Byzantine Fault Tolerant Consensus for Lifelong and Online Multi-Robot Pickup and Delivery

Kegan Strawn and Nora Ayanian

University of Southern California, Los Angeles, CA, 90007, USA
{kegan.j.strawn, ayanian}@usc.edu

Abstract. Lifelong and online Multi-Agent Pickup and Delivery is a task and path planning problem in which tasks arrive over time. Real-world applications may require decentralized solutions that do not currently exist. This work proposes a decentralized and Byzantine fault tolerant algorithm building upon blockchain that is competitive against current distributed task and path planning algorithms. At every timestep agents can query the blockchain to receive their best available task pairing and propose a transaction that contains their planned path. This transaction is voted upon by the blockchain network nodes and is stored in the replicated state across all nodes or is rejected, forcing the agent to re-plan. We demonstrate our approach in simulation, showing that it gains the decentralized Byzantine fault tolerant consensus for planning, while remaining competitive against current solutions in its makespan and service time.

Keywords: Robotics · Blockchain · MAPD · Consensus.

1 Introduction

Path planning for teams of robots is a fundamental problem for many applications of multi-robot systems. Often, teams of robots must reliably navigate to given locations without collisions. In Multi-Agent Pickup and Delivery (MAPD), given a set of tasks with known locations, agents must find an assignment of tasks to agents and plan the paths necessary to arrive at the locations of those tasks. We are interested in the lifelong and online version of MAPD (LO-MAPD), wherein tasks arrive over time for indefinite periods, in known environments. Examples include warehouse robots [33], video game characters [26], aircraft-towing vehicles [21], and general autonomous mobile robots [30].

In this work we focus on MAPD in a warehouse environment with multiple interested parties operating within the same system with no central authority. Here, multiple self-serving agent teams must collaborate to accomplish all tasks without complete trust in the other agents. Therefore, not only do centralized algorithms scale poorly for large numbers of agents, but are not a valid choice. Current distributed algorithms attempting to solve this, generally, produce longer paths for the overall system and make large assumptions on the behavior of agents. We assume all agents continuously seek to accomplish tasks,

but make no further assumptions about the behavior of agents. For example, we do not assume that agents seek to perform the best tasks for the overall system or that agents do not attempt to plan paths with collisions. In this way, agents may be selfish or uncooperative and introduce Byzantine failures in planning that break current solutions.

These Byzantine failures are faults beyond simple failures (such as losing a message or agent) where there is imperfect information on whether there has been a failure. Existing work on swarm and multi-robot systems often consider themselves failure resistant due to the number of agents, but in many cases a single Byzantine failure is enough to stop the system from working [27]. Outside of the MAPD problem, blockchain based solutions have recently been proposed as a potential solution to provide Byzantine fault tolerance, a distributed and immutable storage, and anonymity for multi-robot systems [1, 5, 6, 16, 27].

Our algorithm builds upon a blockchain framework to provide Byzantine fault tolerance in planning for the decentralized warehouse MAPD problem. In our system and algorithm, each individual agent has a limited scope of information. They receive only the current timestep, the used location-timestep pairs planned so far to avoid collisions, the endpoint locations of agents where they stay indefinitely, and their best available task without any other attached agent or task information. This reduces the amount of state information sent between distributed agents and enables a larger system where multiple teams of agents operate without conflicts and without sharing identifiable data or trusting any single source for assignments. As we will show with our simulation results, our work gains this decentralized Byzantine fault tolerant consensus from building on top of the blockchain for task and path planning, while remaining competitive against current distributed solutions in its makespan and service time.

2 Background

2.1 MAPF and MAPD

MAPD is an extension and generalization of the popular Multi-Agent Path Finding (MAPF) problem. In the MAPF problem paths must be found for all agents from their starting locations to their destinations without colliding with other agents or obstacles in the environment. Much work exists on the MAPF problem, its variants, and benchmarks [19]. Similar to Target Assignment and Path Finding (TAPF) [18], MAPD extends MAPF to incorporate tasks that are composed of pickup and delivery locations agents must move to in order to accomplish them.

We address discrete LO-MAPD, wherein a stream of tasks arrive over time in an online setting on a general graph. The goal of the system is to accomplish each task while minimizing the cost. Depending on the application, costs could be: service time (the average number of timesteps needed to execute each task), the makespan (the last timestep used), and/or the runtime (the time it took to execute the planning of all tasks). To successfully solve a MAPD instance the

algorithm must have a bounded service time and runtime. While not all MAPD instances are solvable, sufficient conditions exist that ensure solvability; we only work with such instances, called well-formed instances [29].

Many algorithms have been developed to solve the MAPF, TAPF, and MAPD family of problems. Examples include Conflict-Based Search and it's variants [3, 8, 10, 18, 25], Answer Set Programming [23], Enhanced Partial Expansion A* [9], suboptimal algorithms [11, 28, 31, 32], and Windowed-Hierarchical Cooperative A* and similar approaches [15, 24, 26]. For LO-MAPD, Token Passing and Token Passing with Task Swaps are the state of the art distributed solutions [17, 20]. However, the distributed approaches can suffer from deadlocks, make strong assumptions, and sacrifice performance for distribution.

2.2 Blockchain and Byzantine Faults

Our algorithm differs from existing solutions by taking advantage of blockchain technology. Created as part of Bitcoin [22], a blockchain framework is presented as a distributed ledger with peer-to-peer sharing of encrypted and linked data transactions with timestamps. Blockchain technology can provide an immutable state, decentralized consensus, fault tolerance, and a dynamic framework for the flexible control of a network of nodes [2]. A blockchain transaction can be developed to store different forms of data. In this work, we store different types of MAPD information as transactions on the blockchain. Each transaction has a string labeling the type of data (for example: a timestep, used timestep-location pairs, or task assignment) and the data itself in string format.

In the Byzantine Generals Problem, a set of generals with no trust in each other surround a castle and can only communicate via a messenger to agree upon when to attack. Lamport et al. [13] present this problem and prove a well-designed system can survive 1/3 of these Byzantine failures where the generals are unresponsive or unreliable. There exists a thorough body of work on Byzantine failures in collaborative networks in distributed systems literature [14]. In distributed systems, to be Byzantine fault tolerant (BFT), nodes in the network must agree regularly about the current state and have a 2/3 majority agreement, surviving 1/3 of all nodes differing in their judgement [7]. A blockchain can solve the Byzantine Generals Problem using cryptography and various validation models. Each blockchain framework defines the effort, investment, and resources a node must put into the system and the outcome of the vote in order to incentivize and maintain honest and positive behavior. It is important to note that the use of blockchain alone does not necessarily provide Byzantine fault tolerance, but that our work here builds upon blockchain to provide this additional fault tolerance in a specific multi-robot setting. In the context of robotics, the definition and categorization of Byzantine faults is not yet widely agreed upon. We focus on a subset of arbitrary behavior in adversarial agents' planning and communication, although Byzantine faults could take more advanced forms in real-world applications.

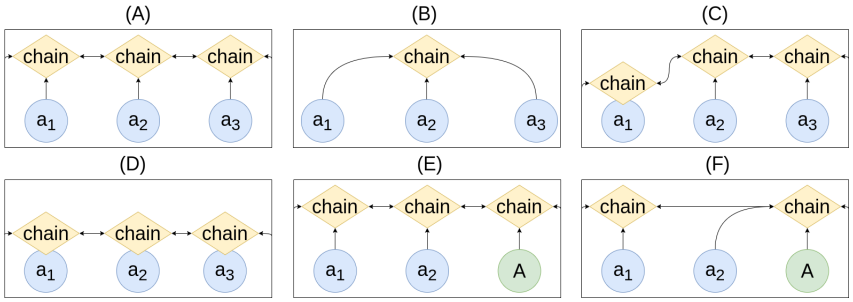

Fig. 1. Potential node-agent configurations. Yellow diamonds are Tendermint ABCI nodes running our BFTC Chain application. Blue circles a_1, a_2, a_3 are agents. A circles represent the simulation agent that sets the timestep and other simulation/environment variables on the blockchain.

2.3 Tendermint

Tendermint provides an Application Blockchain Interface (ABCI) for a peer-to-peer message passing protocol that checks programmer-defined functions to validate messages passed across the blockchain network. Each proposed transaction goes through Tendermint's voting process and provides a flexible interface to integrate the blockchain into our MAPD algorithm. The setup of the Tendermint network is flexible. Each MAPD agent in the system could run the Tendermint node as well as our algorithm, however, if an agent has insufficient computational resources it could communicate to the Tendermint node running on another computing system. In Fig. 1, some possible configurations are shown, such as all agents connected to separate nodes, all agents connected to the same node, one agent running the node on its local system, an agent A that can interact with the blockchain but does not accomplish tasks, and various other possible compositions for all nodes and agents.

We have designed a Tendermint ABCI application, called the Byzantine Fault Tolerant Consensus (BFTC) chain, that implements this interface and is similar to a key-value database to store used locations, edges, and endpoints as well as the current timestep and the set of available tasks. Tendermint runs this application on all nodes in the Tendermint network. The BFTC chain can be queried by MAPD agents running our algorithm in two ways. It can be sent either a Tendermint DeliverTX message that stores the transaction as part of the state if agreed upon by at least 2/3rds of all chain nodes as a valid transaction (such as no collisions), or a Tendermint Query message that returns identical replicated information about the state across all nodes that are part of the chain's network.

3 Methods

Our proposed BFTC system solves well-formed, LO-MAPD instances by building an application on top of the blockchain platform Tendermint [4] to reach

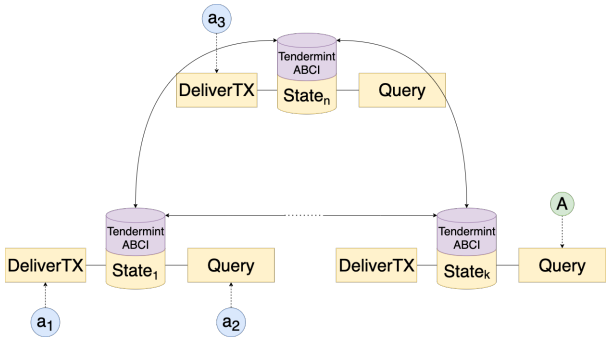

Fig. 2. A visual representation of our BFTC system. Agents send DeliverTX or Query messages to the nodes, who use the ABCI to utilize Byzantine fault tolerant Tendermint consensus before storing the transactions in the state storage on each node. All agents $a_1, \ldots, a_n$ communicate with nodes $1, \ldots, n$ that maintain identical states $state_1, state_2, \ldots, state_n$.

consensus on valid task assignments and valid paths in a decentralized network at every timestep. We assume agents must communicate on the network to receive and accomplish tasks, but not that all agents propose valid transactions. All valid agents follow our BFTC algorithm (Algorithm 1) simultaneously. All agents send messages to a Tendermint node on the network to receive a task and information to plan their paths, as well as propose these planned paths as transactions to collectively accomplish all tasks. In contrast, current state of the art algorithms pass full state knowledge around to each agent in a predefined order based on their ID number [20].

We assume an agent rests in the last location of its path and each agent plans its individual path using A*. Paths are sent as transactions, and, if valid, are stored in the distributed ledger that represents the world state with the order of transactions determined by the order of consensus rather than the order determined by a central system. The BFTC chain application has DeliverTX methods to handle requests for: adding a new timestep, adding a new task, posting an agent's location, assigning an agent to a task and storing the path to the task, storing an agent's path to a safe endpoint, and checking in for an agent at a timestep. The chain has Query methods to handle requests for: the current timestep, current delivery spots, best task for a specific agent's location (if any), used timestep-location pairs, edges, and endpoints, as well as if all agents have checked in for the current timestep. We use an agent A that communicates with the chain via a Tendermint node in order to increment each timestep once all agents have checked in for the previous timestep, adding any new tasks to the task set, and logging any accomplished task pickups or deliveries. This allows us to follow the same experimental model as in [20]. A visual representation of our BFTC system is shown in Fig. 2.

Every valid, non-Byzantine failing, node composing the decentralized BFTC chain simultaneously uses the Hungarian Method [12] in a decentralized fashion

to find the best possible unassigned task-agent location pairings for the overall system, at every new timestep. To avoid collisions after assignments, our A* search and validation logic works similar to [20], planning collision-free paths using the time-location pairs and edges, called usedElements in Alg. 1. Path costs only need to be found from a location to another endpoint, so heuristic values for A* are pre-computed. The chain's Hungarian method only uses this heuristic value map and does not compute the A* solution at every timestep for every task, saving computation resources and time. This new best possible task-agent pairing allows our decentralized BFTC algorithm to assign agents to tasks that might not be the closest to them, but minimizes the overall cost for the system. Thus, even without failures, our decentralized solution has the potential to outperform certain distributed algorithms; this is discussed further in Section 5.1.

Each agent begins its outer loop by checking if the state's timestep has been incremented. If it has been, an agent without a task will query the chain for the best possible task assignment for only itself. If it receives one it will use A* to plan a path to the pickup and then the delivery, sending this as a transaction containing a single path to the chain for assignment. If the chain replies with a successful assignment (meaning consensus was reached among Tendermint nodes), the agent will check in for this timestep. If the agent did not receive a best possible task, it will check to see if its current location will cause a collision. If it does, the agent will move out of this endpoint through exhaustive search from it's current location for a safe endpoint with the least cost path before checking in. Otherwise, it will stay in its current location and check in for the next timestep.

4 Benchmark Algorithms

Ma et al. present three algorithms for solving the LO-MAPD problem: two decoupled MAPD algorithms, Token Passing (TP) and Token Passing with Task Swaps (TPTS), and a centralized algorithm, CENTRAL [20]. The TP algorithm involves sending the token that stores all world state data to each agent that needs to choose a task and plan a path. TPTS modifies the TP algorithm to enable swapping tasks before the tasks have been picked up. In TP, all agents must wait until the agents before them have received the token and planned their paths/assignments. In TPTS all agents must wait until the agents before them have planned and all re-planning has taken place before they can be certain of their paths/assignments for that timestep. In decentralized systems this is an important negative attribute, as the system should not prioritize the wait time of certain agents without apparent reason.

Minimizing the sum of costs is NP-hard to solve optimally and NP-hard to approximate the makespan within a constant factor less than 4/3 [34], so they compared their new algorithms with CENTRAL, a centralized strawman MAPD algorithm. CENTRAL uses the Hungarian Method [12] to assign tasks and ECBS [10] to plan conflict-free paths at every timestep in a simulated ware-

Algorithm 1: BFTC (Byzantine Fault Tolerant Consensus)

```
for each agent a_i in A do
    a_i.path = loc(a_i);
    /* Start all agent processes, calling IndividualAgent(a_i, chain) */
while true do
    Add all new tasks if any to the task set T;
    Put the new timestep on the chain;
    while ∃ a_i not checked-in in the chain for the timestep do
        /* system waits, with timeout */
    AdvanceTimestep(chain);
    /* Move agents one step */
Function IndividualAgent(a_i, chain):
    while true do
        success ← False;
        if a_i has no task then
            τ ← QueryBestTaskAssignment(loc(a_i), chain);
            while τ and not success do
                usedElements← QueryUsedElements(chain);
                path ← AStarPath(loc(a_i), τ, usedElements);
                success ← DeliverPath(path, chain);
                if success then
                    assign a_i to τ;
                    a_i.path = path;
                    break;
        if a_i has no task then
            if no τ_j ∈ T with g_j == loc(a_i) then
                while not success do
                    usedElements← QueryUsedElements(chain);
                    path ← AStarPath(loc(a_i), loc(a_i), usedElements);
                    success ← DeliverPath(path, chain);
                a_i.path = path;
            else
                while not success do
                    usedElements ← QueryUsedElements(chain);
                    safeEndpoint ← FindSafeEndpoint(loc(a_i));
                    path ← AStarPath(loc(a_i), safeEndpoint, usedElements);
                    success ← DeliverPath(path, chain);
                a_i.path = path;
        DeliverCheckIn(a_i, chain);
        while Not AdvancedTimestep(timestep, chain) do
            timestep = GetTimestep(chain);
```

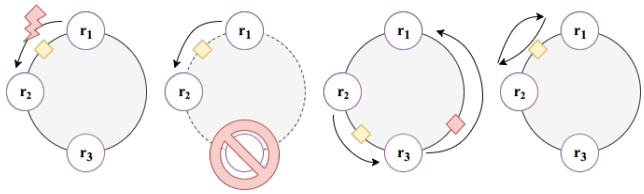

Fig. 3. TP and TPTS use the token passing system. This system can have multiple consensus failures if robots r_1, r_2, r_3, have limited trust in each other, if the network connection is faulty, or if agents can through error or intent reach false conclusions. From left to right are four examples of potential planning Byzantine failures in the MAPD problem: if passing the token fails, if an agent fails, if an incorrect message is passed, or if a cycle occurs when an agent incorrectly believes it has the better path.

house environment. CENTRAL provides a best possible benchmark to compare the distributed algorithms against, not to beat. They proved that their algorithms solve a realistic subclass of well-formed MAPD instances and claim that TP can be extended to a fully distributed MAPD algorithm. Because of this distribution of computation it is their recommended choice for a real-time solution of MAPD instances with large numbers of agents and tasks. Their second algorithm TPTS requires more communication, and is presented as an in-between solution that trades computational complexity for improved solution quality. While presented as a distributed solution, these algorithms assume full trust between agents which restricts them from being decentralized, as shown in Fig. 3.

In our work we assume that all agents seek to accomplish tasks and that at most 1/3rd of the nodes support faulty Byzantine transactions. Our algorithm is similar to the algorithms presented in [20] and solves all well-formed MAPD instances, but in comparison reduces the assumptions made about the network of agents, uses our novel blockchain consensus framework rather than token passing, and allows swapping of tasks during rounds (not between them).

The TP and TPTS algorithms work well for general MAPD, but fail to work in the presence of multi-team collaboration and communication failures. Thus, by comparing against these algorithms we can see that our algorithm performs just as well, while also successfully working in the multi-team collaboration scenario with and without failures.

5 Experimental Evaluation

We ran our simulations on a 5.0 GHz Turbo Intel Core i9-9900K desktop computer with 16GB RAM. We implemented TP, TPTS, CENTRAL, and BFTC in Python to run on a simulated warehouse environment with obstacles as shown in Fig. 4. We implemented all algorithms in Python, with the exception of CENTRAL which uses an existing ECBS implementation in C++ [10].

We generated a single set of 500 randomly selected tasks to be used across all algorithms, task frequencies, and number of agents. Five different task frequen-

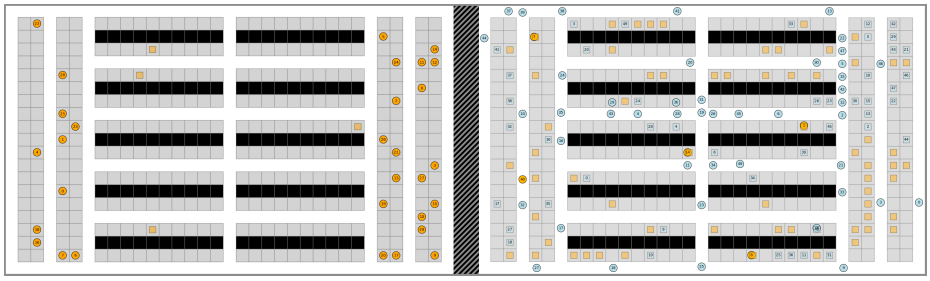

Fig. 4. This figure shows two separate captures of the output of our simulation: a 4-neighbor grid that represents the layout of a simulated warehouse environment. Shown on the left of the divider is the output of the simulation with 30 agents at their initial locations. The black cells are obstacles, the gray cells are endpoints, the orange squares are pickup locations of released tasks, and the orange colored circles are the initial locations of agents and their IDs. Shown on the right of the divider is a capture of the simulation output as it is running. The blue circles are agents that have picked up a task and are delivering them to their corresponding delivery spot ID number.

Table 1. Simulated Warehouse Comparison

		Makespan (timesteps)				**Service Time (timesteps)**				**Runtime per Timestep (sec)**			
Frequency	Agents	BFTC	TP	TPTS	CENTRAL	BFTC	TP	TPTS	CENTRAL	BFTC	TP	TPTS	CENTRAL
0.50	10	**1568**	1683	1610	1261	154.15	152.19	**142.31**	83.14	3.45	0.10	0.24	0.08
0.50	30	**1044**	1087	1054	1042	32.65	44.16	**31.67**	24.79	4.66	0.41	11.10	0.11
0.50	50	1050	1065	**1045**	1038	30.70	43.51	**29.64**	23.09	6.07	0.39	19.91	0.15
1	10	**1546**	1555	1570	1204	**198.93**	216.97	200.06	134.78	3.49	0.10	0.16	0.08
1	30	666	700	**645**	546	55.76	56.59	**53.61**	26.71	4.64	0.25	1.20	0.17
1	50	**576**	595	**576**	542	35.51	46.31	**33.52**	23.98	5.43	0.62	19.67	0.22
2	10	1547	1608	**1544**	1186	**209.70**	230.22	213.47	152.41	3.49	0.10	0.36	0.08
2	30	668	654	**639**	470	81.66	86.20	**80.08**	48.93	4.67	0.24	1.18	0.17
2	50	469	495	**457**	318	54.25	60.45	**53.99**	28.09	5.62	0.36	4.21	0.33
5	10	**1549**	1581	1590	1171	221.63	227.56	**218.66**	166.59	3.49	0.09	0.15	0.80
5	30	659	692	**635**	445	**88.56**	96.86	92.79	60.28	4.67	0.21	1.17	0.18
5	50	**453**	506	464	396	**65.22**	72.56	66.75	40.22	5.79	0.32	4.58	0.27
10	10	**1551**	1654	1580	1174	222.16	230.58	**218.59**	165.62	3.47	0.09	0.17	0.08
10	30	673	677	**626**	487	**95.86**	98.20	97.65	61.58	4.59	0.25	1.23	0.19
10	50	**471**	505	480	302	**68.08**	72.99	69.90	42.91	5.57	0.37	4.49	0.36

cies were used to experiment with the rate of new tasks added to the set of live tasks over time: 0.5 (one task every other timestep), 1 (one task every timestep), 2 (two tasks every timestep, 5 (five tasks every timestep), and 10 (ten tasks every timestep). For each task frequency we ran three different agent set sizes. Each agent set was generated ahead of time with random initial agent locations and each set was used for all algorithm and frequency combinations. Table 1 reports the makespan, service time, and runtime per timestep for experiments without Byzantine failures. Figure 6 shows the performance of the three algorithms when two instances of Byzantine failures are present.

5.1 Makespan and Service Time Comparison

The distributed MAPD algorithms in descending order of average makespan across all non-Byzantine failure experiments (best is last) is: TP (1004), TPTS

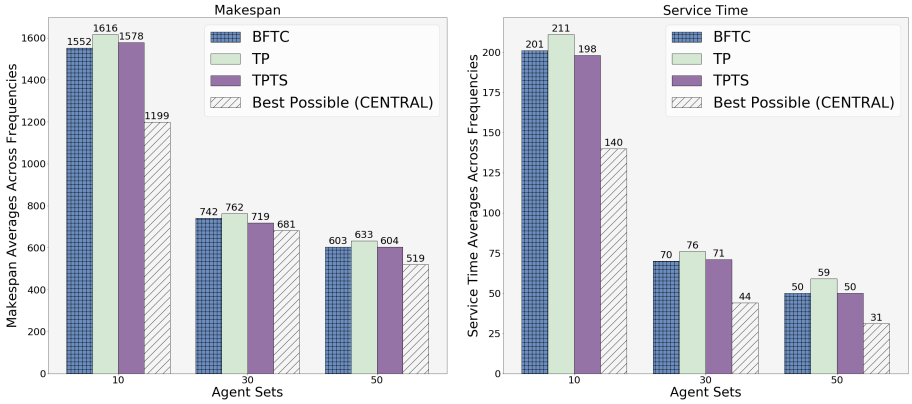

Fig. 5. Makespans and service times averaged across all task frequencies presented for all four algorithms for each set of agents with no failures present. From left to right: BFTC, TP, TPTS, and best possible (CENTRAL). Generally, makespans and service times decrease with more agents as all agent sets serve the same set of tasks.

(967), and BFTC (965). The distributed MAPD algorithms in descending order of average service time across all non-Byzantine failure experiments (best is last) is: TP (115), and both TPTS and BFTC tied with (107). TP lets each agent make the best greedy assignment for itself, producing less optimal makespans. TPTS allows swapping of tasks before the tasks are picked up, decreasing the makespan/service times by improving upon the greedy assignments each individual agent may have made. BFTC is able to take advantage of the replicated state across all nodes in the blockchain network to reply to each agent with the optimal assignment at that timestep, but does not let them swap. Letting them swap could further improve the makespan/service time of BFTC but does not scale well, as more computation and message passing would be necessary. CENTRAL demonstrates the best (sub-optimal) performance possible, outperforming any distributed algorithm, but will not work for systems that cannot have a central authority. Thus, BFTC is just as good, and in some cases better than, current state of the art solutions even in non-Byzantine cases.

5.2 Runtime Comparison

The distributed MAPD algorithms in increasing order of average runtime per timestep across all non-Byzantine failure experiments is: TP, TPTS, and BFTC. The BFTC runtime is longer as it is the only algorithm using the network to send messages and takes on the runtime of the decentralized Tendermint network's consensus protocol. This increases the amount of network messages being sent and BFTC's runtime. In our experiments, each DeliveryTX transaction took 1-2 seconds to be committed or rejected and returned to the agent. At the start of every round, agent A adds any new tasks and increments the timestep on

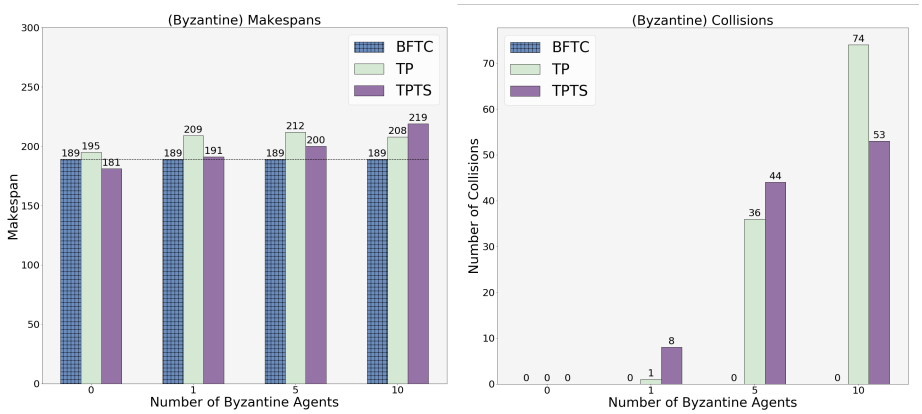

Fig. 6. Makespans and number of collisions presented for the three distributed algorithms as the number of Byzantine agents is increased. From left to right: BFTC, TP, and TPTS. BFTC has zero collisions across all Byzantine agent sets.

the chain through two DeliveryTX calls. Then, agent A must wait for all agents to check in. All agents when planning and checking in make two DeliveryTX calls concurrently with each other. Since each DeliveryTX call takes 1-2 seconds BFTC's average runtime per timestep (across all task frequencies and agent sets) of 4.63 is dominated by the DeliveryTX message passing. In the BFTC algorithm agents are able to plan and receive confirmation of their plan in under 1-2 seconds concurrently with each other.

Using Table 1, we can compare how the algorithms perform as the number of agents increase. Previous work shows that the runtime of CENTRAL and TPTS increases at a much faster rate than TP, and thus CENTRAL and TPTS do not allow for real-time lifelong operation for large numbers of agents [20]. Conversely, while the runtime of BFTC does increase with additional agents, the runtime increases at a slower rate than TPTS and TP. For example, the change of average runtimes per timestep for each distributed algorithm from 10 to 30 agents in decreasing order (best is last) of rate of growth is: TPTS $(1,661\%)$, TP (200%), and BFTC (36%). This shows that the runtime of BFTC grows at a slower rate with the addition of more agents than the other two distributed algorithms.

5.3 Byzantine Failure Comparison

To see how the distributed algorithms performed when Byzantine failures are present we ran two experiments with different types of Byzantine failures in the shared warehouse scenario. In both experiments 30 agents completed a set of 100 tasks with a frequency of 1, varying the number of Byzantine agents $(0, 1, 5, 10)$. The first experiment had Byzantine agents that attempted to assign themselves to the furthest task available. The second experiment had Byzantine agents

ignore the paths of other agents. In Figure 6 we can see BFTC outperforms the other two distributed algorithms in all cases with Byzantine agents present. The distributed MAPD algorithms in decreasing order of average collisions across all Byzantine failure experiments (best is last) is: TP (37), TPTS (35), and BFTC (0). The distributed MAPD algorithms in decreasing order of average makespan across all Byzantine failure experiments (best is last) is: TP (206), TPTS (198), and BFTC (189). The other algorithms show increasing makespans as Byzantine agents are introduced into the system and result in many collisions that increase as the number of Byzantine agents are increased. BFTC remains at the same makespan it had before Byzantine agents were introduced into the simulation and never results in a collision.

6 Conclusion

In this paper, we present a system that solves the lifelong and online MAPD problem by building on top of a blockchain framework to maintain a distributed ledger of assigned tasks and planned paths that agents can query for limited information to individually plan their paths. Our Byzantine Fault Tolerant Consensus algorithm and system demonstrated better makespans and service times and scaled better in runtime as the number of agents increased than state of the art distributed algorithms. BFTC is also the only algorithm able to run in a completely decentralized network with no central authority and is the only algorithm that can survive Byzantine failures in planning and communication that resulted in collisions and increased makespans in the state of the art algorithms. The BFTC algorithm and system is an example of how blockchain can be used in multi-robot and swarm systems to enable consensus when there is no central authority or trust among robots and the deployment of collaborative, decentralized, multi-robot systems.

There are potential trade-offs to consider when using a blockchain-based system: for example, one must consider the latency or throughput of the network, validation models that fit some but not all types of problems, and the computational capabilities of the nodes/robots in the system. While out of the scope of this paper, we consider these not as roadblocks but avenues for related and future research to see what best fits a particular robotic application.

Acknowledgement This work was partially funded by ARL DCIST CRA W911NF-17-2-0181.

References

1. Afanasyev, I., Kolotov, A., Rezin, R., Danilov, K., Kashevnik, A., Jotsov, V.: Blockchain solutions for multi-agent robotic systems: Related work and open questions. In: Proceedings of the 24th Conf. of Open Innovations Association. FRUCT'24, Helsinki, Uusimaa, FIN (2019)

2. Atlam, H.F., Alenezi, A., Alassafi, M.O., Wills, G.: Blockchain with internet of things: Benefits, challenges, and future directions. Int. Journal of Intelligent Systems and Applications **10**(6), 40–48 (2018)
3. Boyarski, E., Felner, A., Stern, R., Sharon, G., Betzalel, O., Tolpin, D., Shimony, E.: ICBS: The improved conflict-based search algorithm for multi-agent pathfinding. In: Eighth annual symposium on combinatorial search. Citeseer (2015)
4. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. Ph.D. thesis, The University of Guelph (2016)
5. Calvaresi, D., Dubovitskaya, A., Calbimonte, J.P., Taveter, K., Schumacher, M.: Multi-agent systems and blockchain: Results from a systematic literature review. In: Demazeau, Y., An, B., Bajo, J., Fernández-Caballero, A. (eds.) Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection. pp. 110–126. Springer Int. Publishing, Cham (2018)
6. Castello, E., Hardjono, T., Pentland, A.: Editorial: Proceedings of the first symposium on blockchain and robotics, mit media lab, 5 dec. 2018. Ledger **4** (04 2019). https://doi.org/10.5195/ledger.2019.179
7. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (Nov 2002). https://doi.org/10.1145/571637.571640
8. Cohen, L., Uras, T., Kumar, T.K.S., Xu, H., Ayanian, N., Koenig, S.: Improved solvers for bounded-suboptimal multi-agent path finding. In: Proceedings of the Twenty-Fifth Int. Joint Conf. on Artificial Intelligence. p. 3067–3074. IJCAI'16, AAAI Press (2016)
9. Goldenberg, M., Felner, A., Stern, R., Sharon, G., Sturtevant, N., Holte, R.C., Schaeffer, J.: Enhanced partial expansion a*. J. Artif. Int. Res. **50**(1), 141–187 (May 2014)
10. Hönig, W., Kiesel, S., Tinka, A., Durham, J.W., Ayanian, N.: Conflict-based search with optimal task assignment. In: AAMAS (2018)
11. Khorshid, M., Holte, R., Sturtevant, N.R.: A polynomial-time algorithm for non-optimal multi-agent pathfinding. In: SOCS (2011)
12. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly **2**, 83–97 (1955)
13. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (Jul 1982). https://doi.org/10.1145/357172.357176
14. Laprie, J.C.: The dependability approach to critical computing systems. In: Nichols, H., Simpson, D. (eds.) ESEC '87. pp. 231–243. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)
15. Li, J., Tinka, A., Kiesel, S., Durham, J.W., Kumar, T.K.S., Koenig, S.: Lifelong multi-agent path finding in large-scale warehouses. In: AAMAS (2020)
16. Lopes, V., Alexandre, L.A.: An overview of blockchain integration with robotics and artificial intelligence. Ledger **4** (Apr 2019). https://doi.org/10.5195/ledger.2019.171
17. Ma, H., Hönig, W., Kumar, T.K.S., Ayanian, N., Koenig, S.: Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. Proceedings of the AAAI Conf. on Artificial Intelligence **33**(01), 7651–7658 (Jul 2019). https://doi.org/10.1609/aaai.v33i01.33017651
18. Ma, H., Koenig, S.: Optimal target assignment and path finding for teams of agents. In: Proceedings of the 2016 Int. Conf. on Autonomous Agents and; Multiagent Systems. p. 1144–1152. AAMAS '16, Int. Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2016)

19. Ma, H., Koenig, S.: Ai buzzwords explained: Multi-agent path finding (mapf). AI Matters **3**(3), 15–19 (Oct 2017). https://doi.org/10.1145/3137574.3137579
20. Ma, H., Li, J., Kumar, T.S., Koenig, S.: Lifelong multi-agent path finding for online pickup and delivery tasks. In: Proceedings of the 16th Conf. on Autonomous Agents and MultiAgent Systems. p. 837–845. AAMAS '17, Int. Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2017)
21. Morris, R., Pasareanu, C., Luckow, K., Malik, W., Ma, H., Kumar, T.K., Koenig, S.: Planning, scheduling and monitoring for airport surface operations. In: AAAI Workshop: Planning for Hybrid Systems (2016)
22. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com (03 2009)
23. Nguyen, V., Obermeier, P., Son, T.C., Schaub, T., Yeoh, W.: Generalized target assignment and path finding using answer set programming. In: Proceedings of the 26th Int. Joint Conf. on Artificial Intelligence. p. 1216–1223. IJCAI'17, AAAI Press (2017)
24. Okumura, K., Machida, M., Défago, X., Tamura, Y.: Priority inheritance with backtracking for iterative multi-agent path finding. In: Proceedings of the Twenty-Eighth Int. Joint Conf. on Artificial Intelligence, IJCAI-19. pp. 535–542. Int. Joint Conf. on Artificial Intelligence Organization (7 2019). https://doi.org/10.24963/ijcai.2019/76
25. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
26. Silver, D.: Cooperative pathfinding. In: Proceedings of the First AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment. p. 117–122. AIIDE'05, AAAI Press (2005)
27. Strobel, V., Castelló Ferrer, E., Dorigo, M.: Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. In: Proceedings of the 17th Int. Conf. on Autonomous Agents and MultiAgent Systems. p. 541–549. AAMAS '18, Int. Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2018)
28. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. 2009 IEEE Int. Conf. on Robotics and Automation pp. 3613–3619 (2009)
29. Čáp, M., Vokřínek, J., Kleiner, A.: Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In: Proceedings of the Twenty-Fifth Int. Conf. on Int. Conf. on Automated Planning and Scheduling. p. 324–332. ICAPS'15, AAAI Press (2015)
30. Veloso, M., Biswas, J., Coltin, B., Rosenthal, S.: Cobots: Robust symbiotic autonomous mobile service robots. In: Proceedings of the 24th Int. Conf. on Artificial Intelligence. p. 4423–4429. IJCAI'15, AAAI Press (2015)
31. Wang, K.H., Botea, A.: Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. J. Artif. Intell. Res. (JAIR) **42**, 55–90 (09 2011). https://doi.org/10.1613/jair.3370
32. Wilde, B.D., Mors, A., Witteveen, C.: Push and rotate: cooperative multi-agent path planning. In: AAMAS (2013)
33. Wurman, P., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI Magazine **29**, 9–20 (03 2008)
34. Yu, J., LaValle, S.: Structure and intractability of optimal multi-robot path planning on graphs. In: AAAI (2013)