was much higher than the max round-trip time between any two nodes. To make the two algorithms comparable, we fixed the convergence rate by using a constant scaling factor. And in all experiments, each node ran consensus iterations and measured network statistics until the euclidean distance between its estimated average vector and the ground truth vector fell below 1e-5.

We could not find a way to reliably measure network statistics for the exact OS processes tied to our algorithms. Instead, we used the cross-platform python library psutil to measure the number of bytes sent / received by the wireless antenna whenever the consensus algorithms were running. We ensured that the antenna was not being utilized for any other task when our experiments were running. To further reduce the effects of any possible interference, each experiment was run multiple times and only the average across all the runs is reported in the results section.

Linux tool netem was used to emulate network adversities. Accordingly, a 15% packet loss and a delay of 15 ± 10 milliseconds sampled from a normal distribution were added to all nodes. Network adversities found in practice are not purely random. So to approximate, a correlation value of 25% was also used. This helped in emulating burst packet losses and delays.

B. Results

Figure 2 shows the amount of bytes sent during the consensus iterations of the two algorithms, with different topologies. We observed similar trends for the amount of bytes received in our experiments, and do not show them here for conciseness. Our experiments showed that both implementations converge, and the convergence rate is in accordance with the graph topology. For example, it is well known that information diffusion in star topology is easier than in line topology. By observing the total amount of bytes sent across all nodes for either implementation in star-vs-line topology results, a similar conclusion is reached. In Ring topology, each node has two neighbors, so the amounts of bytes sent/received by nodes are expected to be roughly equivalent. In Star topology, central node is connected to all other nodes, so the amount of bytes sent/received by the central node is expected to be far greater than any other node. In Line topology, middle three nodes are connected to two neighbors each but the information diffusion is highest for the central node. Our results are in accordance with these basic expectations, which appear because of different levels of graph connectedness. This increases the confidence that the two implementations we wrote are bug-free.

In all three topologies, the amount of bytes sent by any node are lesser in our modified TCP-based implementation than the original UDP-based implementation. To be precise, the average percent change, across all nodes, in the amount of bytes sent when changing from original to modified implementation, was found to be -24% in case of Ring topology, -36% in case of Star topology and -28% in case of Line topology. This confirms our hypothesis that in a realistic setting with both packet loss and delay, the proposed modification makes the original algorithm more communication efficient. The exact amount of percent change depends on a lot of factors such as the timeout set in the original algorithm, amount of packet loss and delay, and the graph connectedness. In our experiments, we set the timeout to a much relaxed value of 0.2 seconds to illustrate that selecting the ideal value is a non-trivial task.

In figure 3, we present results to show the exact effect of different types of network adversities on the amounts of bytes sent for the two algorithms in case of Ring topology. Accordingly, without packet loss and delay both algorithms are equivalent in terms of communication cost. The exact amount of average percent change was found to be +1%. The percent change is not as much as one expects by looking at the relative packet header size of UDP-vs-TCP because

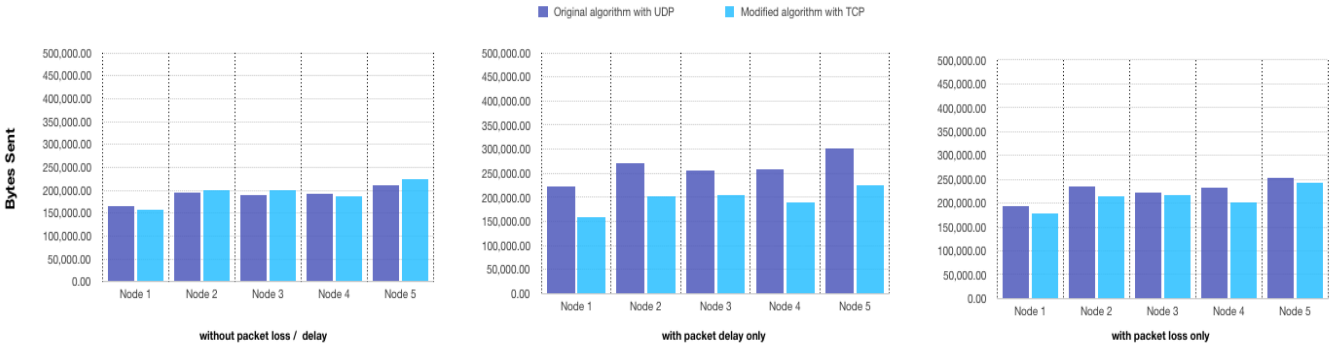

Fig. 3. Amount of bytes sent during the consensus iterations of the two algorithms, with different network adversities.

(1) We are doing vector consensus, and thus the size of datagram in each packet is way larger than the header size in either case (2) TCP implementation adds sequence number in packet header, while the UDP implementation adds sequence number in datagram so that effect is neutralized.

In presence of packet delay only, the modified algorithm presents most savings in network utilization i.e. average percentage change of -25% in the amount of bytes sent, as compared to -24% in case of both delay and loss. This is expected as the TCP protocol is well suited to handle packet delays. It does so by varying its timeout in sophisticated ways. Some implementations initially set the timeout to 2*RTT and then double it every time an expected ack is not received. This is efficient in contrast to a constant timeout set in the original UDP-based algorithm as it sends the packets at fixed regular intervals regardless of network conditions.

In presence of packet loss only, the modified algorithm does not save as much bandwidth as in other cases i.e the percentage change in this case was found to be only -7%. This is also expected as both implementations handle packet losses through retransmissions.

Our results show that the bandwidth savings in individual cases, i.e. packet loss only or packet delay only, do not add up when both adversities simultaneously affect the network. So for ring topology, the percentage change in bytes sent was found to be -24% in presence of both packet loss and delay. With packet losses only, the amount was -25% and with packet delays only, the amount was -7%. This is because of an inherent limitation in TCP to distinguish between a packet loss and delay, which causes it to work not as well as in an environment with packet delay only. Nonetheless, even in the combined case, the bandwidth savings are sizable enough for us to confidently say that our modified TCP-based implementation is indeed communication-efficient as compared to the original UDP-based implementation.

VI. CONCLUSIONS

In this work, we proposed a TCP-based distributed average consensus algorithm suitable for packet-switched networks and empirically demonstrated it to be more communication-efficient as compared to its UDP-based variant. We believe our insights show the need for a robust distributed average consensus algorithm, implementable in UDP, which does not directly or indirectly end up relying on packet retransmissions and sequence numbers to handle packet losses and delays. The only existing works in literature we see in this direction are [11] and [23], but the former only runs in a synchronous mode and the latter cannot be implemented in UDP because of the FIFO order assumption.

In future, we would like to see the effect of our proposed modification on the CPU time of the original algorithm. This could help us make a conclusive statement about the power consumption of the two implementations. Also, in this work we ignored the convergence speed as both the implementations used same convergence factor and were thus inherently similar in the way they did computation; they only differed in the ways of communication. So, another interesting extension to this work would be the comparison of all existing TCP-based robust distributed average consensus algorithms in terms computation and communication cost, as well as convergence speed.

REFERENCES

[1] Schenato, Luca, and Federico Fiorentin. "Average TimeSync: a consensus-based protocol for time synchronization in wireless sensor networks1." IFAC Proceedings Volumes 42.20 (2009): 30-35.

[2] Olfati-Saber, Reza, and Jeff S. Shamma. "Consensus filters for sensor networks and distributed sensor fusion." Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on. IEEE, 2005.

[3] Fax, J. Alexander, and Richard M. Murray. "Information flow and cooperative control of vehicle formations." IEEE transactions on automatic control 49.9 (2004): 1465-1476.

[4] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. Parallel computing, 25:789812, 1999.

[5] Raja, Haroon, and Waheed U. Bajwa. "Cloud K-SVD: Computing data-adaptive representations in the cloud." Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on. IEEE, 2013.

[6] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In Decision and Control, 2007 46th IEEE Conference on, pages 5492 5498, Dec. 2007.

[7] S. Barbarossa and G. Scutari. Decentralized maximum-likelihood estimation for sensor networks composed of nonlinearly coupled dynamical systems. Signal Processing, IEEE Transactions on, 55(7):3456 3470, 2007.

[8] B. Ermentrout. An adaptive model for synchrony in the firefly pteroptyx malaccue. J. Math. Biology, 29(6):571-585, June 1991

[9] Xiao, Lin, and Stephen Boyd. "Fast linear iterations for distributed averaging." Systems & Control Letters 53.1 (2004): 65-78.

[10] Xiao, Lin, Stephen Boyd, and Sanjay Lall. "Distributed average consensus with time-varying metropolis weights." Automatica (2006).
[11] Chen, Yin, et al. "Corrective consensus: Converging to the exact average." Decision and Control (CDC), 2010 49th IEEE Conference on. IEEE, 2010.
[12] Chen, Yin, et al. "Corrective consensus with asymmetric wireless links." Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on. IEEE, 2011.
[13] Chen, Yin, et al. "Accelerated corrective consensus: Converge to the exact average at a faster rate." American Control Conference (ACC), 2011. IEEE, 2011.
[14] Kar, Soummya, and Jos MF Moura. "Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise." IEEE Transactions on Signal Processing 57.1 (2009): 355-369.
[15] Kar, Soummya, and Jos MF Moura. "Distributed consensus algorithms in sensor networks: Quantized data and random link failures." IEEE Transactions on Signal Processing 58.3 (2010): 1383-1400.
[16] Mehyar, Mortada, et al. "Asynchronous distributed averaging on communication networks." IEEE/ACM Transactions on Networking (TON) 15.3 (2007): 512-520.
[17] Wu, Jian, and Yang Shi. "Average consensus in multi-agent systems with time-varying delays and packet losses." American Control Conference (ACC), 2012. IEEE, 2012.
[18] Zhang, Ya, and Yu-Ping Tian. "Consensus of data-sampled multi-agent systems with random communication delay and packet loss." IEEE Transactions on Automatic Control 55.4 (2010): 939-943.
[19] Hadjicostis, Christoforos N., Nitin H. Vaidya, and Alejandro D. Domnguez-Garca. "Robust distributed average consensus via exchange of running sums." IEEE Transactions on Automatic Control 61.6 (2016): 1492-1507.
[20] Fagnani, Fabio, and Sandro Zampieri. "Average consensus with packet drop communication." SIAM Journal on Control and Optimization 48.1 (2009): 102-133.
[21] Khosravi, Adel, and Yousef S. Kavian. "Challenging issues of average consensus algorithms in wireless sensor networks." IET Wireless Sensor Systems 6.3 (2016): 60-66.
[22] Khosravi, Adel, and Yousef Seifi Kavian. "Broadcast Gossip Ratio Consensus: Asynchronous Distributed Averaging in Strongly Connected Networks." IEEE Transactions on Signal Processing 65.1 (2017): 119-129.
[23] Eyal, Ittay, Idit Keidar, and Raphael Rom. "LiMoSense: live monitoring in dynamic sensor networks." Distributed computing 27.5 (2014): 313-328.
[24] Postel, J., "Transmission Control Protocol" RFC 761, USC/Information Sciences Institute, January 1980.
[25] Kriegleder, Maximilian, Raymond Oung, and Raffaello D'Andrea. "Asynchronous implementation of a distributed average consensus algorithm." Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. IEEE, 2013.