SwarmDAG: A Partition Tolerant Distributed Ledger Protocol for Swarm Robotics*

Jason A. Tran[1], Gowri Sankar Ramachandran[1], Palash M. Shah[1], Claudiu B. Danilov[2], Rodolfo A. Santiago[2], and Bhaskar Krishnamachari[1]

[1] Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA {jasontra, gsramach, palashms, bkrishna}@usc.edu

[2] Boeing Research & Technology, Huntington Beach, CA, USA {claudiu.b.danilov, rodolfo.a.santiago}@boeing.com

Abstract. Blockchain technology has the potential to disrupt applications beyond cryptocurrency. This work applies the concepts of blockchain technology to swarm robotics applications. Swarm robots typically operate in a distributed fashion, wherein the collaboration and coordination between the robots are essential to accomplishing the application goals. However, robot swarms may experience network partitions either due to navigational and communication challenges or in order to perform certain tasks efficiently. We propose a novel protocol, SwarmDAG, that enables the maintenance of a distributed ledger based on the concept of extended virtual synchrony while managing and tolerating network partitions.

1 Introduction

Blockchain technology has not only made a big impact on finance by enabling cryptocurrencies, it has emerged as a fundamental tool for building all kinds of decentralized applications that can benefit from having a shared, immutable distributed ledger. This work demonstrates how such a distributed ledger could be built for swarm robotics applications.

Swarm robotics applications comprise of a collection of robots that coordinate and collaborate with one other through a set of control algorithms in a connected network and typically operate with limited global knowledge [3]. In real-world application settings, swarm robots are subject to network partitions due to navigational and connectivity challenges, and such partitions can disrupt the application flow and reduce the effectiveness of the robotic agents. As a result, partition tolerance and the ability to adapt to network partitions are important requirements for swarm robotics applications.

The CAP theorem specifies that a distributed system must achieve a tradeoff among the following three properties: consistency, availability, and partition tolerance [4]. In the case of swarm robotics, partition tolerance becomes a prominent requirement to achieve desired application functionality where swarms have

* Supported by Boeing Research & Technology.

the potential to seamlessly partition and merge their network. In this work, we focus on ensuring the availability of swarm robotics applications running over a distributed ledger. However, per the CAP theorem, ensuring partition tolerance implies that swarm robotics applications cannot maintain consistency without sacrificing availability. Swarms can achieve eventual consistency if the partitions merge together within a reasonable time span for the application of interest and follow a protocol for reconciling any conflicts as per the model presented by BASE [13].

Partition tolerance in swarm robotics is not well-studied in the literature. Existing approaches for partition tolerance in robotics applications rely on distributed control theory [6, 9] or consensus-based on graph theory [8, 15]. A particularly interesting consistency concept for robotics applications that experience network partitions is Extended Virtual Synchrony (EVS) [10], which we adopt in this work. Rather than trying to guarantee a globally consistent state at all nodes in the system, EVS provides agreement only between nodes that can exchange information with each other. EVS introduces the concept of *membership view*, which defines a set of nodes that are connected to each other and an identifier for that set. For a certain view identifier (ID), all nodes in the membership view agree on the content of the set. For a membership view ID, consensus is provided only between the nodes in the view set, and there are no guarantees regarding ordering of messages exchanged in different membership views. For nodes or applications that transition together between multiple membership views, view IDs are ordered with respect to message IDs.

One immediate benefit of using EVS in swarm robotics is that, in the presence of partitions, all partitions can continue updating their progress rather than only one partition continuing (often the partition with a majority of the members) as per the semantics provided by global ordering protocols such as Paxos [7] and Raft [11]. The drawback is that when different network partitions merge, in the absence of a global agreement of the updates, the different network components may be in different states that may be conflicting. We believe this better reflects actual environments, and swarm robotics applications need to handle conflicting actions that may have occurred across partitions. Our approach of using a DAG structured distributed ledger based on EVS provides such a mechanism for deconfliction.

In this work, we propose the SwarmDAG protocol, an approach based on a Directed Acyclic Graph (DAG) structured distributed ledger to enable robot swarms to achieve eventual consistency. SwarmDAG consists of membership management and a distributed ledger to handle network partitions. While our work is inspired by prior work on DAG-based distributed ledger protocols such as IOTA [12] and Hashgraph [2], these protocols are not designed for swarm robotics where partitions may be frequent and do not provide explicit support for an EVS-based consistency model. To the best of our knowledge, SwarmDAG is the first partition tolerant distributed ledger technology for swarm robotics applications that supports the EVS consistency model.

2 SwarmDAG: A Partition Tolerant Distributed Ledger

The SwarmDAG protocol is a system-level architecture which provides an EVS-based, eventually consistent, and DAG-based distributed ledger across a robot swarm which encounters frequent partitions. The main idea of SwarmDAG is to respond to partitions by spawning new network instances with corresponding ledgers. The new ledgers that are created become forks of the original ledger before the partitioning occurred. When the partitions merge back together, SwarmDAG provides a protocol to reconstruct the merged ledger while maintaining the fork point. Thus, the resulting ledger structure is a DAG. An example of this fork and merge process can be seen in time slots 0 to 4 in Fig. 1. While a DAG ledger structure is always the result, SwarmDAG is flexible in that it allows the ledger structure within a partition instance to be either a DAG or a blockchain (i.e., a linear ledger). In addition, any consensus protocol within a single partition instance can be used, such as a classical Byzantine Fault-Tolerant (BFT) consensus protocol. Because any ledger consensus protocol can be used within a single partition, the SwarmDAG protocol is intended to be implementable using existing distributed ledger consensus platforms such as Tendermint [16] or Hyperledger [14]. The novelty of SwarmDAG is that it provides a system-level approach to handling partitions while providing an EVS-based eventually consistent distributed ledger.

2.1 SwarmDAG Ledger Overview

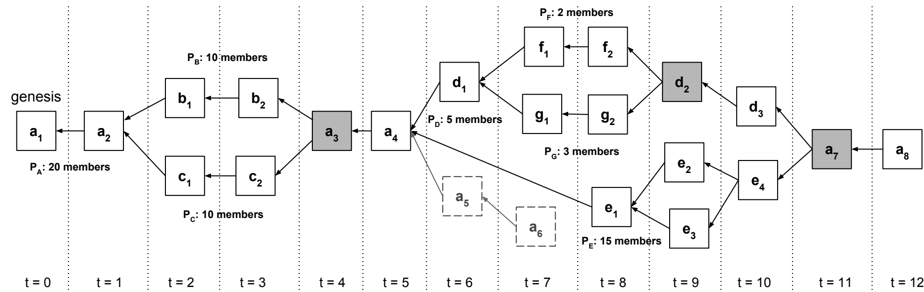

Fig. 1. Example SwarmDAG ledger when network partitions occur and merge. Details of the sequence of events leading up to this example ledger is described in Section 2.1

The SwarmDAG protocol assumes the members of an entire swarm S is known. Fig. 1 illustrates an example SwarmDAG ledger when a swarm of 20 robots (i.e., $|S| = 20$) experiences several network partitions and merges. Each transaction on the SwarmDAG ledger stores state information updates. We assume that one or more transactions are placed into a block, and that a sequence of blocks is created within each partition.

To better understand the SwarmDAG protocol, we will explain in detail the sequence of events leading to the example SwarmDAG ledger illustrated in Fig. 1. A SwarmDAG genesis block a_1 is first created by partition P_A. Partition P_A includes the entire swarm (i.e., swarm $S = P_A$). Blocks appended to the ledger at any time will have the ID of the partition in which the block occurred. In Fig. 1, each block is labeled by a single letter which represents the partition ID with a subscript denoting the transaction number increasing in chronological order (similar to the notion of "height" in blockchains). For illustrative purposes, Fig. 1 is broken into time slots, although it is important to note that SwarmDAG does not use time as a means of determining the order of transactions or blocks on the DAG ledger. At $t = 2$, the network splits into two partitions, P_B and P_C, each with 10 members. The SwarmDAG protocol spawns a new network and corresponding distributed ledger (i.e., resulting in two new genesis blocks) for each new partition. The two partitions continue in parallel with blocks b_1 and c_1 as the first blocks of P_B and P_C, respectively. These new parallel ledgers will continue to grow during the duration of the network split. When the two partitions merge at slot $t = 4$, a reconciliation process takes place in which the now merged partitions (P_B and P_C merge to recreate partition P_A) distribute via a gossip-based protocol only the *finalized* transactions on their independent ledgers and reconstruct the ledger from slots $t = 0$ to $t = 3$ to now have a fork after transaction a_2. We use the term *finalized* for blocks that are officially appended to the SwarmDAG ledger, and the term *confirmed* for blocks that are appended to a ledger within a partition network instance. At this point, the ledger manifests its DAG structure. Since P_B and P_C spawned new networks, their ledgers cannot simply merge together. Thus, each robot in the swarm will manage their ledgers in a local database which only holds the SwarmDAG finalized blocks. Lastly, a new network and ledger is spawned with merge block a_4 serving as the first block of the recreated P_A concluding the merge process. For the protocol to operate as explained, a method for managing partition membership updates regularly is needed. The method used in SwarmDAG is explained in detail in Sec. 2.2. However, readers can assume that the swarm is able to adequately capture partition changes for the rest of this section.

The ledger in Fig. 1 progresses from slot $t = 5$ and onward in a fashion similar to the first 5 slots. However, slots 5 and onward show several special cases which SwarmDAG may encounter when the underlying consensus protocol within a partition is a classical BFT 2/3 consensus protocol. A network split occurs in $t = 6$, but transactions a_5 and a_6 still become confirmed with respect to P_A. This is because this network splits into partitions of size 5 and 15, and the partition of size 15 can still confirm (not finalize) blocks a_5 and a_6 since the leftover members is larger than the 2/3 threshold. Eventually, however, the partition of size 15 will encounter a partition membership update and realize that a network split has occurred leading to the formation of partition P_E. In this example, a network partition was detected two blocks after it occurred, and P_E decides blocks a_5 and a_6 were not finalized (i.e., because it cannot be guaranteed that P_D indeed saw those two blocks). To this end, SwarmDAG must

require a k-block orphaning policy where, at any point in time, the last k blocks should not be considered final, even though the underlying consensus protocol has deemed the block confirmed, to prevent conflicts across partitions. When properly designed, this k-block orphan rule will minimize the possible conflicts robot swarms will face. The number of blocks orphaned should be chosen based on the transaction rate of the network and the expected time in which partition membership changes can be detected in the network. In this example, P_D did not orphan any blocks after dectecting a membership change because the 5 robots were unable to confirm any new blocks to the P_A ledger when the split occurred. Lastly, a fork is illustrated in $t = 9$ for P_E to illustrate a DAG ledger structure can be used within partitions.

2.2 Membership Management Service (MMS)

In the SwarmDAG protocol, every transaction must include the ID of the partition in which the transaction was made to aid in block finalization upon splits and merges. To support this, the SwarmDAG protocol includes the Membership Management Service (MMS), which is a service that shall run on each robot.

At its core, the MMS uses the Totem Membership Protocol [1] but with slight modifications. To control the network traffic and rate of partition membership changes of the Totem Membership Protocol, all nodes will stay in Totem's Gather State until a periodic *Membership Update (MU)* event occurs. When these periodic MU events occur, all partitions in the network will move to Totem's Commit State. At all times, each robot node will maintain a peer list and partition membership list. The peer list includes a list of nodes within communication range which is updated regularly based on any broadcast packets received by robots within communication range. Broadcast packets are received as a part of the background gossip messages of the underlying distributed ledger consensus protocol used as well as gossip messages generated by the MMS. The MMS gossip messages include a robot's unique identifier as well as its peer list. At regular periods, T_{MU}, these gossiped peer lists will be processed to determine partition MU events. T_{MU} should be chosen long enough to not overload the network but short enough to capture the expected interval of partition changes in the swarm network. A partition requires a minimum of two robots, so any lone robot nodes are addressed by its identifier. Lastly, the MMS provides applications access to the current partition membership to allow intelligent decisions on when to submit particular transactions onto the network.

3 Partition-Aware Transaction Verification (PTV) Policy

Applications can use the SwarmDAG ledger as a voting platform to reach consensus on items such as task allocation and decision making as proposed by the authors in [5]. However, it is important to note consensus on these items is eventual in the face of partitions. For example, when a collaborative task is proposed by a robot in a partition with insufficient robots, the task cannot be accepted

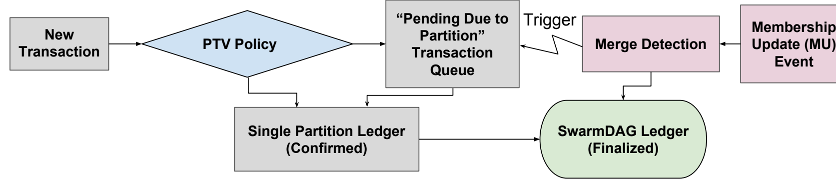

Fig. 2. Partition-Aware Transaction Verification (PTV) Policy.

and consensus has not yet been reached. Partitions containing a sufficient number of robots need to merge with the partition of the proposing robot for the swarm to come to consensus. Thus, it is important to have a Partition-Aware Transaction Verification (PTV) policy to aid in optimizing consensus time and reducing possible conflicts. The SwarmDAG PTV policy allows applications to decide when to queue transactions when insufficient robots are available. The SwarmDAG PTV flow chart is illustrated in Fig. 2.

4 Target Properties and Ongoing Work

As we have noted above informally, the aim of the SwarmDAG protocol is to provide an EVS-based eventually consistent distributed DAG ledger for a robot swarm. More precisely, what we seek to provide is the following guarantee: at any moment in time, a given node i in the swarm S will contain a subset of the global DAG-ledger (the union of DAG-ledgers contained in all nodes across all partitions in the swarm) such that a) it includes all finalized blocks produced in the partitions that node i was ever part of, and b) it includes all finalized blocks present in the DAG-ledger of all nodes j that i ever encountered in the same partition as itself, right up to the last time it encountered them. This guarantee can be proved to hold through an inductive argument, which we will present formally in future work.

In future work, we intend to investigate the tradeoff of choosing the membership update period T_{MU} as well as the tradeoffs when designing a k-block orphaning policy when network splits occur. To do so, we are currently evaluating the SwarmDAG protocol by implementing it using various distributed ledger consensus platforms (e.g. IOTA, Tendermint) for each partition network instance.

5 Conclusion

Swarm robotics applications are susceptible to network partitions, which can make it challenging to satisfy desired application goals such as maintaining a distributed ledger for decentralized applications. We have presented here a novel protocol called SwarmDAG which manages a distributed ledger to provide extended virtual synchrony based eventual consensus while handling partitions through a membership management protocol.

References

1. Amir, Y., Moser, L.E., Melliar-Smith, P.M., Agarwal, D.A., Ciarfella, P.: The totem single-ring ordering and membership protocol. ACM Trans. Comput. Syst. **13**(4), 311–342 (Nov 1995). https://doi.org/10.1145/210223.210224, http://doi.acm.org/10.1145/210223.210224
2. Baird, L.: The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Swirlds, Inc. Technical Report SWIRLDS-TR-2016 **1** (2016)
3. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence **7**(1), 1–41 (Mar 2013). https://doi.org/10.1007/s11721-012-0075-2, https://doi.org/10.1007/s11721-012-0075-2
4. Brewer, E.: Cap twelve years later: How the "rules" have changed. Computer **45**(2), 23–29 (Feb 2012). https://doi.org/10.1109/MC.2012.37
5. Ferrer, E.C.: The blockchain: a new framework for robotic swarm systems. CoRR **abs/1608.00695** (2016), http://arxiv.org/abs/1608.00695
6. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. Frontiers in Robotics and AI **3**, 29 (2016)
7. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (May 1998). https://doi.org/10.1145/279227.279229, http://doi.acm.org/10.1145/279227.279229
8. Li, Z., Wen, G., Duan, Z., Ren, W.: Designing fully distributed consensus protocols for linear multi-agent systems with directed graphs. IEEE Transactions on Automatic Control **60**(4), 1152–1157 (April 2015). https://doi.org/10.1109/TAC.2014.2350391
9. Lopes, Y.K., Trenkwalder, S.M., Leal, A.B., Dodd, T.J., Groß, R.: Supervisory control theory applied to swarm robotics. Swarm Intelligence **10**(1), 65–97 (2016)
10. Moser, L.E., Amir, Y., Melliar-Smith, P.M., Agarwal, D.A.: Extended virtual synchrony. In: 14th International Conference on Distributed Computing Systems. pp. 56–65 (June 1994). https://doi.org/10.1109/ICDCS.1994.302392
11. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference. pp. 305–320. USENIX ATC'14, USENIX Association, Berkeley, CA, USA (2014), http://dl.acm.org/citation.cfm?id=2643634.2643666
12. Popov, S.: The tangle, iota whitepaper (2018)
13. Pritchett, D.: Base: An acid alternative. Queue **6**(3), 48–55 (May 2008). https://doi.org/10.1145/1394127.1394128, http://doi.acm.org/10.1145/1394127.1394128
14. The Linux Foundation: Hyperledger, https://www.hyperledger.org
15. Silvestre, D., Hespanha, J.P., Silvestre, C.: Broadcast and gossip stochastic average consensus algorithms in directed topologies. IEEE Transactions on Control of Network Systems pp. 1–1 (2018). https://doi.org/10.1109/TCNS.2018.2839341
16. Tendermint: http://tendermint.com