Using Distributed Ledger Technology for Command and Control and Decentralized Operations

David Last, Michael Atighetchi, Partha Pal, Edward Lu
Raytheon BBN Technologies
Cambridge, Massachusetts, USA
{david.last, michael.atighetchi, partha.pal, edward.lu}@raytheon.com

Ryan Toner
Air Force Research Laboratory
Rome, New York, USA
ryan.toner@us.af.mil

***Abstract* — The US military is developing the warfighting philosophy of Multi-Domain Command and Control (MDC2), which integrates land, sea, air, space, and cyberspace into a unified operation environment. MDC2 depends on the consistent sharing of operational plans and intelligence reports, which will be contested by adversary advances in communications-denying technology. Thus, the MDC2 system of the future must enable to development and dissemination of plans within the context of intermittent communications. We are developing a proof-of-concept MDC2 prototype to explore the requirements and constraints of this space. This system will be built on top of a distributed database; after evaluating the available options, we believe that Distributed Ledger Technology (DLT) is a strong candidate to meet the particular requirements of the MDC2 use case. Here, we investigate several DLT options and compare their capabilities to the MDC2 requirements, analyzing the design tradespace. We also examine several DLT alternatives and identify why they do not meet these requirements. We develop two initial prototype MDC2 systems, a baseline system based on an SQL-type relational database and one based on DLT. We run experiments to compare the performance of the two prototypes, and we discuss how these results relate to their suitability for MDC2. Finally, we outline the future path for this research in order to complete a full-functionality prototype MDC2 system.**

Keywords—Distributed Ledger Technology; blockchain; Command and Control; Multi-Domain Command and Control.

I. INTRODUCTION

With a view towards the battlefield of the future, Department of Defense (DoD) policy over the past half-decade has been moving towards the Multi-Domain Command and Control (MDC2) concept (also called Distributed Maritime Operations, Multi-Domain Operations, and All-Domain C2) [1] [2] [3]. MDC2 integrates the warfighting domains of land, sea, air, space, and cyberspace into a unified planning process under a single Joint Forces Commander. At the same time, adversaries are advancing their battlefield capabilities for jamming and otherwise hindering communications. Thus, the battlefield of the future will not resemble communications-permissive battlefields the DoD has enjoyed the last few decades. Based on these two trends, there is an increased need for front line units to share military plans and intelligence, but also the potential for significant barriers to doing so.

In order to address this situation, the DoD needs to develop a next-generation MDC2 system that allows commanders to share plans and orders across an entire theater of operations, but also empowers frontline units to collaboratively update plans in response to changing battlefield conditions. This MDC2 system must maintain a consistent view of the data among all parties (when in communication) and reconcile conflicts in the data (when re-connecting after a period of denied communication).

This MDC2 system should be built on top of a distributed database that can operate through intermittent communication and also reconcile divergent database copies that result from evolving data during network disconnection. Upon analysis of the system requirements, we believe that Distributed Ledger Technologies (DLT) are a promising option for this MDC2 system. The research presented in this paper describes an investigative study to evaluate the suitability of DLT for such a system.

This paper is organized as follows. In Section II, we present the MDC2 use case that motivates our research. Section III explores our reasons for selecting DLT as a solution for this problem, and Section IV details the different DLT implementations we considered. There are other solutions to this problem besides DLT; Section V presents some industry-standard alternatives and discusses their advantages and disadvantages compared to DLT for our use case. Section VI discusses our experiments with our DLT-based prototype, and Section VII recommends avenues for future research (both experimentation and development).

II. MULTI-DOMAIN COMMAND AND CONTROL USE CASE

Consider the following scenario: an Air Force base in a war zone is under threat of imminent attack. In an effort to protect his forces, the Air Commander divides his air units into small groups called Dispersed Units (DU); these DUs are organized into a hierarchy of Parent Dispersed Units (PDU). These DUs are then deployed to forward operating bases to geographically separate them (Figure 1).

Prior to dispersing the DUs, the Air Commander and his planning staff plan out the air war for the next 2 weeks. They assign different DUs to destroy or recon different targets, and they distribute these plans to the DUs. During the mission, the forward-deployed DUs are disconnected from the mission planners and other DUs due to geography, adversary jamming, cyber attack, etc. Additionally, the DUs discover that the battlefield is changing from its initial state as seen by the mission planners. Targets are in different locations than originally thought, new threats are discovered, etc. These changes invalidate the original mission plans, and necessitate updates/modifications to the plans in order to achieve mission success and deal with these new threats. Normally, the DUs would request plan updates from the mission planners, but they are now out of communication. Therefore, front line units in the DUs must be delegated authority to re-plan and re-task local units, and they must record these updates to the plan. When the DUs re-establish communications with the home base, these changes must be communicated to the Air Commander and mission planners. The original plans and the updated plans must be reconciled into a consistent, updated view of the plans so that all parties will be on the same page.

There are several critical requirements for a database to store these plans and enable decentralized operations. First, the database must provide ***consistent data*** between different network participants (to the extent possible). Second, when plans do diverge due to disconnected communications, different versions of the plans must be ***deconflicted*** once communications are re-established. Third, this database must be

 ISBN: 978-1-61208-940-9

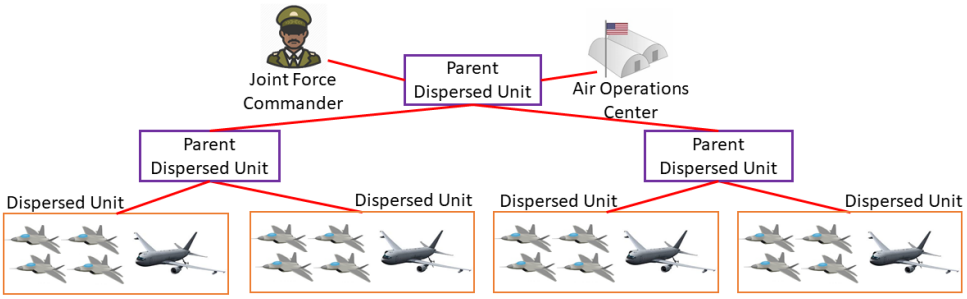

Figure 1. Organizational Hierarchy of Dispersed Units.

leaderless; there can be no single point of failure where disrupted communications means that network participants are unable to update or disseminate the plans. Fourth, the database must provide ***immutable***, ***auditable provenance***; any party examining the database should be able to examine the entire evolution history of the plans to see who made what changes, when, and why. This provenance history must be tamper-proof so that adversaries and bad actors cannot modify the history after the fact. Fifth, the database must also provide ***non-repudiation*** guarantees so that parties writing to the database can be held accountable for their updates.

Although we have presented a Department of Defense use case here, such a system would have wide applications. Consider a wilderness search and rescue scenario. Drones and human volunteers are collaborating to search a National Park for a lost hiker. This is a remote, austere environment where the searchers (human and drone) use relatively low-powered radios for communication, and communications will often be interrupted due to distance and geography. The search is led by a single Search and Rescue Coordinator, and on-the-ground searchers are divided into groups (DUs) and assigned to different search areas. The different groups need to collaboratively re-plan and re-assign roles during the search in response to changing intelligence and environmental conditions, and they need to communicate their search results back to the Coordinator after they complete their search pattern.

This type of command and control system has many other applications. It could be used to coordinate Border Patrol units or to manage drug enforcement operations. This system would be useful for collaborative planning between groups of autonomous drones working towards a common goal, such as mapping a remote area. It could even be used to manage logistics systems like the United States Postal Service (USPS), UPS, FedEx, or DHL.

III. WHY DLT?

The Command and Control system described here requires a distributed database that allows multiple parties to modify the same data. It must allow updates to the data in disconnected network partitions, and it must automatically reconcile data conflicts taking into account mission context in order to determine which version of the data is authoritative. In this research, we explore Distributed Ledger Technology (DLT), e.g., blockchain, as a potential solution for this system. DLT/blockchain was designed as a leaderless, distributed database that can tolerate network disconnection, and it has properties that are attractive for distributed Command and Control systems.

A. How DLT\blockchain works

In a DLT/blockchain network, all network participants maintain a local copy of the distributed database (ledger). All data operations on this database (create/update/delete) are encoded as Transactions and submitted to a pool of Proposed Transactions. Certain network participants (called "miners" in Bitcoin networks) select a set of Proposed Transactions, check them for "correctness" (according to a set of rules based on the characteristics of the system), bundle them into a Block, append that Block to the end of a chain of Blocks (hence blockchain), and advertise the new Block to other network participants. Other network participants verify the correctness of the Transactions and then accept the new Block. The Block contains a cryptographic hash of its own contents and the contents of the previous Block in the chain, which makes the Block ***immutable*** (any tampering with the Block will invalidate the hash, making the tampering instantly detectable).

In some cases, two different miners will create and advertise two different next Blocks. This results in different network participants having different versions of the blockchain; this is called a blockchain ***fork***. Different blockchain implementations have methods for avoiding forks, and they also have methods for determining which fork will be accepted as the authoritative blockchain; the other fork will be discarded.

A key aspect of DLT/blockchain is the consensus algorithm. If it is easy for miners to generate and propose new Blocks, then it will be easy for bad actors to manipulate the system. The consensus algorithm makes it difficult to generate a valid Block, but easy to check the validity of the Block. The consensus algorithm is also used to limit the blockchain mining speed, which reduces the frequency of blockchain forks and makes it more difficult for bad actors to manipulate. There are different types of consensus algorithms. Proof of Work requires a large amount of compute power to solve a difficult, but easily verifiable, math problem to generate a new Block; this algorithm is used in the Bitcoin network [4]. Proof of Stake requires network participants to "stake" a certain amount of owned cryptocurrency in order to become validators (same role as miners) who are randomly selected to create the next Block; this algorithm is used in the Ethereum network [5]. Proof of Authority is similar to Proof of Stake except that validators are chosen to create the next Block with probability of being chosen being proportional to the validator's reputation (based on its past behavior in the network) [6].

B. Design tradeoffs

Distributed Ledger Technology has a number of advantages related to the requirements of the distributed Command and Control use case. First, DLT/blockchain guarantees eventual consistency of the data (when all nodes have the same copy of the blockchain). Second, blockchain is designed for leaderless management so that there is no single point of

 ISBN: 978-1-61208-940-9

failure that can bring down the network. Although the command hierarchy in Figure 1 seems to indicate the Joint Forces Commander as a single leader for the C2 system, once authorities to modify plans are conditionally delegated to frontline units, this situation more closely resembles a leaderless network. Third, blockchain provides the required immutable provenance auditing and non-repudiation. Finally, DLT also enables smart contracts, which can be used for the delegation of authority described previously.

However, there are also drawbacks to using blockchain for distributed databases. Because the system is leaderless, it relies on the consensus algorithm to reach a consistent data state (rather than a single leader dictating the data state). The consensus algorithm is necessarily slower and more complex than a single master database. This has significant implications for the latency and throughput of Write transactions for a blockchain-based distributed database.

Based on these design tradeoffs for blockchain-based distributed databases, we evaluate that blockchain is best suited for data that evolves slowly and can tolerate latency. Therefore, high-volume data like sensor information or video streams are not well-suited for blockchain. The Command and Control data for collaborative mission planning are a good fit for blockchain-based systems; however, C2 applications with real-time requirements may not be suited for blockchain databases.

This paper outlines an investigatory effort that explores the feasibility of using blockchain for Command and Control collaborative planning.

IV. DLT FRAMEWORKS

As part of this research, we investigated a number of different open-source DLT implementations to serve as the basis for a distributed, collaborative Command and Control system. The final Command and Control system has the following requirements, so the basis DLT implementation should support these:

- Flexible roles/leaderless operation for network nodes (no single point of failure)
- Network partitions must support continued operation to some degree
- Configurable access control/authorities management
- Support for blockchain forking and reconciliation
- Not resource intensive (operation on mobile platforms with constrained communications)
- Permissioned network – all participants are known and authorized

With these requirements in mind, we evaluated to applicability of several different DLT implementations.

Option 1: **Hyperledger Fabric**

Hyperledger Fabric is an open-source project developed under the Linux Foundation. In contrast to many blockchain implementations (like the Bitcoin network), Fabric is permissioned, rather than permissionless. All network participants are known, and only certain network participants are authorized to add Transactions to the blockchain. Network participants are authorized with X.509 security certificates issued by a certificate authority. Network participants are divided into *organizations*, which share a single distributed ledger. Organizations can be grouped into a *consortium*, which allows participants from different organizations to access the distributed ledger of the other organizations. Hyperledger Fabric encodes access control policies into chaincode, which is used to govern database read/write operations. Because Hyperledger Fabric uses X.509 certificates and defined network validators, it does not require a resource-intensive consensus algorithm like Proof of Work [7].

Option 2: **R3 Corda**

R3 Corda is a distributed ledger that was developed for the financial services sector [8]. It uses the Unspent Transaction Output (UTXO) model (similar to Bitcoin) for managing data assets. R3 Corda does not use Proof of Work as a consensus algorithm; rather, it defines the concept of a Notary, where a single Notary must control all data assets consumed by a Transaction. If a single Notary does not control all the data assets, then control most be transferred before the proposed Transaction can be executed. The need for Notaries poses a significant problem for Command and Control in disconnected environments. A disconnected Command and Control system will require multiple ownership of data assets since we do not know a priori which node will need to modify which assets.

Option 3: **Algorand**

Algorand is a blockchain-based digital currency like Bitcoin that was created in 2017 by MIT professor Silvio Micali [9] to address some of the shortcomings of Bitcoin. Algorand uses Proof of Stake as a consensus algorithm. The Algorand network is permissionless, so any party can join as a network node. Additionally, Algorand supports only one class of user; all nodes in the system have the same authority level (although weighted by their stake in the system). This disallows the designation of "trusted" parties with varying levels of authority, which is necessary for the Command and Control delegation of authority. Because of these reasons, Algorand is not suitable for a Command and Control system.

There are many more blockchain implementations in this technology space, but most of them share basic characteristics with these three systems. Based on our analysis, we selected Hyperledger Fabric as the basis for our Command and Control system. Hyperledger Fabric supports some of the requirements for the Command and Control system (leaderless operation, configurable authorities, operation of network partitions, permissioned network), but other capabilities will need to be built around it as part of our prototype (blockchain forking and reconciliation). One key point in favor of Hyperledger Fabric is that **since it does not use a resource-intensive consensus algorithm like Proof of Work**, it does not suffer from well-known energy consumption concerns about cryptocurrency implementations like Bitcoin.

V. NON-DLT ALTERNATIVES

Blockchain is a relatively new approach to distributed databases. There are a number of more traditional distributed database solutions that should be considered as direct competitors to a blockchain-based system; any blockchain system should be evaluated against these other solutions.

SQL (Structured Query Language)–type databases are relational databases organized into tables with columns and rows. In a standard configuration, a single SQL-type database serves Read and Write requests from multiple Clients. SQL databases like MySQL, PostgreSQL, and Microsoft SQL Server do not natively support a distributed database configuration, but they can be configured into a single master/multiple replica configuration to support distributed Read operations but not distributed Write operations [10].

Git is a popular open-source Distributed Version Control System (VCS) that was developed in 2005 to manage soft-

 ISBN: 978-1-61208-940-9

ware development projects with multiple contributors. Git users can download local copies of a master database, make updates to the data, and then push the updates to be merged with the master database. Any local updates that do not conflict with the master database are merged automatically, but local updates that conflict with updates to the master database are flagged to the human user for manual merging. Git records a full history of who made what changes to the database, at what time [11]. Git works as a distributed database, but it is not leaderless; it relies on one database node serving as the authoritative master node.

The **Interplanetary File System (IPFS)** was designed as a Peer-to-peer file sharing system that works as a distributed database. Users of the database create files locally and IPFS divides the files into chunks, generates a cryptographically hashed Content ID (CID) for each chunk, and advertises the CIDs to other users in the network. If other users wish to download the file, IPFS queries the network for the location of the data associated with the relevant CIDs and downloads the chunks. That user then becomes a secondary provider for those CIDs until they are deleted. When new versions of a file are added to IPFS, they are stored using new CIDs; old versions of the file cannot be tampered with or erased (unless all providers of the CID delete their local copy). Within IPFS each file exists as an independent entity; there is no concept of conflicting versions of the same data and no merge/reconcile functionality [12].

VI. Experimentation

A. Prototypes

The goal of this research and the initial experiments it encompasses is to evaluate the feasibility of using Distributed Ledger Technology as a distributed database for a Command and Control system, as opposed to other distributed database solutions. To fulfill this goal, we built two prototypes for experimentation: one representing the state of the practice (built on top of PostgreSQL, a relational database), and one representing the state of the possible (built on top of Hyperledger Fabric). These two prototype networks each contain three nodes that function as a distributed database (Figure 2). PostgreSQL is not natively a distributed database, so that prototype is set up with a single master and two replicated copies. In this configuration, clients can only write to the master node, and these write operations are propagated to the replicated copies.

It is important to note that these two prototypes do not provide the same functionality. Because the PostgreSQL prototype is not a true distributed database, it has no need for a consensus algorithm, because only one node (the master node) is the arbiter of the correct data state; this also represents a single point of failure. Because there is no need for consensus in the PostgreSQL database, the message exchange between nodes will necessarily be much more complex in the Hyperledger Fabric prototype than the PostgreSQL prototype. Therefore, we fully expect that the PostgreSQL prototype will outperform the Hyperledger Fabric prototype in terms of throughput and latency when processing Write operations. The main advantage of Hyperledger Fabric over a more traditional database is that it does not contain a single point of failure, and that a partition of the network can continue operation even when disconnected from the rest of the network. The PostgreSQL prototype does not support either of these capabilities. The following experiments demonstrate how much of a performance downgrade Hyperledger

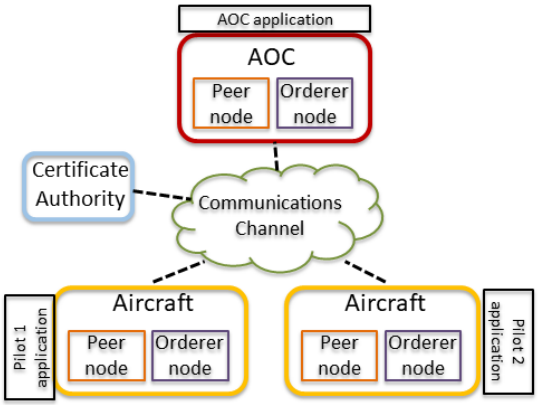

Figure 2. Hyperledger-based MDC2 Initial Prototype.

Fabric suffers as opposed to a more traditional approach in order to analyze the tradeoffs between basic performance metrics and the special functionality that blockchain provides. These experiments also provide indications as to which use cases are best suited for a blockchain-based approach.

B. Experimentation

We ran several experiments to compare the performance of the Hyperledger Fabric- and PostgreSQL-based prototypes. The different nodes of the network were run in Docker containers on an Ubuntu Linux VM. We used the Pumba tool [13] to simulate bandwidth degradation and disconnection between different network nodes. For these experiments, we use Write Transactions that write representations of Link 16 J2.2 messages to the distributed database (Link 16 J2.2 messages are Air Force self-position reports for military aircraft) [14].

C. Experiment 0 – Hyperledger Fabric parameters

Our first experiment was an initial exercise of the Hyperledger Fabric prototype to explore its capabilities and experiment with major configuration parameters to identify the optimal configuration for our use case. We experimented with two independent variables: Batch Timeout and Traffic Density.

One of the major configuration parameters for Hyperledger Fabric is Batch Timeout. This value, expressed in fractions of a second, instructs the Hyperledger Fabric nodes that once they receive a Write Transaction, how long they should wait for additional Transactions before bundling all available Transactions into a new Block to be added to the blockchain. If this parameter is set to a low (short) value, then new Transactions will be bundled into Blocks almost as soon as they are received. This may improve the Transaction throughput, but an increased number of Blocks being processed by the consensus algorithm can increase Transaction latency. On the other hand, if this parameter is set to a high (long) value, it can increase Transaction throughput (because there are fewer Blocks, there is less network overhead per Transaction), but because there are fewer Blocks, it can also (counterintuitively) decrease the average Transaction latency. In this experiment, we vary the value of Batch Timeout to find the optimal setting for our use case.

For the Batch Timeout experiment, we use the 3-node network configuration shown in Figure 2. Two Clients commit Link 16 J2.2 Write Transactions at a frequency of 500 milliseconds for each Client. These Clients commit Transactions

 ISBN: 978-1-61208-940-9

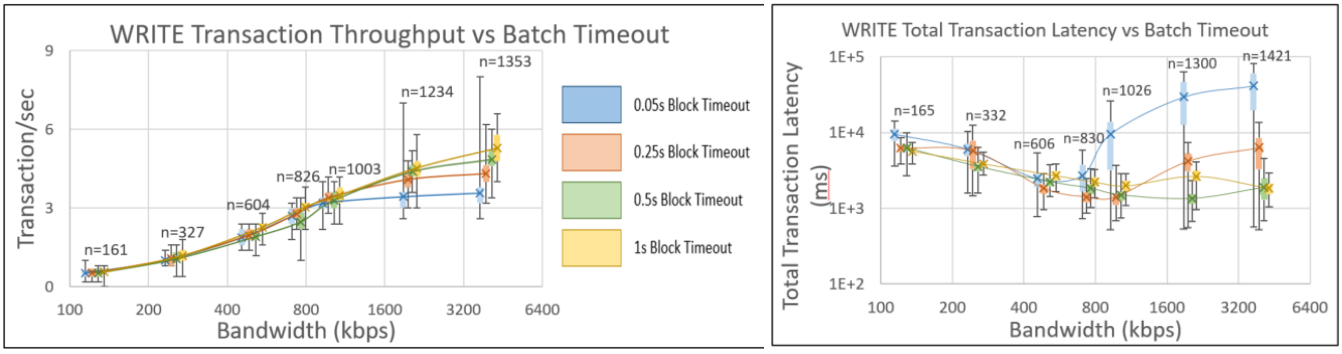

Figure 3. Experiment 0 - Hyperledger Fabric prototype Write Transaction throughput and latency vs. Batch Timeout.

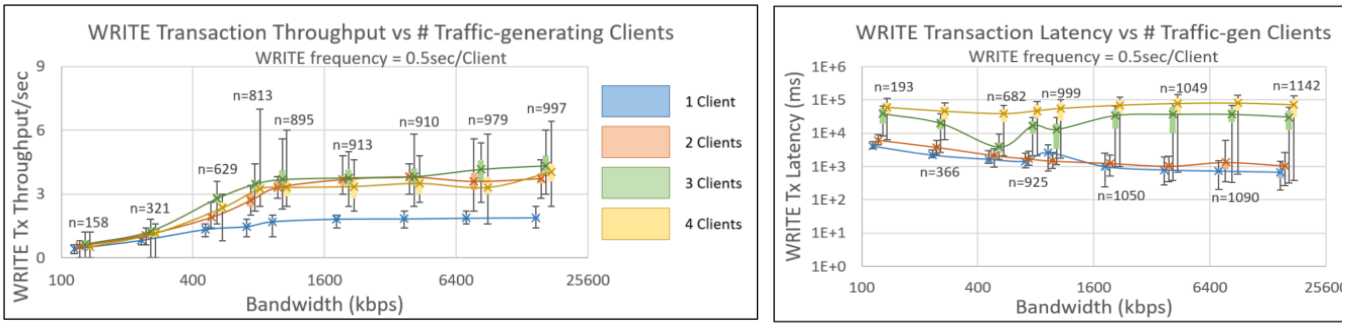

Figure 4. Experiment 0 - Hyperledger Fabric prototype Write Transaction throughput and latency vs. Traffic Density.

for a period of 300 seconds. We vary the Batch Timeout parameter between 0.05s, 0.25s, 0.5s, and 1s, and we also vary the bandwidth of the Hyperledger Fabric network to simulate degraded communications. We measure the Transaction throughput and latency to determine the optimal Batch Timeout for Hyperledger Fabric. The experimental results are shown in Figure 3; we determine that the optimal Batch Timeout parameter is 0.5 seconds.

We also run an experiment to determine the maximum traffic density the Hyperledger Fabric network can handle before it begins to affect performance. We use 1-4 Clients, which each submit Link 16 J2.2 Write Transactions to the network at a frequency of 500 milliseconds, and we run the experiment for 300 seconds. We use a Batch Timeout vale of 0.5 seconds based on the previous experiment. We measure the Transaction throughput and latency, and the results are shown in Figure 4. We also ran these experiments for 5 and 6 Clients, but those results are similar to the results for 4 Clients. Based on the experimental results, we determine that the amount of traffic generated by 2 or 3 Clients represents the best tradeoff between throughput and latency, depending on the situation.

D. Experiment 1

Following the Hyperledger Fabric parameter tuning in Experiment 0, we run the first experiment that compares the Hyperledger Fabric and PostgreSQL prototypes head-to-head. In this experiment, 2 Clients write Link 16 J2.2 messages to the distributed database at a frequency of 300 milliseconds per client; the database Transactions are approved (according to the prototype's approval mechanism) and then propagated to all nodes in the network. The experiment lasts for a period of 300 seconds. We vary the bandwidth available to the networks to simulate degraded communications, and we measure the throughput and latency of the networks.

Based on the difference in complexity of the Transaction approval mechanism between the two prototypes, we expect the PostgreSQL prototype to outperform Hyperledger Fabric in both throughput and latency; however, we wish to see if the difference between these metrics is sufficiently low to justify the benefits of Hyperledger Fabric in our Command and Control use case. We show the experimental results in Figure 5.

E. Experiment 3

We also run an initial experiment to compare the performance of the two prototype systems in a disconnected communications scenario. In this experiment, we measure how long it takes to merge new database Transactions into a database node that has not yet received them. At the beginning of this experiment, a set of Clients write 500 Link 16 J2.2 Transactions to the database; these Transactions are propagated to all nodes in the network. Then, one of the database nodes is partitioned from the rest of the network (in PostgreSQL, this is one of the replicated nodes). The Clients write an additional 500 J2.2 Transactions to the main network; the partitioned node does not receive these Transactions. We then reconnect the partitioned node to the network, and the network automatically pushes the new Transactions to the reconnected node. We measure how long it takes for the reconnected node to be brought fully into sync with the rest of the distributed database nodes. The results are shown in Figure 6.

F. Future Experiments

In the future, we plan to run additional experiments to further evaluate the performance of the Hyperledger Fabric prototype against the PostgreSQL prototype.

Experiment 2 – Hardware and Network Requirements: This experiment will use the same procedure as Experiment 1. We will measure the disk storage required at each node, the processing power for 1 Write Transaction, and the network overhead for submitting 1 Write Transaction and propagating it to all distributed nodes.

Experiment 4 – Dynamic Data Merging: This experiment will use the same procedure as Experiment 3, except that the

 ISBN: 978-1-61208-940-9

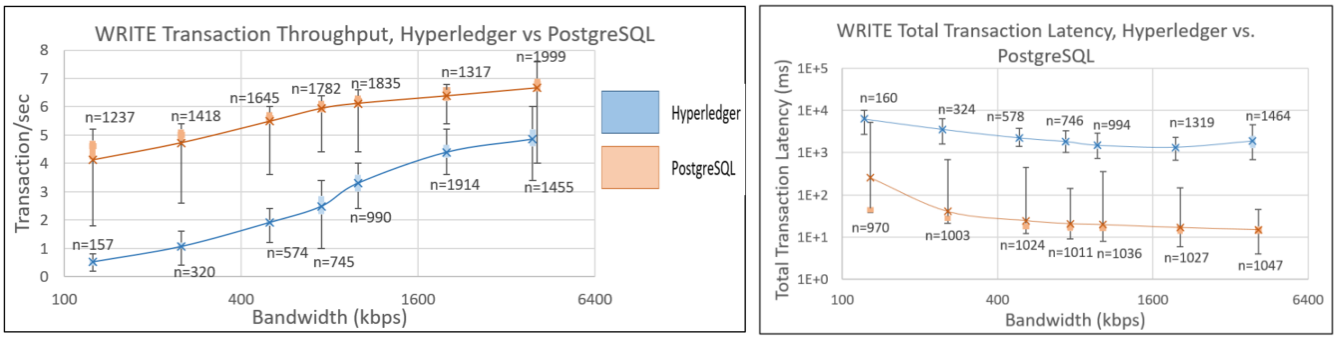

Figure 5. Experiment 1 - Hyperledger Fabric vs. PostgreSQL Prototypes Write Transaction throughput and latency.

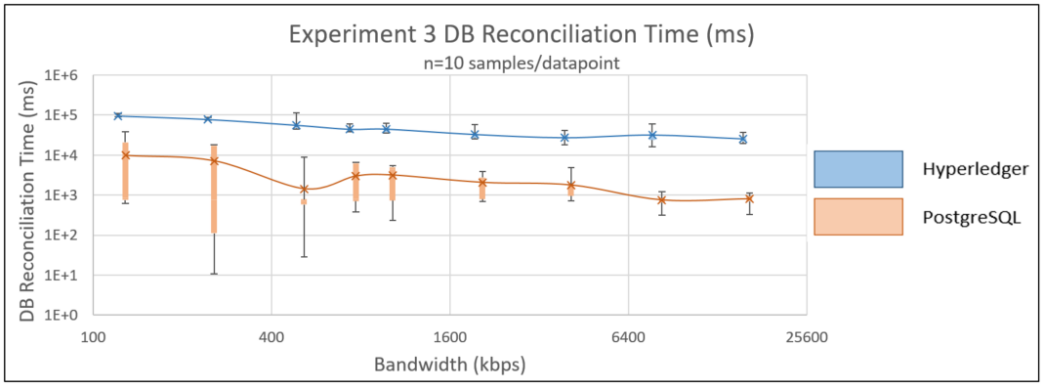

Figure 6. Experiment 3 - Hyperledger Fabric vs. PostgreSQL prototypes Write Transaction throughput and latency.

Clients will continue to write new Transactions to the database during the Merge period. We will measure how long it takes for the partitioned node to come back into synchronization with the rest of the nodes, and what is the effect of the increased network traffic due to the new Write Transactions.

G. Discussion

The experiments discussed here show that in initial performance tests, a blockchain-based Command and Control system performs worse in terms of latency and throughput than a simpler, non-collaborative SQL-type relational database. The key distinction between the two is the consensus algorithm; it enables leaderless operation and disconnected communication tolerance, but it introduces significant complexity in validating and committing Write Transactions. Therefore, the key questions for evaluating the suitability of blockchain are:

- Based on its performance constraints, what type of data is blockchain best suited for?
- Do the benefits of blockchain (security, leaderless operation to tolerate degraded communications) outweigh its performance penalties for our specific use case?

The Experiment 0 results indicate that a blockchain-based system is best-suited for low Write volume data with a moderate tolerance for Write latency. Therefore, blockchain is most applicable to high value data that does not change frequently (like military campaign plans), but not high-volume, low latency data (like real-time control signals, sensor data, or video streams). Experiments 1 and 3 bolster this determination. Over the course of a multi-day military engagement, plans will probably be added to the database at less than 3-5 Transactions per second, and the Command and Control system can tolerate a 1-10 second latency on the dissemination of these plans to frontline units (Figure 5). Additionally, isolated frontline units re-establishing communication with the main group can tolerate 20-100 seconds to download Command and Control updates (Figure 6).

VII. Recommended Future Work

The research discussed in this paper was performed as a study to answer the question, “Is it feasible to use blockchain as a Command and Control distributed database?” Since this research has answered this question in the affirmative, the next step in this research is to build a full prototype. This prototype will incorporate several innovations beyond the initial study.

The first innovation will be blockchain branching and merging. Most current blockchain implementations address blockchain forks by requiring a majority of network nodes to write to the blockchain, thus explicitly preventing forks (Hyperledger Fabric uses this approach), or they resolve forks by determining one fork branch to be authoritative and discarding the other branches (Bitcoin’s blockchain implementation uses this approach). Neither of these approaches are sufficient for the Command and Control scenario: requiring a majority of nodes to write new Transactions prevents minority partitions from writing new data, and discarding a blockchain fork (which represents the collaborative planning of a minority partition) invalidates previous planning and decision-making, throwing the entire Command and Control system into chaos. Therefore, the full prototype needs new functionality to allow blockchain forks in minority partitions, as well as merging these blockchain forks within an understanding of the context of the larger mission.

The second innovation for the full prototype will be the implementation of a conditional authority calculus. In a full communications environment, planning decisions should be made by the highest-ranking authority and disseminated to lower-ranking units. If communications are disconnected, these lower-ranking units must be authorized to make these planning decisions. However, if there are no constraints on which units can make which planning decisions, this can lead to an explosion of blockchain branches that will be very com-

 ISBN: 978-1-61208-940-9

plex to maintain and merge. Our conditional authority calculus will use a dynamic ruleset that is evaluated in the context of the mission environment to determine which parties are allowed to make which planning decisions at a specific point in time. This will constrain the complexity of the planning process and the resultant blockchain merges.

As we develop this full prototype, we will also pursue opportunities to deploy it during Department of Defense field exercises in order to evaluate its performance in operational scenarios and begin building acceptance within the user community.

VIII. Conclusion

In recent years, the DoD has been moving towards the Multi-Domain Command and Control philosophy as the most effective way to integrate warfighting domains. However, adversary advances in communications-denying technologies jeopardize the ubiquitous communications needed to realize MDC2. Therefore, the DoD needs an advanced MDC2 system that enables collaborative planning and information sharing in the presence of constrained, intermittent communications. Based on our investigation, we believe that Distributed Ledger Technology is a strong candidate for such a system that supports the communication requirements of the MDC2 scenario. In this paper, we investigate different DLT implementations and evaluate them against the MDC2 scenarios; we identify Hyperledger Fabric as meeting the key requirements for MDC2. We built two different MDC2 prototypes: one based on standard distributed database technology, and one based on Hyperledger Fabric. We ran a number of experiments to evaluate the performance of the two systems, and to evaluate whether Hyperledger's performance is sufficient for an MDC2 system. Our experimental results are encouraging, so we chart a path forward to build a production-grade DLT-based MDC2 system that can operate in modern, communications-denied environments.

Statements/Disclaimers

Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). Case # AFRL 2022-0076. This effort is sponsored by the Air Force Research Laboratory (AFRL).

The views expressed are those of the authors and do not reflect the official guidance or position of the United States Government, the Department of Defense or of the United States Air Force.

Statement from DoD: The appearance of external hyperlinks does not constitute endorsement by the United States Department of Defense (DoD) of the linked websites, or the information, products, or services contained therein. The DoD does not exercise any editorial, security, or other control over the information you may find at these locations.

References

[1] US Air Force, "Air Force future operating concept: A view of the Air Force in 2035." Washington, DC: Government Printing Office, 2015. Accessed: Apr. 20, 2022. [Online]. Available: https://www.af.mil/Portals/1/images/airpower/AFFOC.pdf

[2] J. M. Richardson, "A design for maintaining maritime superiority, Version 2.0." Naval College War Review, Dec. 17, 2018. Accessed: Apr. 20, 2022. [Online]. Available: https://media.defense.gov/2020/May/18/2002301999/-1/-1/1/DESIGN_2.0.PDF

[3] US Army Training and Doctrine Command, "The U.S Army in multi-domain operations 2028." Training and Doctrine Command, Ft. Eustis, VA, Dec. 06, 2018. Accessed: Apr. 20, 2022. [Online]. Available: https://adminpubs.tradoc.army.mil/pamphlets/TP525-3-1.pdf

[4] "Proof of work," *bitcoin.it*. https://en.bitcoin.it/wiki/Proof_of_work (accessed Apr. 20, 2022).

[5] "Proof of stake (POS)," *ethereum.org*, Dec. 09, 2021. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (accessed Apr. 20, 2022).

[6] "Proof of authority explained," *binance.com*, Dec. 09, 2020. https://academy.binance.com/en/articles/proof-of-authority-explained (accessed Apr. 20, 2022).

[7] E. Androulaki *et al.*, "Hyperledger Fabric: A distributed operating system for permissioned blockchains," 2018.

[8] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: an introduction," *R3 CEV August*, vol. 1, p. 15, Aug. 2016.

[9] E. Auditore, "Algorand: origins," *Algorand: Origins*. https://community.algorand.org/blog/algorand-origins/ (accessed Apr. 20, 2022).

[10] M. Kamaruzzaman, "Top 10 databases to use in 2021," *towardsdatascience.com*, Jan. 20, 2021. https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba (accessed Apr. 20, 2022).

[11] D. Spinellis, "Git," *IEEE Softw.*, vol. 29, no. 3, pp. 100–101, Jun. 2012.

[12] "Interplanetary File System (IPFS)," *IPFS*. https://ipfs.io/ (accessed Apr. 20, 2022).

[13] A. Ledenev, "Pumba: chaos testing tool for Docker (Github)," *Pumba: chaos testing tool for Docker (Github)*. https://github.com/alexei-led/pumba (accessed Apr. 20, 2022).

[14] Air Land Sea Application Center, "Introduction to Tactical Digital Information Link J and quick reference guide (TADIL J)." Jun. 2000. Accessed: Apr. 20, 2022. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a404334.pdf

 ISBN: 978-1-61208-940-9