Toward Time Synchronization in Delay Tolerant Network based Solar System Internetworking

Alan Hylton
NASA Goddard Space Flight Center
alan.g.hylton@nasa.gov

Natalie Tsuei
American University
nt9913a@american.edu

Mark Ronnenberg
Indiana University Bloomington
maronnen@iu.edu

Jihun Hwang
Purdue University
hwang102@purdue.edu

Brendan Mallery
Tufts University
Brendan.mallery@tufts.edu

Jonathan Quartin
University of Colorado Boulder
jonathan.quartin@colorado.edu

Colin Levaunt and Jeremy Quail
University of Vermont
{colin.giles, jeremy.quail}@uvm.edu

Justin Curry
University at Albany, SUNY
jmcurry@albany.edu

Abstract— **The expanding presence in space will place an increased dependency on networked communications – a scalable communications infrastructure; that is, the Solar System Internet (SSI). Upcoming developments towards a SSI include NASA's upcoming LunaNet, or lunar Internet, which provides multi-hop multi-path communications using Delay Tolerant Networking (DTN). DTN has been an active area of research and development, particularly in routing, security, and optimization. DTNs are marked by mobility, disconnection, and a wide variance of latencies (propagation and processing delays). In this paper, we outline progress towards a theory of time synchronization across such a network.**

An underlying assumption of DTN is that the network is time synchronized already, rather than synchronization being provided as a service. While this is necessary for schedule-based routing, which is necessarily prevalent in DTNs, it is so deeply ingrained as to be built into the primary unit of data in DTNs – the bundle. Indeed, a bundle's creation timestamp and its time to live (called the *lifetime*) are based on time, and there are special recommendations for systems that lack accurate clocks. The assumption of time synchronization makes sense when limiting considerations to smaller-scale and more traditional space communication. However, just as end-to-end connectivity cannot be guaranteed in DTNs, neither can access to a reference or authoritative clock. In this more general case, it might be necessary to synchronize over time-varying meshes, and perhaps even to consider relativistic effects. Moreover, by imposing synchronization restrictions in order to sustain a network, the effectiveness of the network to achieve scalability will be necessarily muted.

To work towards a time synchronization theory for DTNs, we build upon past successes in modeling DTNs using time-varying graphs and sheaves. This includes error and limitation estimation, which allows one to define domains over which schedule-based routing is possible, up to some threshold sensitivity. Despite the theoretical nature of these results, the approaches taken are also algorithmic, and hence lend themselves to practical implementations. The paper concludes with comparisons of the various methods along with suggestions for future work.

TABLE OF CONTENTS

1. INTRODUCTION

Communication is key to the success of most missions, whether terrestrial or space-based. NASA's current approach to the upcoming Solar System Internet (SSI) is to develop and implement a Delay/Disruption Tolerant Network (DTN), which is a suite of communication protocols that are not susceptible to intermittent connectivity, high latency, and variable delays [1]. DTN is a well-studied topic and there are at least three directly relevant RFCs—4838, 5050, and 9171—defining necessary specifications and required architectures. Other RFCs—5326, 6255, and 7242—cover further implementation details and extensions, as well as security considerations—RFC 5327, 6257, and 9172. Despite this body of work, DTN has not yet been fully integrated into NASA's Near Space Network (NSN) for human missions or interplanetary (IPN)[1] missions due to several identified issues, such as scalability and practicality. Another prominent issue, which is the topic of this paper, is global clock synchronization. In what follows, we introduce and motivate the problem of clock synchronization, survey existing work, and then outline several novel fundamental structures to support time synchronization in DTNs.

Clock synchronization is a long-standing problem in computer networks and distributed systems. Even though it is not a strict requirement for widely-used network protocols on Earth, such as TCP/IP, the vast majority of popular or advanced programs have precise synchronization of clocks across the network as a necessary condition for their accurate performance [2]. For instance, clock synchronization errors could hinder performance of navigation systems [3], data transmissions and signal modulations [4], and even commu-

[1] For the sake of simplicity, the term 'IPN Internet' may be used interchangeably with 'SSI' throughout this paper.

nication programs such as messaging applications in our daily life. In a general sense, navigation and distributed applications are limited by the network's ability to synchronize. Because the vast majority of space missions are time sensitive and schedule-driven (prior to and during their execution), these limitations apply to DTNs as well. Numerous mission-critical DTN applications such as communication/navigation (LunaNet) [5], routing [6], [7], congestion control [8], and network simulation [9] are known to function accurately only when there is a global clock synchronization achieved across the entire network.

It is well-known that DTNs, as specified in the above RFCs, must maintain global time synchronization in order to guarantee its network-wide accurate and efficient performance [1], [10], [11]. Moreover, time synchronization is assumed to be provided as an external service rather than as a service provided by the network [12]. We should note that achieving time synchronization across a DTN is believed to be tractable, as demonstrated in [13], [14], and [15], especially in small-scale settings. However scaling these solutions is believed to be exponential in the number of nodes in the network, and is especially vulnerable to non-deterministic events, thus making synchronization an ongoing challenge [10]. In particular, asymmetric data rate and delays, invalid or unsuccessful (re)transmission, and radioactive and electromagnetic interference are all what makes global time synchronization over SSI consistently problematic [16]. As a final motivating example, the DTN definition of time to live (TTL) is time based (as opposed to hop based), yet in RFC 4838 expired data are not necessarily deleted, in part due to time sync difficulty.

Organization of the Paper

The primary purpose of this paper is to exhibit potential mathematical and algorithmic approaches to achieve global clock synchronization over DTNs that scale well and are hence suitable for large-scale or time-sensitive missions. The next section is dedicated to giving an overview of the current architecture of DTN (RFCs 4838 and 9171) while explaining the reasons that global clock synchronization in DTN is necessary, yet unresolved. This paper also provides an introduction to computer clocks and clock synchronizations for terrestrial internet, as well as a brief summary of previous works regarding time synchronization on DTNs. Finally, the heart of this paper is dedicated to introducing several mathematical concepts and structures, such as dominating sets and cellular sheaves, that can be used to address clock synchronization. The paper concludes by discussing future directions for research.

2. NETWORK ARCHITECTURE

Internet vs. DTN-based Solar System Internet

The (terrestrial) Internet is based on the Internet Protocol suite, which is systematically modeled as a layer/stack of protocols and is largely broken down into five layers: Application, Transport, Network, Link, and Physical layer. Briefly, the application layer is responsible for supporting network applications (e.g. HTTP, SMTP, DNS, etc.), the transport layer is for process-to-process data transfer and encapsulation of messages from application layer, the network layer is for delivering segments from one node to another, the link layer is for transferring packets from one node to its neighbors, and the physical layer is for moving bits from one node to another connected node [17]. One also simply refers to Internet Protocol suite as TCP/IP, as TCP and IP are the most widely used principle protocols today for the transport layer and network layer, respectively. The ability to use TCP is predicated on end-to-end connectivity and "reasonable" round trip times, among other things. Having these assumptions met enables time synchronization as we know it: the Network Time Protocol (NTP).

DTNs are protected against intermittent connectivity, high latency and delays as they use *store-and-forward* methods to move packets from one place to another. This separates (and really, generalizes) DTNs from the classical Internet. Moreover, just as TCP/IP may rest on top of Ethernet or 802.11, so too is DTN an overlay, which may rest on a multitude of underlying protocol stacks connecting space and ground assets. These links are typically point-to-point scheduled contacts, and may have multiple characteristics in terms of latency, variance of latency, asymmetry of rates, and even protocol stacks. At its core, DTN must aggregate disparate space links into a network, where neither the individual links nor their aggregation can be assumed to support NTP.

To aggregate these links, the data structure used in DTNs is called a *bundle*, which may be of effectively any size.[2] Thus DTN adds a layer called the *bundle layer* between the application and transport layers. Currently, the most well-known protocol that resides in bundle layers is the *bundle protocol (BP)*, which was initially defined and specified in RFC 5050. TCP or UDP can be used for the transport layer, but it is also compatible with Licklider transmission protocol, or LTP (RFC 5326), which is a convergence layer protocol specifically developed for space communication networks that resides in the transport layer; see [18] for examples and further detail.

Bundles and Their Lifespans

Once an application residing in the application layer of a DTN node passes a message down to the bundle layer, the message then gets "encapsulated" in a bundle by getting a block of information (which we will refer to as the *bundle header* in this paper) prepended to the message. This encapsulation can be done in a way similar to how it is done on the Internet, for example, as how an application-layer message is passed down and becomes a transport-layer segment by having a transport-layer header cascaded in front of the application-layer message. Every bundle header comes with a *creation timestamp* (or "timestamp") and *lifespan*, measured in (milli)seconds from a given epoch. The timestamp denotes the creation time of the bundle and the lifespan gives the time at which the bundle (message) is no longer valid and hence needs to be purged. Here, one may immediately observe that inconsistency of time over nodes in a DTN could cause misinterpretation of times written in timestamps and the lifespan fields of bundle headers. There are additional reasons that accurate timestamps and lifespans are necessary for every bundle. The timestamp of a bundle along with endpoint identifiers (EIDs) allows that bundle to be uniquely identified. This is needed for the assembly of bundles or bundle fragments, but can be done only when basic global time synchronization is provided [6]. Also, time synchronization is required for having accurate sleep schedule protocols, which can be used to save energy by setting infrequently used nodes into sleep mode [19], [14].

A simulation conducted by NASA Glenn Research Center

[2] For bundles to be made compatible with lower-layer protocols, the so-called *convergence layers* must be used.

(GRC) with Surrey Satellite Technology Ltd (SSTL) and Universal Space Network (USN) at Alaska gives experimental evidence that global time synchronization is a necessary condition. Bundles were configured and sent from NASA GRC to Guildford, England, and then sent from Guildford to USN. Even if all three clocks were synchronized at the beginning, clock drifts had occurred during the course of the experiment, which resulted in discrepancies in timestamps, and hence to unplanned rejection and expiration of bundles. The problems were resolved after clocks were all synchronized again. This demonstrates the importance of global time synchronization for the successful operation of DTNs in practice; see [20] and [11] for more details about this experiment.

Characteristics of DTN

By construction, DTNs are often said to be analogous to the postal service due to their store-and-forward property [1]. BY comparison, the Internet is considered more akin to an airline in functionality, as generally described in introductory computer networking textbooks such as [17]. The TCP/IP-based Internet fully serves its purposes only when continuous end-to-end connections and synchronous communications between nodes are guaranteed. For instance, TCP establishes a connection between two nodes based on its three-way handshaking process, which is susceptible to latencies and packet loss by its design. However, DTN's underlying assumptions of prevailing delays and disruptions make the nature of their communications *asynchronous* or *partially synchronous* [21]. To be precise, we say communication is asynchronous if there is no upper bound on message delivery time, and is partially synchronous if either the upper bound of message delivery time exists but not known to the nodes or it is known but will apply only after a certain amount of time that no processes know (i.e. the message sent from a node will be delivered to the other node eventually, but no one knows when it is going to be exactly) [22]. This concept shall be revisited in later sections of this paper.

Moreover, TCP/IP provides reliable data transfer but does not always guarantee delivery. It may discard packets for congestion or flow control purposes, e.g. if no free buffers are available in the queue of a router, or if another end-point is unavailable or unreachable. DTNs prevent such packet losses by utilizing persistent storage along with network-layer acknowledgments (called *custody transfer*); instead, DTN purges expired bundles based on their lifespan and timestamp to preempt storage overflows or congestion in bundle layers [10]. Studying methods of managing buffers and storage for bundle layers efficiently (such as [23]) is another current topic of research. Consequently, DTNs are more suitable than TCP/IP-based networks for communications under extreme settings, such as: military [24], satellite [25], or space (both near and deep) [26] where disruptions can be present at any times or occur unexpectedly. See [27] for more examples and their details.

3. PRELIMINARIES

Computer Clock Model and Time Synchronization

The formal scientific definition of a second is the time it takes Caesium-133 (^{133}Cs) atom to make 9,192,631,770 cycles of radiation [28], and a clock made with ^{133}Cs atoms is called an atomic clock. It is, however, not practical to distribute atomic clocks to the general public due to their price and size, so the vast majority of computers today have an embedded oscillator (circuit) called a *hardware clock* designed to keep track of accurate time. A *software clock* then calculates a second based on the number of oscillations that its hardware clock made. More specifically, if a hardware clock is supposed to oscillate F times per second, then its software clock increments the second by 1 once its hardware clock oscillates F times (or increments by $1/F$ if it oscillates 1 time). With this said, software clocks are totally dependent on the behavior of their hardware clocks, so a malfunction of a hardware clock could lead its software clock to return incorrect values; for example, a temperature increase in a machine could cause the hardware clock to oscillate faster than it is supposed to, causing a clock skew.

A software clock can be modeled as a function returning a numerical value that happens to be (or is supposed to be) asymptotically close to the real-time [29]. To model this, let $C_p(t)$ be the value of the software clock and $F_p(t)$ be the frequency of the hardware clock at time (e.g. UTC) t on machine p. Let F be the ideal frequency, i.e. $C_p(t) = t$ for all t if $F_p(t) = F$ for all t. For simplicity, suppose that the clocks were synchronized when $t = 0$, i.e. $C_p(0) = 0$ for all p. Define clock drift rate as the difference in time from a perfect clock, and ρ be the maximum clock drift rate (usually given by manufacturers). Consequently, for all t and p [29],

$$\left|\frac{F_p(t) - F}{F}\right| \leq \rho \iff 1 - \rho \leq \frac{F_p(t)}{F} \leq 1 + \rho. \qquad (1)$$

Assume for mathematical convenience that $F_p \colon [0, t] \to \mathbb{R}$ is bounded and continuous almost everywhere (a.e.) for all $t \in \mathbb{R}$. Between time 0 and t, the hardware clock with frequency F_p would have oscillated $\int_0^t F_p(\tau)\, d\tau$ times. Note that F_p is integrable due to Lebesgue integrability criterion. By definition,

$$C_p(t) - C_p(0) = \frac{1}{F} \int_0^t F_p(\tau)\, d\tau\,.$$

By the fundamental theorem of calculus, C_p is differentiable (hence continuous) on $(0, t)$, and

$$\frac{dC_p(t)}{dt} = \frac{F_p(t)}{F}. \qquad (2)$$

Indeed, the ideal scenario would be where $C_p(t) = t$ and $dC_p(t)/dt = 1$ for all t and p. Combining Equation (1) and (2) gives us the following [30]

$$1 - \rho \leq \frac{F_p(t)}{F} = \frac{dC_p(t)}{dt} \leq 1 + \rho \qquad \forall t, p. \qquad (3)$$

The smaller $\rho \in (0, 1)$ is the more accurate clock we have as it is closer to the most ideal scenario mentioned above. Since (definite) integration of positive continuous function preserves inequalities (this can be proven using the mean value theorem),

$$(1 - \rho)t + C_p(0) \leq C_p(t) \leq (1 + \rho)t + C_p(0) \qquad (4)$$

Most of the computers today have $\rho \approx 10^{-6}$ [31] so, for the purpose of mathematical modeling, $C_p(t)$ can be approximately computed as a first-order system

$$C_p(t) = \alpha_p t + \beta_p \qquad (5)$$

for some constants α_p and β_p.

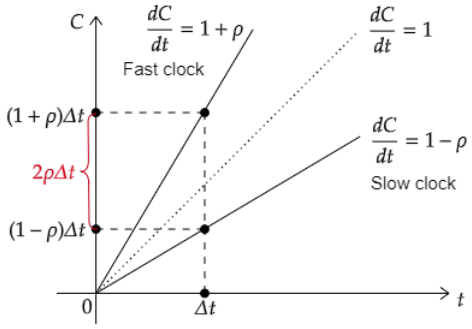


Figure 1. Depicts the range of error of a software clock

Even if two clocks were synchronized to the same correct time (UTC, for example), they can be as much as $2\rho\Delta t$ apart from each other at time Δt, as described in Figure 2. Although ρ is considered a small number, $2\rho\Delta t$ may still be larger than the requirement of a system or network. Denote the time we need to re-synchronize clocks to as t_{resync}. Let π be a constant that denotes our desired clock precision, i.e. the maximum time deviation between any two machines at any time [29] is

$$|C_p(t) - C_q(t)| \leq \pi \quad \forall t, p, q.$$

Since the maximum possible time deviation between two clocks at $t = t_{\text{resync}}$ is $2\rho t_{\text{resync}}$,

$$|C_p(t) - C_q(t)| \leq 2\rho t_{\text{resync}} \leq \pi \iff t_{\text{resync}} \leq \pi/(2\rho),$$

meaning that we need to synchronize the clocks at least once in $\pi/(2\rho)$ unit time to maintain the clock precision π. See Figure 2 for its plot. The discrete case where F_p and/or C_p are discrete can be analyzed analogously; for example, a discrete clock can be modeled as a continuous clock with an error up to $1/2$ [29].

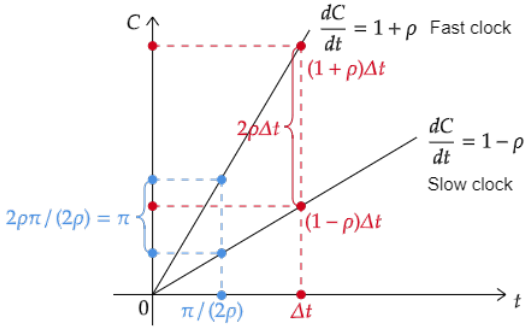


Figure 2. Depicts desired precision, and hence, a resynchronization window

Network Time Protocol (NTP)

The most natural approach to time synchronization would be to constantly reach out to the designated time server (preferably running an atomic clock or equivalent) for the accurate time, or its approximation. Vendors of mainstream operating systems typically have their own time server, which users' machines synchronize to. For example, Microsoft provides Windows Time service (`w32tm` in Command Prompt) that periodically synchronizes the user's software clock by querying the time server `time.windows.com`.

The Network Time Protocol (NTP) is a time synchronization protocol developed by Mills [32], [33]; more precisely, it was first implemented by Mills, Mamakos, and Petry and formally documented in RFC 958 in 1985, which is called NTPv0 today. It evolved to NTPv1 (RFC 1059) a few years later into a form where it is compatible with User Datagram Protocol (UDP) and the Internet Protocol (IP), and the most recent version of NTP currently (as of October 14, 2022) is NTPv4 defined in RFC 5905 and 7822.

At a high level, NTP uses a version of an intersection algorithm by Marzullo [34] that estimates the round-trip network delay time by comparing the client's clock to the time server's clock [33]. In more detail, suppose a client with computer p sends a synchronization request message at t_1, in the client's time, and the message arrives at the server at t_2, in the server's time. The server then sends a reply message at t_3, in the server's time, which arrives to the client at t_4, in the client's time. We can first observe that it took $(t_3 - t_2)$ seconds for the server to process the synchronization request. From the client's perspective, the round-trip time of the message was $(t_4 - t_1)$, as it departed the client at t_1 and returned back at t_4; see Figure 3 for a visual representation. Hence, it can be said that it took approximately $\delta := (t_4 - t_1) - (t_3 - t_2)$ seconds for a message to travel across the network.

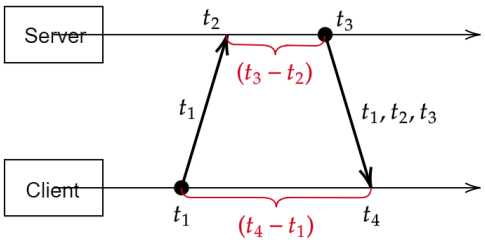


Figure 3. Depicts message travel times

Assuming there was no major network disruption, we may assume that the travel time (propagation delay) from client to server is the same as from server to client. Then, by the time the reply message from the server at t_3 reaches the client, which was t_4 from the client's perspective, the server's clock would be approximately $t_3 + \delta/2$. So the clock skew θ is

$$\theta \approx \left(t_3 + \frac{\delta}{2}\right) - t_4 = \frac{(t_2 - t_1) + (t_3 - t_4)}{2},$$

and the client then can adjust their clock accordingly based on the θ above and Eq. 5. The client also may make multiple synchronization requests to enhance the accuracy of their clock as necessary.

Thanks to its accuracy, fault-tolerance, and scalability, most of the popular time synchronization services today such as Windows Time service were all implemented using NTP as its core functionality [35]. Its variations have also been used for several space missions such as time and frequency transfer between satellites or stations [36]. However, NTP is presumed to be not practical for DTNs in interplanetary settings. As mentioned, NTP approximates its clock skew δ based on the assumption that propagation delays will remain the same for both directions of communications (client to server and server to client). DTN's nature of asymmetric data rates renders such assumptions invalid. Moreover, NTP is highly dependent on the fact that an end-to-end connection between a machine and a clock server is continuous and stable with minimal disruptions, which DTNs cannot guarantee. When there is a disconnection between a node and the time server, the node must stand by until the connection is re-established, which could take more than a conceivable or

tolerable amount of time. By the time when the node is able to communicate with the time server, the error may have already exceeded the acceptable range of error. By the same reasoning, any NTP-like centralized protocols by themselves are also speculated to be unsuitable for large-scale DTNs [13]. There are also architectural reasons; in particular, NTP is hierarchical in nature, dividing intermediary nodes from the source to the client into strata, where the error accumulates with additional strata. These hierarchies might not exist in a DTN, which features temporal network structures that might be more mesh-based.

Fault Tolerant Consensus: Byzantine General Problem

Recall that a set of computers is referred to as a distributed system when it operates cooperatively as one coherent entity. To successfully accomplish this, there must be a high level of coordination occurring between devices, and an obvious prerequisite for this condition is ensuring that every device is 'on the same page' by achieving *consensus* on certain values and data structures at times, as necessary. Consensus algorithms are fault-tolerant in the sense that the system must be able to reach consensus even in the presence of a certain number of faulty nodes. There are numerous different ways a node can fail and cause an error. Paxos [37], one of the most commonly used consensus algorithms today, is said to be crash fault-tolerant as it is capable of solving consensus problems as long as more than half of the nodes are responsive *on time*; more specifically, it is crash f-fault tolerant as long as there are $2f + 1$ or more nodes in the system. However, Paxos can be ineffective if a system is not fully synchronous, in an environment where connections are frequently intermittent, or when processes/nodes can exhibit Byzantine faults. A process or node is said to have *Byzantine fault* if it behaves in an arbitrary or malicious manner. There are other types of fault models: crash faults, described above); omission faults, where messages are being lost; and timing faults, where response time is not within an acceptable range. By definition, the Byzantine fault model includes omission faults and crash faults. Consequently, Paxos or Paxos-like consensus protocols may not be sufficient for DTN for SSI, as nodes can act arbitrarily and even unreliably. We remark that by studying consensus proofs, we believe that initial expectations and limitations of time synchronization of DTNs can be more carefully formed.

A few Byzantine faulty nodes in a system, even if they take less than 50% of the system, can cause catastrophic outcomes. Consider a fictional scenario where three generals are seizing a city with their armies, waiting to establish a common plan of action – attack or retreat – which is crucial for them as they must all attack the city simultaneously to win the battle. If one general is a traitor who delivers contradicting messages to the other generals (e.g. sending 'attack' to one general and 'retreat' to another), then a general may receive two conflicting messages and be unable to identify which is from the traitor, and thus fail to reach consensus.

The preceding scenario is called the *Byzantine General Problem*. It was first resolved with mathematical proof by Lamport, Shostak, and Pease in [38] and [39]. In general, they proved that a system must have more than $3f$ nodes to be up to f Byzantine fault sufficient. To see that $3f + 1$ is indeed enough, let N be some sufficient number of nodes that allows the system to tolerate f Byzantine faulty nodes. There would be at least $N - f$ non-faulty nodes and the system must be designed to be able to reach consensus with $N - f$ nodes, i.e. $N - f$ messages, for the liveness of the protocol. In the worst case scenario, since there are f faulty nodes, f out of $N - f$ messages might have originated from faulty nodes, making only $(N - f) - f = N - 2f$ messages being reliable at minimum. Since reliable messages must outnumber unreliable messages,

$$f \leq (N - 2f) + 1 \implies N \geq 3f + 1,$$

and hence $3f + 1$ is a sufficient and minimum requirement. A similar proof with the same logic can be found in [40] as well. One can also directly prove that a system with $N \leq 3f$ nodes will always fail. Figure 4 describes the case where $N = 3$ and $f = 1$, but this argument can be extended to $N = 3f$ case for all f.

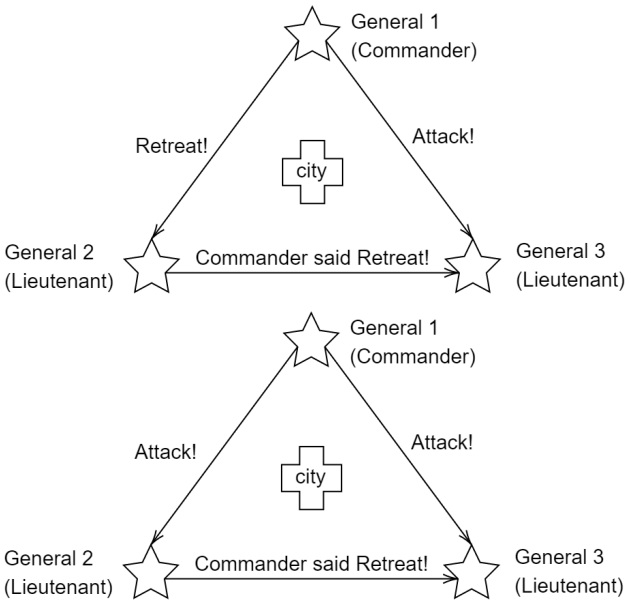


Figure 4. A toy example demonstrating that the Byzantine General Problem is unsolvable when $N \leq 3f$. Note that General 3 for both cases cannot make a final decision as to who is the bad actor, i.e. in the top scenario, General 2 is faulty, but in the bottom one, General 1 is faulty.

Lamport et al [38] also constructed a consensus algorithm called *Byzantine fault tolerance (BFT) algorithm* for reaching agreement with the presence of f Byzantine nodes. Once a node acting as the leader sends out its message to every other node, then every node that just received the message broadcasts its received message to every other node as if it has become a new leader of the system. However, such a recursive (flooding) method would not scale well. It would end up creating

$$(N-1)(N-2)\cdots(N-(f+1)) = O(N^{f+1})$$

messages in total. This communication complexity can be too high to deal with for networks with low-end devices or ones with small throughput/bandwidth. Notice that this protocol must be running over a synchronous (or at least partially synchronous) system [41], as its consistency and correctness are only guaranteed with an underlying assumption that every message will be delivered from one node to another before timeout. If nodes crash or messages get lost in transit, an asynchronous system will be unable to detect this, thus this protocol is unsuitable for DTNs.

Byzantine Fault Tolerance over Asynchronous Settings

The main downsides of Lamport et al's original solution to the Byzantine General Problem in [38] was its computational expense and incompatibility with asynchronous settings. Liskov and Castro in [40] introduced *Practical Byzantine Fault Tolerance (PBFT) protocol* which achieves consensus in a distributed system with Byzantine failures and asynchronous communication. Similar to the original results, it could tolerate up to f faulty nodes given $3f + 1$ nodes in total. PBFT is often regarded as a combination of BFT and Paxos due to its construction, though PBFT is not as decentralized as Paxos and may not work well in a completely asynchronous system.

The main idea of PBFT is *state machine replication*, which is a commonly used method for building fault tolerant protocols (see [42] for introduction). The PBFT protocol divides the nodes in a system into three groups: *client*, *primary*, and *backups*. In order to tolerate up to f faulty nodes, similar to Lamport et al's original protocol, PBFT requires (at least) $3f + 1$ *replica* nodes – one node being the primary (leader) and $3f$ nodes being the backups. The PBFT protocol is then divided into five stages: *request*, *pre-prepare*, *prepare*, *commit*, and *reply*. Each phase does roughly the following (for more technical details, such as regarding how those packets are constructed, see the original paper [40]):

(1) **Request**: Client sends a message m to a replica node requesting a change of a value/state. The replica node receiving the message will now be the leader of this consensus process, which we call the primary node.
(2) **Pre-Prepare**: Primary generates a sequence number (often referred to as timestamp) N for the request m, then broadcasts pre-prepare messages to all the backups (remaining replicas). A pre-prepare message is a digitally signed packet (signed by the primary with the signature σ_p) that consists of the current view of the message m, sequence number, and the digest (summary) of m.
(3) **Prepare**: Once a backup i receives a pre-prepare message from the primary, it verifies the signature, makes sure that the sequence number N has not been used before (or is larger than other sequence numbers used before), then confirms that they are in the same view as the primary for m and the digest of m matches with the primary's. Then each backup broadcasts prepare messages signed with their own digital signature σ_i to every other replicas. Each prepare message from i contains i and the same information as the pre-prepare message as what i received and verified.
(4) **Commit**: If a replica receives $2f + 1$ pre-prepare messages and $> 2f$ prepare messages, then that replica is said to be in the status of 'prepared certificate' and broadcasts the commit message to every other replica. The content/format of commit messages is the same as prepare messages.
(5) **Reply**: Once a node receives more than $2f + 1$ commit messages, it goes into status of 'commit certificate' if it was also in the 'prepared certificate' status. It then replies back to the client with the result that they agreed upon.

Digital signatures can be replaced with message authentication codes (MACs). Liskov and Castro [40] claim that using a MAC might be more practical than using a digital signature because MACs can be computed and verified faster than digital signatures. They compared a digital signature scheme based on RSA-1024 (with MD5 used for computing the message digest) with a MAC scheme using MD5[3]. However, given that we commonly use SHA-256 or SHA-512 for hashing messages and RSA-2048 for digital signatures today, we expect that this comparison may not be as accurate as it used to be, not to mention that MD5 has been proven to be insecure. This topic will be discussed in detail in a later section, but, until then, we shall assume that this protocol was implemented using MACs instead of digital signatures.

[3]The authors did not specifically state it in their paper ([40]), but it appears to be an HMAC.

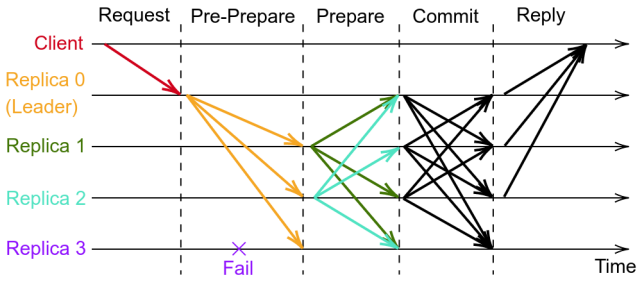


Figure 5. A visual description of the five stages of PBFT consensus protocol over four backups with one faulty node.

We now calculate the communication complexity of PBFT with respect to the number of nodes n. The request stage uses only one message, the pre-prepare stage up to $n-1$ messages, prepare goes up to $(n-1)^2$, and commit and reply use up to $n(n-1)$ and n messages respectively if verification results are true. Hence, PBFT in total requires

$$1 + (n-1) + (n-1)^2 + n(n-1) + n = O(n^2)$$

messages, which is a vast improvement over the previous or original BFT consensus algorithms. Also, notice that every node in the system replies back to the client. This makes the PBFT a decentralized protocol despite the existence of the primary node (leader) and also immune to Byzantine failure of the leader node, yet it only creates $O(n)$ messages and does not contribute much negatively to the overall performance. Another benefit of PBFT is the round complexity. Recall that Lamport et al's original solution required $f + 1$ rounds of communications, but PBFT requires only three rounds of communications (among replicas). PBFT is considered to provide low latency and be energy efficient [43], making it attractive to apply in various settings that are not tolerant to long delays such as blockchain technologies [44].

To discuss the correctness of PBFT in-depth, we need to look at the formal definition of correctness first. We largely separate the correctness condition into two criteria: *safety* and *liveness*, where safety is informally treated as 'nothing bad will happen' and liveness is treated as 'something good will happen eventually' [45]. More precisely, safety is when every pair of non-faulty nodes agree upon the same value or order of messages, and liveness is the guarantee that the protocol will always terminate with a correct output given a correct input. Formal definitions of safety and liveness from a mathematical logic perspective can be found in [46]. From the consensus viewpoint, one can interpret the safety property as partial correctness and the liveness property as the total correctness (correct for every input) [47]; thus, strictly requiring both safety and liveness properties to hold could be an unnecessary requirement in practice.

Combining safety, liveness, and practicality (or scalability) altogether into one protocol is not an easy problem to solve. In fact, Fischer, Lynch, and Paterson [48] proved that reaching consensus over an asynchronous system is not possible even in the case where only one node is faulty, Byzantine or

not. This theorem is commonly referred as the *FLP (Fischer-Lynch-Paterson) impossibility theorem*. This result may seem to contradict the existence of asynchronous consensus protocols. However, the FLP impossibility theorem only gives a negative result to the existence of a deterministic consensus protocol where both safety (correctness) and liveness (termination) hold for any input. For example, Byzantine fault tolerance is unachievable in an asynchronous system as the deterministic termination property cannot hold. However, for the cases where the termination property is defined weakly (i.e., the case where a faulty transmitter can result in a non-termination), Byzantine fault tolerance can be achieved with $3f + 1$ or more nodes [49].

Both Paxos and PBFT sacrificed liveness for safety, but PBFT can also satisfy the liveness property in a partially synchronous system. It is important to note that safety is always guaranteed in PBFT protocol thanks to the two-step verification system—prepared certificate and commit certificate. However, PBFT may not guarantee forward progress without eventual synchrony. This means that an algorithm must either have a timeout function or have synchronous communication after a certain time period [50].

4. RELATED WORKS

Double-Pairwise Time Protocol

The Double-Pairwise Time Protocol (DTP), constructed by Ye and Cheng in [13], is an extended version of NTP modified to serve its purpose for networks with intermittent connections and asymmetric transmission rates. As mentioned, NTP requires the end-to-end connections between a client and the time server to be established continuously, and any link disruption could halt the entire process, and such disruption could continue for a long period of time. DTP is designed to combat this issue by sending a pair of requests instead of a single request at a time.

Ye and Cheng [13] make an analogy of describing the NTP as a process of consecutive *ping-pong* between two ends (e.g. see Figure 3), where the client sends a request message (*ping*) and the server replies with its current time (*pong*). Then DTP can be metaphorically described as a sequence of *pingping-pongpong*'s. The client sends a synchronization request message and then sends another one after a certain amount of time (*pingping*). By doing so, we increase the chance of both request messages being closely located in the buffer of the time server. This makes both messages more likely to undergo the same queuing delay, hence makes it easier for us to approximate the network propagation delay. Once the client receives replies to both requests from the server (*pongpong*), it repeats the same process again multiple times. The protocol then chooses the process with the minimum difference between the two time differences – the difference between two requests and between two replies – and use that as a reference for estimating clock skews. Instead of fitting the client's clock closely to the reference clock as NTP does (fixing the slope of the linear clock model (Eq. (5)) to be $\alpha_p = 1$), DTP makes a linear approximation between the reference clock and the client's current clock.

As authors demonstrated [13], DTP is proven to be more accurate than NTP on both types of networks: one with intermittent connections and asymmetric transmissions (e.g. DTNs) and the other without them (e.g. terrestrial internet). However, DTP is still a NTP-like protocol assuming the existence of the central time server or a reference node(s), which SSI may not have or may be too distant to reach as needed. Also, experiments were allegedly conducted under the assumption where the time server (reference node) replies almost immediately upon its recipient of the request message. Although this is a reasonable assumption that does not invalidate the correctness of the protocol, this may not be able to accurately capture the case of SSI where there might be multiple decentralized time servers working collaboratively to keep accurate clocks. Further details will follow shortly.

Consensus-based Clock Synchronization Protocols

Clock synchronization problem over a network can be reformulated as the problem of reaching consensus on time or its approximation in the presence of faulty nodes. Such approaches have been popular for studying clock synchronizations on unstable decentralized networks such as mobile ad hoc networks (MANETs). MANETs are distributed wireless networks characterized with random movements of nodes and dynamic network topology, and use multihop routing techniques for routing between two nodes [51]. MANETs are considered unreliable as nodes can only communicate with their neighbors within a certain distance, and hence an end-to-end connection between two arbitrarily chosen nodes may not be possible. As a result, MANETs are subject to difficulties ranging from routing to packet losses, especially when nodes are highly mobile and suffer limited bandwidth, buffer, and computational power [51]. DTN can be viewed as a variation of MANET that improves the reliability of packet transfer under non-real-time traffic by using store-and-forward technique [52]. MANETs are often studied under the assumption that every pair of nodes is strongly connected, but such an assumption can be unrealistic for DTNs.

Byzantine fault is the most general failure model, and hence is the most popular (yet the most difficult) approach of modeling and solving the consensus problems. Lamport and Melliar-Smith [53] extend the solution to the Byzantine General problem in [38] to fit the frame of clock synchronization problem. In their protocols, a node sends a copy of its clock function (Eq. 5) to the other nodes instead of its clock values as static numbers. Among three protocols in [53], two of them can achieve f-fault tolerance with $3f + 1$ nodes, whereas the third protocol, which involves the use of digital signature as the second algorithm in [38], can with $2f + 1$ nodes. Lamport and Melliar-Smith conjectured that a system without authentication would require at least $3f + 1$ nodes to achieve clock synchronization in the presence of f Byzantine nodes. This conjecture was then proven to be true by Dolev, Halpern, and Strong in [54]. More precisely, [54] proves that without authentication, the network must have $3f+1$ or more nodes and its connectivity must be $2f + 1$ or greater. With authentication, the $3f + 1$ condition can be relaxed, but the network must be $(f + 1)$-connected.

All three protocols in [53] are created with underlying assumptions that every non-faulty node is capable of accurately computing the difference between its clock and another non-faulty node's clock on its own, which may not be a realistic assumption for DTNs. Moreover, besides the high computational and communication complexity that is already entailed, and beyond the statistical evidence that they will reach consensus and converge to a certain value, there is no guarantee that the time they agree upon is actually the accurate time [32]. However, the emphasis must be put on the fact that a system requires a time that every node agrees upon. With an absence of reliable connection to the central time server, or the absence of connection itself, a system is more or less 'isolated' from the 'outside world' because it

does not have access to a clock that it can use as a reference.

Fault Tolerant Average Algorithm

The simplest way to reach consensus on a certain data would be to have all nodes reset their values to the average of everyone's value. This method has been used since 19th century, then theoretically formulated by Poincare and Einstein as a form of Einstein synchronization in relativity theory. Sasabe and Takine [55] adopted this logic as an asynchronous clock synchronization protocol where two nodes exchange their clock values with each other upon contact/connection and adjust their clock as the average of two values. It was anticipated that such a protocol would be efficient in terms of time complexity and energy consumption yet inaccurate, but the experimental results showed that it could achieve the accuracy at the level of NTP for small-scale DTNs. However, its accuracy varies heavily by the number of connections established during the course of operation, and this gets even worse when it comes to large-scale DTNs. For DTNs where connections are rather dynamic or uncertain and limited in terms of the number of neighbors a node can contact with and the frequency of connection establishments, such asynchronously averaging protocols can be highly inaccurate, especially at the presence of multiple faulty clocks.

For the case of synchronized networks, a fault-tolerant version of such protocols can be constructed fairly easily, as was done by Welch and Lynch in [30]. Suppose that there are N nodes in the network and f of them are expected to be faulty. Each node receives the clock value from all the other nodes in the system. After this step, every node should have a list of N clock values that consists of its own clock value and the values received from $N-1$ nodes. Then, every node computes the list of relative clock offsets by subtracting every element in its list with its clock value. For fault-tolerance, each node discards the smallest f elements and the largest f elements from the list, then takes the average of $N-2f$ remaining elements in the list and uses that as a reference for adjusting its clock.

We believe that there could be a way to apply this approach that Welch and Lynch proposed [30] to the protocol by Sasabe and Takine [55]. However, the protocol by Welch and Lynch makes an underlying assumption that faulty nodes are considered 'faulty' because their clock will be either faster or slower by a conceivable amount of time. Such assumptions are indeed plausible and practical. However, networks under harsh conditions such as in SSIs can make even non-faulty nodes' clocks deviated from each other by a large amount, and nodes demonstrating Byzantine failure can broadcast different clock values to each node to prevent the non-faulty nodes' clocks from being synchronized; for example, a Byzantine faulty node could adaptively broadcast a clock value to a node close to that node's clock value; by doing so, each node would believe that the Byzantine faulty node is not faulty yet other non-faulty nodes are faulty. Moreover, this still requires every node to communicate with each other directly and synchronously, which is not a desirable requirement for large-scale DTNs.

Distributed Asynchronous Clock Synchronization

Distributed asynchronous clock synchronization (DCS) protocol was first introduced by Choi et al. in [14] and [15] as a method of achieving clock synchronization asynchronously over a (terrestrial) DTN. The overall logic behind DCS is that for networks with intermittent connectivity where end-to-end connection between two nodes are not always guaranteed, it is more efficient and more accurate to exchange tables with relative clock offset/skew values. By doing so, we can obtain information about the clocks of other nodes even without getting in a direct communication with all them synchronously.

We summarize the version of DCS protocol introduced in [14] briefly below. As we did before, we represent the clock value of node p at time t as $C_p(t)$. Denote the clock offset between node p and q as C_{pq}; for all node p, there is a clock table in node p that stores every C_{pq} value for all q. Upon contact, node p and q exchanges the values $C_p(t)$ and $C_q(t)$ with each other. Then p computes the observed relative clock offset as

$$C^o_{pq} = C_q(t) - C_p(t)$$

and q computes C^o_{qp} on its own as well. Then node p updates its clock table by setting

$$C_{pk} = C^o_{pq} + C_{qk}$$

for all k, unless the information about k that p just received from q is outdated compared to the information p had already. Then p finally computes its own clock offset $\tilde{C}_p$ as the weighted average of all C_{pr}'s with weights w_{pr} corresponding to the age of information (which [14] models it as $w_{pr}(\tau) = \lambda^\tau$ where $\lambda \in (0, 1]$ and τ represents the time since the last update). Finally, p adjusts its clock as $C_p \leftarrow C_p + \tilde{C}_p$. The node q that was in contact with q also adjusts its clock following the same process.

However, as authors mentioned, DCS was initially designed for terrestrial DTNs, which are far smaller than the SSI. DCS updates a clock based on the clock information gathered from all other nodes. This may be inefficient or inaccurate if direct end-to-end connections between some nodes are rarely established, and this was proven during one of the experiments in [15]. More formally speaking, it is difficult for a node to gather global information about the (entire) network both accurately and timely at all instances. This will be discussed further in later sections. Requiring a node to remember and update the clock table storing information of every other nodes may cause inefficient use of data storage and create huge additional data traffics globally across the network taking up the network capacity. In spite of these probable downsides, its capability as a practical synchronization protocol has been well-established by the authors through a series of mathematical proofs and numerical experiments. DCS still is a promising protocol for clock synchronization in a relatively small scaled network or a local area network of a large-scaled networks.

5. LOCAL SYNCHRONIZATION

Our proposed approach to studying clock synchronization over a DTN-based SSI is analogous to the history of the Internet. The expansion of the Internet started off as a development of ARPANET, by connecting computers within a close proximity as a form of local area networks, which later got interconnected with each other. A similar direction of development is recurring to the SSI. A DTN also consists of (local) regions called DTN regions interconnected with each other via gateways and each region with a unique region ID [56]. Every clock synchronization protocol, distributed and centralized, has clear advantages and disadvantages of their own. SSI may be large enough compared to the Internet today to have disadvantages accumulated up to the point

where it can be easily compensated. We hence suggest an adaptation of the cluster-based synchronization method where a cluster corresponds to a DTN region and a gateway is a DTN gateway.

In cluster-based synchronization protocols, gateways are actively communicating with each other and function as a de facto time server for the nodes in their respective clusters. See [57] for one of the recent developments of cluster-based clock synchronization technique. For SSI, however, gateways can also be incomparably distant from each other just as a pair of two nodes can be. A piece of information delivered from one node/gateway to another may not be arriving on time or be inaccurate by the time it gets delivered, making a node only able to accurately represent or store 'local' information gathered within a close proximity than the 'global picture' of the network. See [58] for more details and mathematical treatments. This suggests that clock synchronization over local network is not the same problem as achieving global clock synchronization across the entire network. This section will focus on the local clock synchronization problem, and global clock synchronization will be discussed in the next two sections.

Throughout this paper, we shall assume that propagation delays are always known. That is, prior to transmission, a node can accurately calculate the amount of travel time of a packet to reach its immediate destination. We consider this as a fair assumption to have, as it be done fairly accurately using (relativistic) Doppler effects. For example, see [59]. Suppose that a node A wants to send its time $C_A(t)$ to its neighbor B. As soon as a connection between the two nodes gets established, A transmits $C_A(t) + D_{A\to B}$ to B where $D_{A\to B}$ is the propagation delay from A to B computed by A using Doppler effect.

Our construction of a clock synchronization protocol over a DTN region involves both a centralized/hierarchical approach and a consensus-based approach. The purpose of such an intercession is to maintain all desirable features of time synchronization—fault-tolerance, efficiency, accuracy, scalability, etc.—at certain levels, yet not sacrificing any of them. To be specific, having every node run a fault-tolerant consensus protocol such as PBFT as a time synchronization process may result in excessively high message complexity and delay. More notably, it can be difficult for a node to identify which clock values are from faulty nodes (and hence needs to be discarded), purely based on the numerical clock values it received. Recall from the previous section that discarding received clock values based on the magnitude of time difference can be unhelpful.

This implies that having some 'reference' that gives information about within what range a correct clock would approximately be in can alleviate the difficulty of this problem. To do this, we create a set of nodes entitled the *time synchronization hub*, or *hub*, that serves as a de facto distributed time server within a region. Nodes in the hub constantly synchronize their clocks and maintain them consistent. Since a hub plays a significant role, it has to be reachable from every node in the region within a reasonable time and distance. We hence select a hub as a k-hop connected dominating set of the region, where every node in the region is within distance k from a hub and every node in the hub is connected. A hub would be a connected subgraph of a region, we hence anticipate that it would be relatively compact enough to operate time synchronization protocols designed for small-scaled or terrestrial networks (for practical purposes, if needed, a region can be divided into subregions and have each subregion elect their own hub). Due to the absence of global time server, we use a distributed protocol that can function asynchronously, such as DCS protocol proposed by Choi et al. [15], which has a benefit of having a fast convergence speed and being energy efficient compared to the other protocols with similar size or purpose.

Assuming the implementation of DCS was done correctly, the clock values across the hub should be 'fairly' consistent. The asynchronous (or partially synchronous) nature of DTN leaves a room for uncertainties. To combat this potential inconsistency and Byzantine failure of nodes, nodes in a hub do not simply diffuse or broadcast its value to the nodes outside the hub requesting a time synchronization. A node in the hub, upon a recipient of time synchronization request, assumes the position as the acting leader and starts a PBFT process across the hub (see Figure 5 and the previous section). By the end of the Commit stage, every node in the hub should have received clock values of other nodes in the hub. The hub can use this as a chance to run DCS again within themselves, then broadcast (enter the Reply stage) their results back to the client. In other words, DCS is used as an additional layer of consensus.

A disadvantage of forming a time synchronization hub constantly running DCS internally to keep their clocks consistent with each other would be the fact that DCS is not fault-tolerant at all. Even one single Byzantine faulty node in a hub can tamper the clock value that the nodes in the hub converges to. However, for any time synchronization protocols, synchronization error tends to increase the energy consumption significantly, and this applies to DCS as well. Hence, if the amount of energy spent in a hub is unprecedentedly higher than the normal amount, there is a node within the hub that is faulty. Moreover, during the course of processing the time synchronization request, if multiple DCS iterations have taken place in the hub already, the clocks in the hub must be fairly consistent with each other and a clock value that is significantly off can be assumed to be from a faulty node. Therefore, together with the PBFT-based synchronization processes that come after every synchronization request, we expect that faulty node tracing can be done easily for many cases. As another layer of fault-tolerance and practicality, it is recommended to use PBFT that operates under harsh environment, such as [60].

6. NETWORK MODELING

Traditionally, computer networks have been mathematically modeled as a graph $G = (V, E)$ where V represents the set of nodes (end hosts, gateways, routers, etc.) and E the set of edges connecting two vertices. However, the static nature of graph as a data structure often makes it insufficient to model the networks used in modern days which are rather ad hoc and dynamic up to the level where the entire network topology can be changed rapidly and continually. This has motivated lots of researchers to search for new mathematical (data) structures to model computer networks. See also [61].

Hypergraph

The typical approach to modeling networks, graphs, does not naturally capture all types of connectivity. Broadcast and multicast, in particular, benefit from a model where a single edge can connect more than two vertices. This is formalized by the concept of a *hypergraph*. A hypergraph $H = (V_H, E_H)$ consists of a set of vertices V_H and a

set of *hyperedges* E_H, where a hyperedge is a non-empty subset of V_H. In particular, a hyperedge may connect an arbitrary number of vertices. See Figure 6 for an example of a hypergraph. Traditional graphs are also hypergraphs. In a graph, each hyperedge contains two vertices.

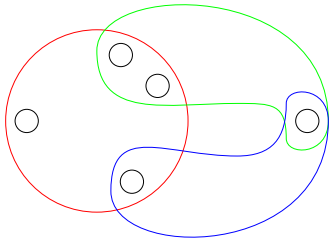


Figure 6. An example of a hypergraph.

Temporal Graph

The temporal realities of DTNs, particularly for space deployment, pushes the need for temporal graphs (and temporal hypergraphs). Representations of temporal graphs over discrete time include sequences of static graphs and time-extended graphs. Over continuous time, representations of temporal graphs include contact graphs as well as edge and vertex labelled graphs, where the edges and vertices are labelled with sets corresponding to the times when they are available.

Cellular Sheaves

One way of rephrasing the time synchronization problem is in terms of local data (times) that may not be globally consistent. The formal structure in mathematics known as the *sheaf* makes precise the notion of either turning local observations into global observations, or providing formal measurements of obstructions to doing so. We use a specialization of sheaves called *cellular sheaves*, which were first described in Shepard's thesis [62] and expanded to include posets and applications in Curry's thesis [63]. Broadly, in the setting of graphs or hypergraphs, cellular sheaves are assignments to the poset of elements (vertex less than edge, or containment of hyperedges) to values in some data category. Each element is assigned a data object and each comparison relation in the (hyper)graph is sent to a corresponding operation on these data structures. Applications of cellular sheaves on posets have found applications in topological data analysis (TDA) [64], sensor networks [65], path optimization problems [66], and neural networks [67].

Network Flows

Consider a diffusion process along a smooth surface (a manifold), for instance, the flow of heat. It is a well-known fact that properties of this diffusion is governed by the surface's curvature [68]. It turns out that there is a method for computing curvature of discrete structures, such as graphs, that analogously captures the behavior of diffusions. Furthermore, this notion of curvature is extendable to temporal graphs. The so-called temporal curvature is readily computable, and can be used to both explore dissemination of information across the network as well as to provide a performance measure that can be employed by a routing algorithm[4] In addition, utilization of paths on the network will perturb the curvature, making curvature a useful diagnostic tool.

[4]For example, consider the Enhanced Interior Gateway Routing Protocol (EIGRP) [69], [RFC 7868].

To be explicit, temporal curvature in a time-evolving network is a quantity that can be computed between any pair of nodes x, y at a fixed time $t \in \mathbb{R}$. This curvature can be negative, zero, or positive-the sign of the curvature gives qualitative information as to the relationship between x and y at this time in the network. Negative curvature indicates that x and y are, on average, "more connected" to each other than "nearby" vertices in the network are. Equivalently, routing information from x to y (at a fixed reference time) is more efficient than routing information from vertices nearby x to vertices nearby y. Conversely, positive curvature indicates that vertices local to x and to y are *more* connected than x and y are. This is particularly evidenced by the following (informally stated result):

Theorem 6.1. *Suppose the curvature between all points x and y at all times $t \in \mathbb{R}$ in a temporal network are positively curved. Then diffusion processes on the network reach equilibrium exponentially quickly in time.*

An alternative characterization (which requires particular assumptions on the temporal network in question) states the following: Suppose one means to transmit information from x to y at time t, in a way that minimizes the average time a "bit" of information originating at x spends en route before reaching y. If the curvature between x and y at time t is positive, it is more beneficial to send the message by splitting it uniformly among all neighbors of x, and if the curvature is negative, it is more beneficial to keep the packet in one component and proceed by single-cast routing. Thus we see that in this case, evaluating curvature provides information as to how well multi-cast versus single-cast routing performs over a temporal network.

An up-coming paper will detail these results and more, but here we give a discussion using a data mule example illustrated in Figure 7. Here there are two small mesh networks on the left and the right, and there are never end-to-end paths between them. Instead, a data mule, shown in blue, periodically moves between them as in the figure. The actual computation of curvature in this system requires choosing several hyperparameters, including the length of time spent in each configuration, but we will avoid this technicality by giving informal estimates: Between vertices in a single cluster, the curvature is constant and always positive. Between vertices in different clusters, the curvature is slightly positive and close to zero. The curvature of the blue node however goes through various changes as the system progresses. Taking a time-average of this curvature over a single period shows that the curvature between the blue node and any of the green nodes is negative. The reason for this is that the blue node acts as a spatio-temporal bottleneck for information passing between the two clusters, and this is detected by the curvature. One checks that this interpretation aligns with the intuitions for curvature provided in the previous paragraph. This example shows that curvature can detect important structural features of temporal networks, including the presence of clusters (via positive curvature) and bottlenecks (with negative curvature). Indeed, this aligns with recent applications of (static) curvature in the network theory literature [70].

If time synchronization is considered as a particular information diffusion process, then the network curvature dictates how this process works. Moreover, different nodes might serve different purposes. For example, some nodes might be a gateway between adjacent network areas, and other nodes might be more authoritative as time sources. Depending on a node's relationship to other nodes, particularly as measured

Figure 7. Example data mule network.

by curvature, role assignment can be optimized. We note that this information is also applicable to routing; for example, consider link state advertisements.

7. Global Synchronization

In this section we present the synchronization problem in the language of algebraic topology and sheaf theory. The first formulation in terms of sheaf theory that we present was already described by Ghrist and Hansen [71], following Bandeira [72]. Later formulations in this section will get at some of the more novel contributions that we have to offer, which sit at varying levels of development.

An Algebraic Topology Take on Clock Synchronization

For a first pass at the general framework, consider the two different network topologies depicted in Figure 8.

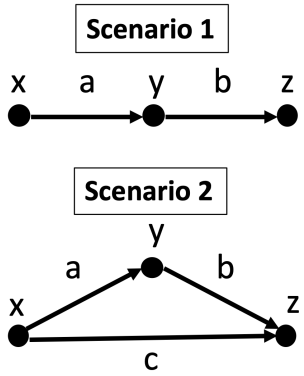


Figure 8. Two Different Network Topologies

In Scenario 1 of Figure 8, we have a network with the "string of pearls" or "daisy chain" topology. In Scenario 2, we have a network with a ring topology, which topologically is equivalent to a circle. Algebraic topology provides us with means to discriminate these two scenarios using linear algebra. To start, we can write down two vector spaces to either scenario: $C^0(X) = \mathbb{R}^{\mathbb{V}(\mathbb{X})}$, the space of *0-cochains*, which is the vector space generated by the vertices in the network X, and $C^1(X) = \mathbb{R}^{\mathbb{E}(\mathbb{X})}$, the space of *1-cochains*, which is the vector space generated by the edges in the network X. The interpretation of these vector spaces for time synchonrization is that a vector $v \in C^0(X)$ describes the set of times being recorded at each node, whereas a vector $w \in C^1(X)$ encodes the possible time differences between each pair of nodes. Purely using the local topology of each network one can then write down the *coboundary matrix* $\delta : C^0(X) \to C^1(X)$, which records the pairwise differences of times, subject to some choice of orientation. Following the scenarios depicted in Figure 8 we have two different coboundary matrices. These are, respectively,

$$\delta_1 = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \quad \text{and} \quad \delta_2 = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ -1 & 0 & 1 \end{bmatrix}.$$

Notice that the presence of an extra edge in Scenario 2 presents itself as an extra row in the corresponding coboundary matrix. In either case, the kernel of the coboundary matrix is generated by the set of times where

$$x(t) = y(t) = z(t).$$

In the language of algebraic topology, this provides an interpretation of the *zeroth cohomology group*:

$$H^0(X) = \ker \delta$$

A standard fact in algebraic topology is that $H^0(X)$ is generated by the connected components in a space. Because both Scenario 1 and 2 involve connected networks, the corresponding zeroth cohomology group is 1-dimensional, i.e. there is only one consistent notion of time. However, vis a vis our introduction, networks that become disconnected are capable of having multiple internally consistent notions of time, and this is perfectly captured in the language of algebraic topology. Methods for finding elements of $H^0(X)$ using diffusion are discussed shortly, but first we consider the somewhat more mysterious higher cohomology groups.

What distinguishes the ring topology from the linear / daisy chain topology, is another quantity connected to the coboundary matrix δ, namely its cokernel. For a graph, this is equivalent to the definition of its *first cohomology group*

$$H^1(X) = \text{coker } \delta = C^1(X)/\text{im } \delta.$$

By the rank-nullity theorem, the two scenarios are distinguished by $H^1(X_1) = 0$ versus $H^1(X_2) \cong \mathbb{R}$, which is generated by the left null space of δ, e.g.

$$H^1(X_2) \cong \left\langle [1, 1, -1]^t \right\rangle.$$

To provide a concrete interpretation of this vector, recall that these are supposed to be time-differences between nodes, so this vector, if it were in the image of our coboundary operator, would yield the system of equations

$$\begin{aligned} y(t) - x(t) &= 1 \\ z(t) - y(t) &= 1 \\ z(t) - x(t) &= -1 \end{aligned}$$

which obviously has no solution. The physical scenario this would describe is that y is one hour ahead of x and z is one hour ahead of y, but, somehow, z is actually one hour *behind* x, where by the transitive property be the case that x is actually two hours behind z. However, because 1-cohomology is defined via a quotient space construction, there are actually lots of time offsets that generate 1-cohomology; see Figure 9 for another example. In effect, as long as the time-delays *don't* satisfy the equation $a + b = c$, then this corresponds to an irreconcilable time synchronization scenario.

As a final point of reflection, one should compare the generating vector for $H^1(X_2)$ with the conflicting instructions given in the Byzantine General Problem, which also had a circular structure as well. The connection between algebraic topology and distributed computing is well established; see [73] for a modern textbook-length treatment.

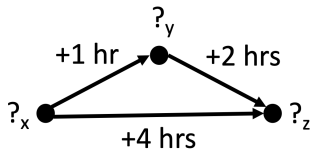


Figure 9. An impossible sequence of temporal offsets is captured via 1-cohomology.

A Sheaf-Theoretic Take on Clock Synchronization

In this section we move beyond algebraic topology and into cellular sheaves, where one can generalize 0-cochains and 1-cochains considerably. Most importantly, with a cellular sheaf, one can instead work with 0-cochains and 1-cochains *valued in the sheaf* $\mathcal{F}$. For the graphs depicted in Figure 8, this means that we can have an arbitrary (potentially infinite dimensional) vector space for each node in the network and each edge in the network. The sheaf restriction maps dictate how to move from, say, $\rho_{x,a} \colon \mathcal{F}(x) \to \mathcal{F}(a)$. The definition of cellular cochains is almost identical:

$$C^0(X;\mathcal{F}) = \bigoplus_{v \in V} \mathcal{F}(v) \qquad C^1(X;\mathcal{F}) = \bigoplus_{e \in E} \mathcal{F}(e)$$

The coboundary map $\delta_{\mathcal{F}}$ is essentially the same as the coboundary defined over the base graph, except instead of simply having $\{0, \pm 1\}$ as entries, one takes that as a coefficient that scales the correspond restriction matrix $\rho_{v,e} : \mathcal{F}(\mathrm{v}) \to \mathcal{F}(\mathrm{e})$, which is now a block matrix inside of $\delta_{\mathcal{F}}$. Definitions of the sheaf cohomology groups are identical—$H^0(X;\mathcal{F}) := \ker \delta_{\mathcal{F}}$ and $H^1(X;\mathcal{F}) := \operatorname{coker} \delta_{\mathcal{F}}$—but sometimes different language is used. In particular, one tends to call $H^0(X;\mathcal{F}) = \{\text{global sections of } \mathcal{F}\}$.

To understand why one might consider a cellular sheaf, consider the setting where instead of having a single time, one has a whole sequence (i.e. a vector) of times over each node. This could be the the time-stamps of the most recent few packets/bundles or even an entire clock history, viewed as a non-decreasing sequence of integers, which count local (milli)seconds in an epoch. Perfect synchronization would correspond then to finding a constant section of this so called *time sheaf*.

Before addressing the impracticalities of perfect synchronization, let us review one efficient method for learning global sections from an arbitrary starting 0-cochain, e.g. a collection of local times, or even whole clock histories.

Laplacians, Sheaf Laplacians and Heat Flows

Given a coboundary map $\delta : C^0 \to C^1$, that is either the ordinary one or the one enriched in a cellular cosheaf, one can naively define the associated Hodge-* or Laplace-Beltrami operator as

$$\Delta = \delta^t \delta : C^0 \to C^0.$$

It is a classical fact $H^0 \cong \ker \Delta$. This identification actually suggests an algorithm: after choosing an appropriate inner product structure on the corresponding cochain spaces, one can define the normalized Laplacian to be the appropriate self-adjoint operator. Under this scaling one can consider the heat equation

$$\partial_t u = \Delta u.$$

Picking an arbitrary co-chain and using this as an initial condition for the above PDE is equivalent to picking arbitrary initial times/clocks for each node in the network. By running a simulation of this heat flow long enough, one eventually diffuses into a steady state u_∞ where

$$\partial_t u_\infty = 0 \implies \Delta u_\infty = 0,$$

i.e. one reaches a 0-cochain that is harmonic, which by Hodge theory coincides with elements in the zeroth cohomology group. One can effectively summarize many of the consensus algorithms described earlier as message-passing alforithms, which are in effect carrying out some simulated heat flow of information.

Group Synchronization and Non-Abelian Sheaf Cohomology

We now describe an ideal synchronization scenario, which operates on the entire, idealized clock function, and phrase this as a group synchronization problem. Following [72], [71], the *group synchronization problem* asks, given a group G, a graph X, and functions $f_{ij} : G \to \mathbb{R}$ for each (possibly directed) edge (i,j) of X, find a function $g : V(X) \to G$ minimizing

$$\text{cost}(g) := \sum_{(i,j) \in E} f_{ij}(g(i)g(j)^{-1}).$$

In the context of angular synchronization, one can imagine that this problem is asking the following: Suppose we're given a collection of pictures of an object, but we don't know what the "true" orientation of each picture is. Instead, all one can do is take each pair of pictures and compute the angular difference between them, written g_{12} in Figure 10[5].

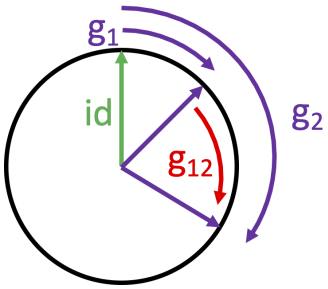


Figure 10. An example of group synchronization on $\mathrm{SO}(2)$.

If we could discover the "true" orientations of each photo, as determined by a rotation from some god-given reference frame, we'd know the actual values of the group elements g_1 and g_2 and these would satisfy the equation:

$$g_{12} = g_2^{-1} g_1$$

In this case the functions would be

$$f_{ij}(\bullet) := \text{dist}(\text{id}, g_{12} \cdot \bullet)$$

where we have implicitly used the fact that angular rotations, like any rotation group, forms a Lie group, which is metrizable. Secretly the functions f_{ij} are just penalty functions measuring deviation from a (equivariant) global section.

For our ideal clock synchronization application, now one wants to assume we have characteristics for how to reparameterize time to go from clock i to clock j. This is encoded

[5] $\mathrm{SO}(2)$ is defined to be the group of square orthogonal matrices, i.e. the set of square matrices M such that $M^\top = M^{-1}$, with determinant 1. It can be seen as the set of all two-dimensional rotation matrices.

as a function $\sigma_{ij} : \mathbb{R} \to \mathbb{R}$, which is a non-decreasing reparameterization of $\mathbb{R}$ that could take a signal $s_i(t)$ measured using node i's internal clock to $s_j(t)$—a signal taken at node j using its internal clock. This is clearly expressible as a cellular sheaf—*the signal alignment sheaf*—which assigns the Hilbert space of L^2 integrable functions to each node and edge, but the restriction morphisms take a signal $s_i(t)$ to $s_i(\sigma_{ij}(t))$. A global section would then correspond to a collection of compatible signals, observed faithfully by all nodes.

8. CONCLUSION

Establishing clock synchronization over large-scale networks under harsh environment is consistently a major challenges in distributed systems and computer networks society. In this paper, we introduced a new perspective of studying global time synchronization over DTN-based SSI. Our proposed approach was analogous to the historical expansion of the Internet we use today, in the way that local development was completed first then global connections were established by interconnecting multiple local area networks via gateways, routers, and bridges. This was not only a natural/canonical approach but also rather inevitable due to the nature of large-scaled DTNs where accurate global 'snapshots' of the entire network if often unobtainable, implying that a novel approach is needed to address this problem. Time synchronization is necessary for routing, behavioral expectations (e.g. deleting expired bundles), and for networked approaches to navigation and distributed science.

The local synchronization protocol we presented involves combination of previously developed approaches and protocols. To summarize, a DTN region first computes a k-hop connected dominating set and designate that set as the time synchronization hub that functions as a distributed 'pseudo' time server. The nodes in the time synchronization hub constantly work together to keep their clocks consistent. When a client (a node outside the hub) sends a time synchronization request to one of the nodes in the hub, the hub will run a fault-tolerant consensus process and return the result(s) back to the client. In this way, we can keep both scalability (complexity) and fault-tolerance while not sacrificing anything significantly. This protocol might be theoretically viable; however, it has not yet been implemented nor experimentally verified.

For global synchronization, we used a data structure derived from algebraic geometry and algebraic topology called a (cellular) sheaf, which is used to 'glue' local data on subsets to construct global information across the entire space. To do this, we introduced an object so-called time sheaf, and reduced the global synchronization problem into the problem of approximating global section of the time sheaf. The definition of the time sheaf remains elusive, but the starting point is to determine what mathematical structure should correspond to time at the vertices, formalizing the local data of local clocks. We present two main candidates here. The first is all real numbers, $\mathbb{R}$, which implies that time extends from negative to positive infinity. We can also consider non-negative real numbers, $\mathbb{R}^{\geq 0}$, corresponding to seconds since some epoch.

The value of this is not just the rigor of the sheaf structure, but that sheaves give rise to algorithms [66]; indeed on top of these sheaf data structures further algorithms and mathematical machinery can be applied and also deduced. In [66], the active algorithm of path finding was recast into a passive data structure – a particular sheaf where the global sections of this sheaf represent shortest paths. We expect that finding a rigorous definition of time sheaf could allow us to begin the implementation of our protocol. As the intention of this paper was to found a theoretical ground, implementation details are omitted, but will (and need to) be studied as it gets more well-established in both theoretical and algorithmic perspectives.

9. FUTURE WORKS

The following are proposed research topics that we believe are closely related to the problems we discussed throughout this paper. Addressing them can create a significant advancement on the current state of this research.

Reliable Message Delivery

Most of the networks an d distributed systems actively used today assume the processes behave synchronously. However, asynchronous phenomenons, such as unbounded message delay, can be present at any time. If the messages are sent or delivered unordered, reaching consensus over such systems can be very difficult compared to the cases where messages are ordered. Fortunately, TCP provides reliable message delivery service which ensures not only the messages are not lost or duplicated but also they are not delivered out of order. However, not every node in SSI is expected to have TCP for their transport layer; some nodes (e.g., nodes in space) can use LTP on top of UDP, which do not have packet reordering mechanisms that exist in TCP as mentioned above. Given that LTP is designed to be an additional reliable convergence layer for BP (and UDP), an upgraded version of LTP with reliable data transfer protocol can immensely improve the development and integration of SSI.

Secure Protocols

Security of DTN or SSI may not be a topic highly relevant to this paper, but there are evident trade-offs between security and computational efficiency, especially when constructing Byzantine fault-tolerant protocols. Our current understanding of decentralized consensus algorithms is that one cannot improve performance without the expense of security [74]. To support this, numerous consensus and clock synchronization schemes appear to be not only more secure but also more efficient when a digital signature protocol is added. As discussed, the initial PBFT protocol in [40] requires the use of a digital signature, but it can be computationally optimized by using MAC instead of a digital signature. This is a valid approach; however, MAC does not satisfy message secrecy and assumes the users utilize a private-key encryption scheme where key exchanges can be more difficult than public-key encryptions and should be only done amongst trusted users. Moreover, BP and BPSec specifications [RFC 9171, 9172] state that every bundle must have a bundle header with an entry (extension block) called Block Integrity Block (BIB) specifically allocated for the signature of the source node's application signed using its private key, which is used to verify the integrity of the plaintext. This implies that PKI more or less is a strict requirement. However, as demonstrated in [75], PKI for the purpose of authentication may not be necessary.

There also seem to be canonical trade-offs among security, efficiency, and fault tolerance. As mentioned, one of the BFT clock synchronization schemes in [53] manages to tolerate f Byzantine faulty nodes only with $2f + 1$ nodes when

using a digital signature. This can be easily implemented and executed today; for instance, any end hosts connected to the Internet can generate and exchange a pair of keys (public and private) with TLS/SSL protocol that uses the Diffie-Hellman key exchange scheme. However, just as our current understanding of the DTN-based SSI still remains purely at the theoretical level, it is a challenge to even begin thinking of the security of SSI, let alone the construction of a viable public key infrastructure (PKI) – if it is even viable. Per BPSec specification [RFC 6257, 9172], key management and distribution are currently being assumed to be provided as a part of external services from the network management or centralized key server (certificate authority), which can be impractical for some instances, such as where the end-to-end connection between a client and the server cannot be established on time. For example, this might be before the expiration of the request or the key. See [76] for further details regarding recent progress in key management in DTNs.

Byzantine failure is the most general type of fault model that encompasses any class of faults, including the case where a node is arbitrarily malicious. Byzantine adversaries can be fatal to any kind of network or protocol. They can even be adapted to interfere with a consensus process specifically directing the system to reach consensus incorrectly as they desire [77]. This can be directly applied to our or previous constructions, as a form of an adversary intentionally broadcasting incorrect clock values while disguising itself as a non-faulty node. This suggests that more intensive studies regarding (Byzantine) adversary detection and prevention, not only in theoretical levels but also for application and practical settings specifically for DTNs, must be further conducted as done in [78] and [79].

Complexity and Scalability

Although PBFT requires $O(n^2)$ messages only, it still can be intensive to handle for low-performance devices if n grows significantly or on networks with infrequent data exchanges. Recall that we must have $n \geq 3f + 1$ in order to tolerate f faulty nodes. This does not scale well as it requires triple or more nodes to defend the system against one more faulty node. Also, it is challenging to correctly predict the supremum of the number of faulty nodes of a system beforehand; in fact, the supremum may not even exist or be finite. It is clearly cost-ineffective to add multiple nodes simply to prevent faulty nodes from dominating the system based on their numbers that we do not even have a good estimation of.

Continuing the discussion from the previous subsection, we expect that the assumption of Byzantine failure is overly general and can be relaxed or reduced to different types of failure models. That is, under the premise that our system is free of adversaries and is robust enough, it is unlikely to demonstrate any Byzantine failure symptoms. As mentioned, it was already proven in [49] that $3f+1$ is the minimum number of nodes that can tolerate up to f Byzantine faulty nodes in an asynchronized system (even with a digital signature), and the same result holds regarding clock synchronization in the presence of Byzantine faults. However, indeed, this does not imply that one can be completely oblivious or disregard Byzantine failures. Byzantine failures are more common in practice than they are generally believed to be [80]. Byzantine failures are not 'isolated' from other types of failures; in many occasions, they occur as a form of a developed version of a non-Byzantine failure that was existing in the system already. See [80] for more details regarding Byzantine general problems in practice.

The nature of BP canonically makes DTNs closer to partially synchronous than asynchronous, because every bundle comes with a timestamp and its lifespan [RFC 9171]. There clearly is an advantage of choosing synchronized models over asynchronized one. Katz and Koo in [81] presented a Byzantine consensus algorithm that tolerates f faulty nodes with only $2f + 1$ nodes with 24 expected rounds. Then Abraham et al. in [82] constructed an algorithm with a better lower-bound of 8 expected rounds. Partially synchronous models, however, has not been studied actively.

Self-Stabilization

Another promising perspective of studying fault-tolerant clock synchronization is to model it as a self-stabilizing distributed system. Self-Stabilization, first introduced by Dijkstra in [83], is defined to be the characteristic of a distributed system that is always guaranteed to (1) end up in a legitimate state starting from any state within a finite amount of time, and (2) remain in a legitimate state if started from a legitimate state and no failure occurring during the process. The first property is often referred to as convergence and the second is as closure. Basically, a distributed system is called self-stabilizing if it is capable of automatically recovering from transient faults and their effects.

For clock synchronization, it is an unfair requirement for a system to have its clocks never fail, especially if it is asynchronous or partially synchronous. The works of Lamport and Melliar-Smith [53] were revisited by Dolev and Welch in [84] as a form of probabilistic self-stabilizing Byzantine fault-tolerant clock synchronization protocol, which is later revisited by Malekpour in [85] as a form of a deterministic protocol with self-stabilization property, which, however, is still computationally extensive for low-end devices.

Approximation Algorithm

Recall from our construction that the nodes responsible for local clock synchronization were chosen as the nodes forming a k-hop connected dominating (CDS) set of a DTN region. We currently expect that the gateway is the one that is responsible for computing a k-hop CDS and designating them as the time synchronization hub. However, dominating set problem is computationally difficult problem. The decisional dominating set problem that takes a graph G and a number n as inputs and returns whether there exists a dominating set with n or fewer vertices is NP-Complete, and the computational version of the problem that returns the smallest dominating set given a graph G is NP-Hard. Finding the minimum connected dominating set (MinCDS) problem is also NP-Hard. Hence, in practice, approximation and distributed algorithms are used for those problems.

Moreover, DTNs are characterized by a constant change of network topology, hence are more accurately captured when modeled as temporal graphs. There are multiple types of dominating sets that can be defined on a temporal graph. One of them is a permanent dominating set, which is the set of nodes that is a dominating set of that graph at all times. It can be shown that the permanent dominating set (PDS) problem is also NP-Complete by reducing the traditional dominating set problem to the PDS problem. There are many other types of dominating sets of temporal graphs available. However, it has not yet been fully understood what type fits our purpose the most and how they can be computed or approximated efficiently (i.e. algorithmic lower/upper

bounds and hardnesses). Resolving these problems will be beneficial to not only the time synchronization problem but also to the theoretical computer science community overall.

Synchronization Error

It is not always possible to have every machine's software clock perfectly synchronized without any prior knowledge about the network connections. Its accuracy can also be negatively affected by propagation delay, especially if it is asymmetric [36]. Synchronization error is rather inevitable. If two nodes are far apart from each other yet do not establish direct communication between the two, their clocks would be loosely synchronized with a high probability, and the situation would indeed be worse for DTNs.

Fan and Lynch in [86] introduced a concept of gradient clock synchronization (GCS), where the term 'gradient' here denotes the property where the time difference between two nodes must be bounded above by a non-decreasing function of the distance between the two. Assuming that the communication is reliable and message delay and clock drift are bounded, it can be shown that the clock offset between any two nodes is $\Omega\left(d + \log D / \log\log D\right)$ where d is the distance between two nodes and D is the diameter of the network. It is yet unknown whether this bound is the tightest upper bound or not. This bound indeed does not hold exactly at the presence of Byzantine faults, but fault-tolerant GCS protocols can still exist; for example, see [87].

In many occasions, time skew or offset occurs by the software clock miscounting or the hardware clock misproducing oscillations. This leads to the conclusion that, just as [15] does, it is always beneficial to synchronize clock frequencies as well, not just the clock values. However, clock frequencies are computed as $f(t) = dC(t)/dt$, so a clock computing its own clock frequency would not result in a correct frequency, because a node would have $C(t)$ but there is no t that it can reference from. A more precise way of computing relative frequency in an accurate way must be studied.

Relativistic Effects

The SSI can be large-scaled enough to experience the effects of relativity, such as time dilation and redshift. In fact, there have been multiple papers reporting the effects of relativity that occurred during the process of satellite-based clock synchronizations, such as [88]. For a theoretical treatment of this, we suggest that a mathematical formulation of clock synchronization in terms of coordinate time with relativistic transformation should be done first.

DTN Local Area Network

While it is true that the concept of local area networks and regional networks exist in DTN [56], [RFC 4838, 5050], they still remain as rather non-well-defined terms as pointed out in, for example, [89]. It sometimes is defined in terms of the geographical locations of the nodes or even defined in persistent homological ways. We have also noticed that the term '*DTN Region*' has only appeared seldomly in recent publications studying DTN architectures, whereas it frequently appeared in publications before the early 2010s. We speculate that this could be another part of the scaling issue of DTNs, and further research studying the most efficient method of 'dividing up' the DTNs into multiple local regions must be conducted, especially as our construction of local synchronization relies on it.

Other Mathematical Approaches

As discussed, the formal definition and construction of the so-called time sheaf are still not formally well-defined. We concluded the paper by suggesting two candidates, namely $\mathbb{R}$ and $\mathbb{R}^{\geq 0}$. Another more abstract possibility comes from torsors, which correspond to relative measurements (among other things); see [90] for a quick introduction. Then the structures – perhaps vector spaces – sitting above the edges must be determined, along with the restriction maps [91].

While this sheaf-based approach seems promising, we do not exclude the potential of other directions or methods being useful for our problem. Readers unfamiliar with recent progress in the development of mathematical tools for modeling DTNs can use [61] as a reference. We suggest a new possible approach here. For global synchronization, we may represent the SSI as a graph where each node is a gateway. Assuming the SSI is for IPN Internet connecting two planets (e.g., Earth and Mars), the topology of this network would be close to a ring. This network then can be translated into a closed two-dimensional polygon where the (discrete) curvature [92] at each node is $\kappa = 1/\tau$ such that τ is the current time of each node. Then this polygon can be evolved to a constant-mean-curvature surface, which would be a circle in the two-dimensional case, by 'flowing' mean curvature flow (or curve-shortening flow) across the network. One issue with such an approach is the singularities that appear in non-convex regions. Kazhdan, Solomon, and Ben-Chen in [93] constructs a working modified formulation of discrete mean curvature flow that removes or ignores singularities with experimental evidence, but the paper lacks mathematical proofs.

Analysis and Simulations

This paper aimed to build—and suggest the necessity of—a theoretical foundation for the global time synchronization of large-scale DTNs. The implementation and testing were not conducted during the course of this research. However, the practicality of our method indeed needs to be tested to be considered fully valid. Our construction is built using algebraic topology, in particular the sheaf theory, where practical software implementations and simulations are possible by using PySheaf [94], [65].

A method that one can endeavor is to find a topological analog of dynamic time warping using techniques from topological data analysis, and design a loss function that numerically measures closeness to a (global) section using persistence homological methods such as the one introduced in [95]. Once that is accomplished, another future direction would be to compare our method with other methods serving similar purposes such as [15], [96].

ACKNOWLEDGEMENT

This work was primarily done during the Summer 2022 NASA Space Communications and Navigation (SCaN) Internship Program (SIP) at NASA Goddard Space Flight Center (GSFC). The authors are grateful to NASA GSFC and SCaN program for generous support and funding. Hwang thanks Wenbo Xie and Ethan Dickey at Purdue University for providing extensive assistance in understanding the performance of distributed consensus algorithms and digesting important technical details in RFC 9171, respectively. Hylton thanks Mahyar Malekpour at NASA Langley Research

Center for inspiring conversations regarding realistic failure models, the practicality of Byzantine failure, and the current state of clock synchronization research at NASA. Lastly, the authors thank anonymous reviewers for constructive comments, especially the one who suggested a better title for this paper.

REFERENCES

[1] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," in *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY: ACM, 2003, pp. 27–34. [Online]. Available: https://doi.org/10.1145/863955.863960

[2] NASPI Time Synchronization Task Force, "Time Synchronization in the Electric Power System," North American Synchrophasor Initiative (NASPI), Tech. Rep., 03 2017, NASPI-2017-TR-001, PNNL-26331. [Online]. Available: https://www.naspi.org/node/608

[3] I. Skog and P. Handel, "Time Synchronization Errors in Loosely Coupled GPS-Aided Inertial Navigation Systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1014–1023, 2011. [Online]. Available: https://doi.org/10.1109/TITS.2011.2126569

[4] Y. Mostofi and D. Cox, "Mathematical Analysis of the Impact of Timing Synchronization Errors on the Performance of an OFDM System," *IEEE Transactions on Communications*, vol. 54, no. 2, pp. 226–230, 2006. [Online]. Available: https://doi.org/10.1109/TCOMM.2005.861675

[5] D. J. Israel, K. D. Mauldin, C. J. Roberts, J. W. Mitchell, A. A. Pulkkinen, D. C. La Vida, M. A. Johnson, S. D. Christe, and C. J. Gramling, "LunaNet: A Flexible and Extensible Lunar Exploration Communications and Navigation Infrastructure," in *IEEE Aerospace Conference*, 2020, pp. 1–14. [Online]. Available: https://doi.org/10.1109/AERO47225.2020.9172509

[6] I. Cardei, C. Liu, J. Wu, and Q. Yuan, "DTN Routing with Probabilistic Trajectory Prediction," in *Wireless Algorithms, Systems, and Applications*. Springer Berlin Heidelberg, 2008, pp. 40–51. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-88582-5_7

[7] S. Grasic and A. Lindgren, "An Analysis of Evaluation Practices for DTN Routing Protocols," in *Proceedings of the Seventh ACM International Workshop on Challenged Networks*, ser. CHANTS '12. New York, NY: ACM, 2012, p. 57–64. [Online]. Available: https://doi.org/10.1145/2348616.2348629

[8] M. Seligman, K. Fall, and P. Mundur, "Storage Routing for DTN Congestion Control," *Wireless Communications and Mobile Computing*, vol. 7, no. 10, pp. 1183–1196, 2007. [Online]. Available: https://doi.org/10.1002/wcm.521

[9] H. Li, H. Zhou, H. Zhang, and B. Feng, "EmuStack: An OpenStack-Based DTN Network Emulation Platform," in *2016 International Conference on Networking and Network Applications (NaNA)*, 2016, pp. 387–392. [Online]. Available: https://doi.org/10.1109/NaNA.2016.24

[10] L. Wood, W. D. Ivancic, W. M. Eddy, D. Stewart, J. Northam, C. Jackson, and A. da Silva Curiel, "Use of the Delay-Tolerant Networking Bundle Protocol from Space," in *59th International Astronautical Congress and Exhibition 2008 (IAC)*, vol. 5, Glasgow, Scotland, UK, 09 2008. [Online]. Available: https://ntrs.nasa.gov/search.jsp?R=20090020378

[11] W. Ivancic, W. M. Eddy, D. Stewart, L. Wood, P. Holliday, C. Jackson, and J. Northam, "Experience with Delay-Tolerant Networking from Orbit," *International Journal of Satellite Communications and Networking*, vol. 28, no. 5-6, pp. 335–351, 2010. [Online]. Available: https://doi.org/10.1002/sat.966

[12] J. Morgenroth and L. Wolf, "Time-Reference Distribution in Delay Tolerant Networks," in *Proceedings of the Seventh ACM International Workshop on Challenged Networks*, ser. CHANTS '12. New York, NY: ACM, 2012, p. 1–8. [Online]. Available: https://doi.org/10.1145/2348616.2348618

[13] Q. Ye and L. Cheng, "DTP: Double-Pairwise Time Protocol for Disruption Tolerant Networks," in *The 28th International Conference on Distributed Computing Systems*, 07 2008, pp. 345–352. [Online]. Available: https://doi.org/10.1109/ICDCS.2008.73

[14] B. J. Choi and X. Shen, "Distributed Clock Synchronization in Delay Tolerant Networks," in *2010 IEEE International Conference on Communications*, 2010, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC.2010.5502781

[15] B. J. Choi, H. Liang, X. Shen, and W. Zhuang, "DCS: Distributed Asynchronous Clock Synchronization in Delay Tolerant Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 491–504, 2012. [Online]. Available: https://doi.org/10.1109/TPDS.2011.179

[16] I. F. Akyildiz, Özgür B. Akan, C. Chen, J. Fang, and W. Su, "InterPlaNetary Internet: State-of-the-art and Research Challenges," *Computer Networks*, vol. 43, no. 2, pp. 75–112, 2003. [Online]. Available: https://doi.org/10.1016/S1389-1286(03)00345-1

[17] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Pearson, 2021, E-Book, ISBN-13 978-0-1359-2861-5.

[18] R. Wang, S. C. Burleigh, P. Parikh, C.-J. Lin, and B. Sun, "Licklider Transmission Protocol (LTP)-Based DTN for Cislunar Communications," *IEEE/ACM Transactions on Networking*, vol. 19, no. 2, pp. 359–368, 12 2010. [Online]. Available: https://doi.org/10.1109/TNET.2010.2060733

[19] B. J. Choi and X. Shen, "Adaptive Asynchronous Sleep Scheduling Protocols for Delay Tolerant Networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 9, pp. 1283–1296, 2010. [Online]. Available: https://doi.org/10.1109/TMC.2010.229

[20] W. Ivancic, W. M. Eddy, L. Wood, D. Stewart, C. Jackson, J. Northam, and A. da Silva Curiel, "Delay/Disruption-Tolerant Network Testing Using a LEO Satellite," in *NASA Earth Science Technology Conference (ESTC 2008)*. University of Maryland, 06 2008. [Online]. Available: http://personal.ee.surrey.ac.uk/Personal/L.Wood/publications/uk-dmc-dtn-saratoga-testing-estc-2008.pdf

[21] H. Ochiai, K. Shimotada, and H. Esaki, "IP over DTN: Large-Delay Asynchronous Packet Delivery

in the Internet," in *2009 International Conference on Ultra Modern Telecommunications & Workshops*, 2009, pp. 1–4. [Online]. Available: https://doi.org/10.1109/ICUMT.2009.5345643

[22] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 04 1988. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/42282.42283

[23] S. Dimitriou and V. Tsaoussidis, "Effective Buffer and Storage Management in DTN Nodes," in *2009 International Conference on Ultra Modern Telecommunications & Workshops*, 2009, pp. 1–3. [Online]. Available: https://doi.org/10.1109/ICUMT.2009.5345376

[24] Z. Lu and J. Fan, "Delay/Disruption Tolerant Network and Its Application in Military Communications," in *2010 International Conference On Computer Design and Applications*, vol. 5, 2010, pp. V5–231. [Online]. Available: https://doi.org/10.1109/ICCDA.2010.5541302

[25] C. Caini, H. Cruickshank, S. Farrell, and M. Marchese, "Delay- and Disruption-Tolerant Networking (DTN): An Alternative Solution for Future Satellite Networking Applications," *Proceedings of the IEEE*, vol. 99, no. 11, pp. 1980–1997, 2011. [Online]. Available: https://doi.org/10.1109/JPROC.2011.2158378

[26] P. Wan, Y. Zhan, and X. Pan, "Solar System Interplanetary Communication Networks: Architectures, Technologies and Developments," *Science China Information Sciences*, vol. 61, no. 4, pp. 1–26, 2018. [Online]. Available: https://doi.org/10.1007/s11432-017-9346-1

[27] J. J. Rodriguez, Ed., *Advances in Delay-Tolerant Networks (DTNs): Architecture and Enhanced Performance*, 2nd ed., ser. Woodhead Publishing Series in Electronic and Optical Materials. Woodhead Publishing, 2021.

[28] D. Newell and E. Tiesinga, *The International System of Units (SI), 2019 Edition*. NIST, Gaithersburg, MD, 08 2019, U.S. version of BIPM SI Brochure (9th Edition). [Online]. Available: https://doi.org/10.6028/NIST.SP.330-2019

[29] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 07 1978. [Online]. Available: https://doi.org/10.1145/359545.359563

[30] J. L. Welch and N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, 1988, Conference version published in 1984: https://doi.org/10.1145/800222.806738. [Online]. Available: https://doi.org/10.1016/0890-5401(88)90043-0

[31] K. Römer, "Time Synchronization in Ad Hoc Networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '01)*, 10 2001, pp. 173–182. [Online]. Available: https://doi.org/10.1145/501436.501440

[32] D. L. Mills, "Internet Time Synchronization: the Network Time Protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991. [Online]. Available: https://doi.org/10.1109/26.103043

[33] ——, "Improved Algorithms for Synchronizing Computer Network Clocks," *IEEE/ACM Transactions on networking*, vol. 3, no. 3, pp. 245–254, 1995. [Online]. Available: https://doi.org/10.1109/90.392384

[34] K. A. Marzullo, "Maintaining the Time in a Distributed system: An Example of a Loosely-Coupled Distributed Service," Ph.D. dissertation, Stanford University, 02 1984. [Online]. Available: https://searchworks.stanford.edu/view/1137192

[35] "Windows Time Service (W32Time)," Networking Documentation, Microsoft Technical Documentation, 11 2021. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/networking/windows-time-service/windows-time-service-top

[36] D. L. Mills, *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space*, 2nd ed. Boca Raton, FL: CRC Press Taylor & Francis Group, 12 2017, ISBN 978-1-4398-1463-5.

[37] L. Lamport, "The Part-time Parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, p. 133–169, may 1998. [Online]. Available: https://doi.org/10.1145/279227.279229

[38] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, pp. 382–401, 07 1982. [Online]. Available: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/

[39] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/322186.322188

[40] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI '19)*, vol. 99, no. 1999, New Orleans, LA, 02 1999, pp. 173–186. [Online]. Available: https://pmg.csail.mit.edu/papers/osdi99.pdf

[41] P. Feldman and S. Micali, "An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement," *SIAM Journal on Computing*, vol. 26, no. 4, pp. 873–933, 1997. [Online]. Available: https://doi.org/10.1137/S0097539790187084

[42] F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 12 1990. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/98163.98167

[43] B. Wester, J. Cowling, E. B. Nightingale, P. M. Chen, J. Flinn, and B. Liskov, "Tolerating Latency in Replicated State Machines through Client Speculation," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '09, 2009, pp. 245–260. [Online]. Available: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/wester/wester.pdf

[44] M. Vukolić, "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication," in *International workshop on open problems in network security*. Springer, 2015, pp. 112–125. [Online]. Available: https://doi.org/10.1007/978-3-319-39028-4_9

[45] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 2, pp.

125–143, 03 1977. [Online]. Available: https://lamport.azurewebsites.net/pubs/proving.pdf

[46] B. Alpern and F. B. Schneider, "Defining Liveness," *Information processing letters*, vol. 21, no. 4, pp. 181–185, 1985. [Online]. Available: https://doi.org/10.1016/0020-0190(85)90056-0

[47] Z. Manna and A. Pnueli, "A Hierarchy of Temporal Properties," in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, 1990, pp. 377–410, full version published in 1987 https://purl.stanford.edu/mm910jx2556. [Online]. Available: https://doi.org/10.1145/93385.93442

[48] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3149.214121

[49] G. Bracha and S. Toueg, "Asynchronous Consensus and Broadcast Protocols," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 824–840, 10 1985. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/4221.214134

[50] P. Dutta, R. Guerraoui, and L. Lamport, "How fast can eventual synchrony lead to consensus?" in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, 2005, pp. 22–27. [Online]. Available: https://doi.org/10.1109/DSN.2005.54

[51] J. A. Garcia-Macias and J. Gomez, *MANET versus WSN*. Springer Berlin Heidelberg, 2007, pp. 369–388. [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-37366-7_17.pdf

[52] D. Niyato, E. Hossain, D.-I. Kim, V. Bhargava, and L. Shafai, Eds., *Wireless-Powered Communication Networks: Architectures, Protocols, and Applications*. Cambridge University Press, 12 2016, ISBN 9781316471845.

[53] L. Lamport and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," *Journal of the ACM (JACM)*, vol. 32, no. 1, pp. 52–78, 1985, (Note: Its preprint first appeared in 1981 as a form of SRI Technical Reports.). [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2455.2457

[54] D. Dolev, J. Halpern, and H. R. Strong, "On the Possibility and Impossibility of Achieving Clock Synchronization," *Journal of Computer and System Sciences*, vol. 32, no. 2, pp. 230–250, 04 1986. [Online]. Available: https://doi.org/10.1016/0022-0000(86)90028-0

[55] M. Sasabe and T. Takine, "A Simple Scheme for Relative Time Synchronization in Delay Tolerant MANETs," in *2009 International Conference on Intelligent Networking and Collaborative Systems*, Barcelona, Spain, 11 2009, pp. 395–396. [Online]. Available: https://doi.org/10.1109/INCOS.2009.20

[56] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-Tolerant Networking: An Approach to Interplanetary Internet," *IEEE Communications Magazine*, vol. 41, no. 6, pp. 128–136, 2003. [Online]. Available: https://doi.org/10.1109/MCOM.2003.1204759

[57] J. Wu, L. Zhang, Y. Bai, and Y. Sun, "Cluster-Based Consensus Time Synchronization for Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 15, no. 3, pp. 1404–1413, 2015. [Online]. Available: https://doi.org/10.1109/JSEN.2014.2363471

[58] R. Short, A. Hylton, R. Cardona, R. Green, G. Bainbridge, M. Moy, and J. Cleveland, "Towards Sheaf Theoretic Analyses for Delay Tolerant Networking," in *2021 IEEE Aerospace Conference (AeroConf 2021)*, 03 2021, pp. 1–9. [Online]. Available: https://doi.org/10.1109/AERO50100.2021.9438167

[59] C. Neipp, A. Hernández, J. Rodes, A. Márquez, T. Beléndez, and A. Beléndez, "An Analysis of the Classical Doppler Effect," *European journal of physics*, vol. 24, no. 5, pp. 497–505, 2003. [Online]. Available: https://iopscience.iop.org/article/10.1088/0143-0807/24/5/306

[60] V. Ramaswamy and D. Penny, "On the Performance of PBFT-based Permissioned Blockchain Networks in Constraint Environments," in *2021 IEEE International Conference on Communications (ICC 2021)*, 2021, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC42927.2021.9500636

[61] A. Hylton, R. Short, J. Cleveland, O. Freides, Z. Memon, R. Cardona, R. Green, J. Curry, S. Gopalakrishnan, D. V. Dabke, B. Story, M. Moy, and B. Mallery, "A Survey of Mathematical Structures for Lunar Networks," in *2022 IEEE Aerospace Conference (AeroConf)*, 2022. [Online]. Available: https://ntrs.nasa.gov/citations/20220003566

[62] A. D. Shepard, "A Cellular Description of the Derived Category of a Stratified Space," Ph.D. dissertation, Brown University, 1985. [Online]. Available: https://www.proquest.com/docview/303401715

[63] J. M. Curry, "Sheaves, Cosheaves and Applications," Ph.D. dissertation, University of Pennsylvania, 2014. [Online]. Available: https://www.proquest.com/docview/1553207954

[64] ——, "Topological Data Analysis and Cosheaves," *Japan Journal of Industrial and Applied Mathematics*, vol. 32, no. 2, pp. 333–371, 2015. [Online]. Available: https://doi.org/10.1007/s13160-015-0173-9

[65] M. Robinson, *Topological Signal Processing*, ser. Mathematical Engineering. Springer Berlin Heidelberg, 2014, ISBN 9783642361043.

[66] M. Moy, R. Cardona, R. Green, J. Cleveland, A. Hylton, and R. Short, "Path Optimization Sheaves," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2012.05974

[67] C. Bodnar, F. Di Giovanni, B. P. Chamberlain, P. Liò, and M. M. Bronstein, "Neural Sheaf Diffusion: A Topological Perspective on Heterophily and Oversmoothing in GNNs," *arXiv*, 2022. [Online]. Available: https://arxiv.org/abs/2202.04579

[68] A. Figalli, C. Villani, S. Bianchini, E. Carlen, and A. Mielke, *Optimal Transport and Curvature*. Springer, 07 2011, pp. 171–217. [Online]. Available: https://doi.org/10.1007/978-3-642-21861-3_4

[69] J. Jimenez, "Understand and Use the Enhanced Interior Gateway Routing Protocol," CISCO, 01 2003. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/16406-eigrp-toc.html

[70] J. Sia, E. Jonckheere, and P. Bogdan, "Ollivier-Ricci Curvature-Based Method to Community Detection in Complex Networks," *Scientific Reports*, vol. 9, no. 1,

05 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-46079-x

[71] J. Hansen and R. Ghrist, "Toward a Spectral Theory of Cellular Sheaves," *Journal of Applied and Computational Topology*, vol. 3, no. 4, pp. 315–358, 08 2019. [Online]. Available: https://doi.org/10.1007/s41468-019-00038-7

[72] A. S. Bandeira, "Convex Relaxations for Certain Inverse Problems on Graphs," Ph.D. dissertation, Princeton University, 2015. [Online]. Available: http://arks.princeton.edu/ark:/88435/dsp01jq085n29

[73] M. Herlihy, D. Kozlov, and S. Rajsbaum, *Distributed Computing Through Combinatorial Topology*. Elsevier Science, 2013, ISBN 9780124047280.

[74] L. Li, P. Shi, X. Fu, P. Chen, T. Zhong, and J. Kong, "Three-Dimensional Tradeoffs for Consensus Algorithms: A Review," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1216 – 1228, 12 2021. [Online]. Available: https://doi.org/10.1109/TNSM.2021.3133933

[75] J. Burgess, G. D. Bissias, M. D. Corner, and B. N. Levine, "Surviving Attacks on Disruption-Tolerant Networks without Authentication," in *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '07. New York, NY: ACM, 2007, p. 61–70. [Online]. Available: https://doi.org/10.1145/1288107.1288116

[76] S. A. Menesidou, V. Katos, and G. Kambourakis, "Cryptographic Key Management in Delay Tolerant Networks: A Survey," *Future Internet*, vol. 9, no. 3, p. 26, 2017. [Online]. Available: https://doi.org/10.3390/fi9030026

[77] A. Groce, J. Katz, A. Thiruvengadam, and V. Zikas, "Byzantine Agreement with a Rational Adversary," in *Automata, Languages, and Programming*, vol. 7392. Springer Berlin Heidelberg, 2012, pp. 561–572. [Online]. Available: https://doi.org/10.1007/978-3-642-31585-5_50

[78] Z. Gao, H. Zhu, S. Du, C. Xiao, and R. Lu, "PMDS: A Probabilistic Misbehavior Detection Scheme in DTN," in *2012 IEEE International Conference on Communications (ICC)*, 11 2012, pp. 4970–4974. [Online]. Available: https://doi.org/10.1109/ICC.2012.6364184

[79] P. Asuquo, H. Cruickshank, C. P. A. Ogah, A. Lei, and Z. Sun, "A Distributed Trust Management Scheme for Data Forwarding in Satellite DTN Emergency Communications," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 2, pp. 246–256, 02 2018. [Online]. Available: https://doi.org/10.1109/JSAC.2018.2804098

[80] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, "Byzantine Fault Tolerance, from Theory to Reality," in *SAFECOMP 2003: Computer Safety, Reliability, and Security*. Springer Berlin Heidelberg, 2003, pp. 235–248. [Online]. Available: https://doi.org/10.1007/978-3-540-39878-3_19

[81] J. Katz and C.-Y. Koo, "On Expected Constant-round Protocols for Byzantine Agreement," *Journal of Computer and System Sciences*, vol. 75, no. 2, pp. 91–112, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000008000718

[82] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren, "Efficient Synchronous Byzantine Consensus," Cryptology ePrint Archive, Paper 2017/307, 2017. [Online]. Available: https://eprint.iacr.org/2017/307

[83] E. W. Dijkstra, "Self-stabilizing Systems in spite of Distributed Control," *Communications of the ACM*, vol. 17, no. 11, pp. 643–644, 1974. [Online]. Available: https://www.cs.utexas.edu/~EWD/ewd04xx/EWD426.PDF

[84] S. Dolev and J. L. Welch, "Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults," *Journal of the ACM*, vol. 51, no. 5, p. 780–799, sep 2004. [Online]. Available: https://doi.org/10.1145/1017460.1017463

[85] M. R. Malekpour, "A Byzantine-Fault Tolerant Self-Stabilizing Protocol for Distributed Clock Synchronization Systems," in *Symposium on Self-Stabilizing Systems*, vol. 4280. Springer Berlin Heidelberg, 2006, pp. 411–427. [Online]. Available: https://doi.org/10.1007/978-3-540-49823-0_29

[86] R. Fan and N. Lynch, "Gradient Clock Synchronization," in *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '04, 2004, p. 320–327. [Online]. Available: https://doi.org/10.1145/1011767.1011815

[87] J. Bund, C. Lenzen, and W. Rosenbaum, "Fault Tolerant Gradient Clock Synchronization," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, ser. PODC '19. New York, NY: ACM, 2019, p. 357–365. [Online]. Available: https://doi.org/10.1145/3293611.3331637

[88] J. Wang, Z. Tian, J. Jing, and H. Fan, "Influence of Relativistic Effects on Satellite-based Clock Synchronization," *Phys. Rev. D*, vol. 93, p. 065008, Mar 2016. [Online]. Available: https://doi.org/10.1103/PhysRevD.93.065008

[89] A. Hylton and D. E. Raible, "Networked Operations of Hybrid Radio Optical Communications Satellites," in *32nd AIAA International Communications Satellite Systems Conference: Optical Communications Network I*, 08 2014, pp. 1–11. [Online]. Available: https://doi.org/10.2514/6.2014-4440

[90] J. Baez, "Torsors Made Easy," 12 2009. [Online]. Available: https://math.ucr.edu/home/baez/torsors.html

[91] M. Robinson, "Assignments to Sheaves of Pseudometric Spaces," *Compositionality*, vol. 2, p. 2, jun 2020. [Online]. Available: https://doi.org/10.32408%2Fcompositionality-2-2

[92] K. Crane and M. Wardetzky, "A Glimpse into Discrete Differential Geometry," *Notices of the American Mathematical Society*, vol. 64, no. 10, 2017. [Online]. Available: https://par.nsf.gov/servlets/purl/10060772

[93] M. Kazhdan, J. Solomon, and M. Ben-Chen, "Can Mean-Curvature Flow be Modified to be Non-singular?" in *Computer Graphics Forum*, vol. 31, no. 5, 2012, pp. 1745–1754. [Online]. Available: https://doi.org/10.1111/j.1467-8659.2012.03179.x

[94] M. Robinson, "PySheaf: Sheaf-theoretic toolbox," 2017, GitHub repository. [Online]. Available: https://github.com/kb1dds/pysheaf

[95] A. Hylton, G. Henselman-Petrusek, J. Sang, and R. Short, "Tuning the Performance of a Computational Persistent Homology Package," *Software: Practice and*

Experience, vol. 49, no. 5, pp. 885–905, 2019. [Online]. Available: https://doi.org/10.1002/spe.2678

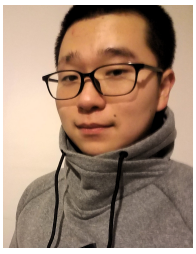
[96] S. S. Woo, J. L. Gao, and D. L. Mills, "Space Network Time Distribution and Synchronization Protocol Development for Mars Proximity Link," in *SpaceOps 2010 Conference: Delivering on the Dream*. AIAA, 11 2010, p. 2360. [Online]. Available: https://doi.org/10.2514/6.2010-2360

BIOGRAPHY

Alan Hylton *should probably be designing tube audio circuits, but instead directs Delay Tolerant Networking (DTN) research and development at the NASA Goddard Space Flight Center, where he is humbled to work with his powerful and multidisciplinary team. His formal education is in mathematics from Cleveland State University and Lehigh University, and he considers it his mission to advocate for students. Where possible, he creates venues for mathematicians to work on applied problems, who add an essential diversity to the group.*

Natalie Tsuei *is a current sophomore at American University. She studies computer science and international studies. Her concentrations are in cybersecurity, foreign policy, and national security. After completing her undergraduate degree, Natalie hopes to continue her studies at the graduate level.*

Mark Ronnenberg *is a Ph.D. candidate in mathematics at Indiana University, and plans to graduate in 2023. His research is in the field of low dimensional topology. Thanks to his time at NASA, he is now also interested in applications of pure mathematics to "real world" problems. Prior to coming to Indiana, Mark earned a BA and MA in mathematics from the University of Northern Iowa.*



Jihun Hwang *(Jimmy) is a second-year Ph.D. student in computer science at Purdue University. He is mainly interested in studying problems in cryptography through the lens of information theory, analysis of Boolean functions, and computational geometry. However, he ultimately defines himself as an aspiring theoretical computer scientist who likes to discuss any topics in or related to algorithms. Before Purdue, he studied mathematics and computer science at the University of Massachusetts Amherst.*

Brendan Mallery *Brendan is a second year Ph.D. student studying mathematics at Tuts University. His research interests include optimal transportation and the study of Markov chains. Prior to Tufts he obtained a Masters in mathematics at the University at Albany, SUNY and a bachelor's in chemistry and mathematics at Bowdoin College.*

Jonathan Quartin *is a Ph.D. candidate in mathematics at the University of Colorado at Boulder, and plans to graduate in 2023. His graduate work is in Algebraic Geometry, focusing on the geometry of stable map moduli spaces. After graduating, he intends to look for government/industry jobs in the LA area that involve applications of high-level mathematics. Before Boulder, he received a BA in mathematics from the University of California at Berkeley.*

Colin Levaunt *graduated with a Master of Science in Mathematics from the University of Vermont in 2022. And, though, intending to study pure mathematics, as a graduate student, he became more interested in the novel application of advanced mathematical techniques to understanding and solving complex multidisciplinary problems. During the summer of 2022, as an intern at NASA, he had the opportunity to work with a team researching the mathematical foundations of network architectures to enable the future Solar System Internet. He hopes to continue to apply his expertise to challenging and impactful projects.*

Jeremy Quail *is a third year Ph.D. student in mathematics at the University of Vermont. His research is in graph and matroid theory, with a current focus on positroids. Following his experience as a NASA intern in the Summer of 2022, he is interested in continuing to explore applications of high-level math. Jeremy received a BA in mathematics from Queens College and an MS in mathematical sciences from the University of Vermont.*

Justin Curry *is an Assistant Professor of Mathematics and Statistics at the University at Albany, SUNY. Before arriving at Albany in 2017, he was a Visiting Assistant Professor at Duke University. Professor Curry earned his PhD in mathematics from the University of Pennsylvania in 2014, under the direction of Robert Ghrist. His research interests include the use of category theory in applied mathematics, with particular emphasis on applied sheaf theory, and inverse problems in topological data analysis (TDA).*