The Distributed Computing Column

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

Clock synchronization is one of the fundamental problems in distributed computing, playing a critical role at one of the lowest levels of the protocol stack. As such, it is a basic building block that many higher level protocols depend on (and many algorithm designers take for granted). It is a long-studied problem, with foundational results going back decades and decades. And it is a basic service that is used by numerous real-world protocols (e.g., in the form of NTP).

In this column, Swen Jacobs and Christoph Lenzen give an overview of the state of the art, as well as argue that there remains a lot of work left to be done! They focus on the problem of robustness: most current synchronization protocols are not secure, and most are susceptible to fairly simple attacks. Protocols like NTP are well-known to be easy to attack, and can have severe consequences for real-world systems like power grids and financial networks. Jacobs and Lenzen discuss a variety of open problems in the area of (robust) clock synchronization, including issues of redundancy, network topology, and trusted computing. They also raise the question of how to determine whether the protocols actually work, i.e., do they do what they are supposed to? To this end, they discuss a variety of formal methods approaches for verifying clock synchronization protocols, including interactive proofs, automated verification, and partial automation. Moreover, formal methods that can prove properties related to robustness remain quite challenging!

Overall, this article provides an interesting overview of robust clock synchronization, and points toward a variety of interesting open questions in the area. I hope that you enjoy the column!

The Distributed Computing Column is particularly interested in contributions that propose interesting new directions and summarize important open problems in areas of interest. If you would like to write such a column, please contact me.

Current Challenges in Reliable and Secure Clock Synchronization

Swen Jacobs and Christoph Lenzen
CISPA Helmholtz Center for Information Security

1 The Task

This article discusses reliable and secure clock synchronization, as well as its verification, with a focus on real-world application scenarios and open problems. Synchronizing clocks in a network $G = (V, E)$ is a fundamental task that has been studied since the inception of the field. The goal of a synchronization algorithm is to perpetually compute a *logical clock* L_v at each participating network node $v \in V$. The optimization criteria might vary with application. In this article, we focus on the following common choices:

- the *global skew* $\mathcal{G} = \sup_t\{\mathcal{G}(t)\}$, where $\mathcal{G}(t) = \max_{v,w\in V} |L_v(t) - L_w(t)|$ between any pair of nodes in the network;
- the *local skew* $\mathcal{L} = \sup_t\{\mathcal{L}(t)\}$, where $\mathcal{L}(t) = \max_{\{v,w\}\in E} |L_v(t) - L_w(t)|$ between any pair of neighbors in the network;
- bounds on the logical clock rates, i.e., $1 \leq \frac{dL_v}{dt}(t) \leq \alpha$ for some $\alpha > 1$.[1] In particular, it is not permitted to simply set all logical clocks to 0 forever or running them at exponentially decreasing rates.

Minimizing skew is challenging due to the inherent uncertainties in the system. Each node is equipped with a *hardware clock* H_v that approximates real time with a rate error of at most $\vartheta - 1$, i.e., for $t' > t$ it holds that[2]

$$t' - t \leq H_v(t') - H_v(t) \leq \vartheta(t' - t).$$

In addition, communication delay cannot be known precisely. To account for this, we assume that messages are under way for at least $d-u$ and at most d time, where

[1] For notational convenience, we normalize the minimum rate to 1.

[2] The error is one-sided to simplify notation. Because $\vartheta - 1 \ll 1$, this corresponds to $(1-\rho)(t' - t) \leq H_v(t') - H_v(t) \leq (1+\rho)(t' - t)$ for $\rho \approx (\vartheta - 1)/2$.

d is the maximum end-to-end *delay* and u the delay *uncertainty;* we assume that $(\vartheta - 1)d < u$.[3] For the sake of simplicity, we disregard heterogeneous systems in which the quality of clocks or links differs in this article, and pretend that all logical clocks can be initialized perfectly, i.e., $L_v(0) = 0$ for all nodes v.

The global and local skew bounds that can be achieved within this model have been identified to be $\Theta(uD)$ and $\Theta(u \log_{\alpha/(\vartheta-1)} D)$, respectively [10, 55], where D is the diameter of the network G. In this article, we discuss the challenges that arise when the theory underlying these results and approaches by practitioners face a reality in which faults and attacks are the norm. As will become clear, this leads to a whole range of new open problems, which require not only the techniques from distributed computing, but also cryptography and formal verification to evolve.

2 Why Do Faults and Attacks Matter?

Access to accurate time, being a basic service, is a crucial building block in many systems. This includes critical infrastructure, meaning that poor reliability or susceptability to attacks is an immense risk on a societal scale. To make this concrete, we now discuss several such systems and how they rely on a shared notion of time.

2.1 The Power Grid

The economic damage from even fairly short power outages is massive [12, 73, 75, 78]. At the same time, the power grid depends on microsecond accuracy in synchronizing monitoring devices to correlate measurements well enough to function; with increasing reliance on renewable energy sources, this becomes more and more important [25]. A failure of or successful attack on the timebase used by the power grid could cause global failure of the system, after which recovery will take at least several hours, cf. [27]. Attacks are viable and have been performed, with the synchronization subsystem being a viable attack vector [81, 59].

Note that the power grid is, inherently, a highly distributed system. Hence, it is virtually impossible to ensure that an attacker can access none of its components at all. This means that techniques dealing with worst-case, i.e., Byzantine, faults play a key role in securing it against attacks. As a convenient side effect, these can also increase the resilience of the system against faults that are not caused by attempted sabotage.

[3]This assumption is typically satisfied. If not, one can simulate hardware clocks with (up to a constant) this quality by bouncing messages back and forth along network links.

2.2 Cellular and Broadcast Networks

Cellular and broadcast networks require synchronization between cells to combat interference [21]; errors as small as microseconds can bring down entire networks [8]. Also here, the economic stakes of failure are high. Moreover, arguably our dependence on these networks also in times of crisis could render them critical infrastructure as well. In contrast to the power grid, for which central processing of measurements requires a small global skew, minimizing interference is a matter of minimizing the local skew.

2.3 Synchronization via the Internet

A standard, if inaccurate, way of obtaining time is to ask the Internet. Typically, this is done by querying time servers via the Network Time Protocol (NTP) [35, 65]. While fairly inaccurate, this method is popular due to requiring no additional resources on the client side. Thus, (too) many systems and services are likely to depend directly or indirectly on this approach.

As we discuss later, NTP and similar services are vulnerable to several attacks. Arguably, this means that critical services should not rely on this synchronization method. However, the sheer volume of NTP users means that improving reliability, security, and accuracy of Internet synchronization is worthwhile. It also suggests that it is likely that some crucial services will nonetheless be subject to NTP-based attacks.

2.4 Financial Sector

Banks are required to obtain "traceable" time with accuracy of 100 microseconds or better [64]. As suggested by this standard, the advantage of responding quicker to new information, even by milliseconds, provides a distinct advantage in high-frequency trading, cf. [14]. Equivalently, obtaining a timestamp "from the past" when committing a transaction yields the same advantage. From a theoretic point of view, the solution is to switch to a discretized, round-based market, in which trade requests are resolved based on reception times of requests at stock exchanges. However, such a change would require regulatory oversight to step in [15], which could necessitate a joint international initiative. In the current system, there is an incentive for the bank and its employees to manipulate timestamps for the sake of profit. Since timestamping occurs *within the bank's system,* it is, in principle, trivial to do so. As previous large-scale instances of fraud and malpractice [9, 85] clearly demonstrate, it is ill-advised to let the fox guard the hen house.

Note that this setting requires to rethink the time infrastructure having in mind that the *user* takes the role of the potential attacker. In contrast to the other examples, here it is insufficient to make sure that time is available and correctly recorded only at trustworthy nodes. On the other hand, accuracy requirements in the microsecond range render it challenging to directly involve remote parties in the timestamping procedure.

3 State of the Art

With the stage being set, let us revisit the state of the art and its limitations. As the examples in the previous section demonstrate in abundance, reliability and security are crucial in the wild, so this discussion will focus on these aspects.

3.1 Estimating Clock Offsets in Networks

Deployed solutions. This task is the larger part of what the Network Time Protocol (NTP) [36, 35] and the Precision Time Protocol (PTP) [71] seek to accomplish. The basic protocols are not concerned with security beyond message authentication. This is insufficient to cope with any mildly determined attacker, since verifying the content of a message does nothing to ensure its timing. In absence of bounds on communication delay, a man-in-the-middle attacker can arbitrarily and undetectably shift perceived relative time without needing to alter the content of messages [66]. There are works on improving resilience to some attacks for NTP (e.g. [41, 74]) and the possibility of using redundancy to increase the resilience of PTP has been considered as well [69]. However, what all of these works appear to have in common is that they consider the network to be given, whereas routing is either not considered or fixed by selecting a tree. Under these conditions, no substantial guarantees in face of faults and attacks are possible, and hence at best generic and vague statements are offered in this regard.

All-pairs estimation. From a theoretical angle, little has been published on the topic either. Of course one might endeavor to simulate full connectivity, as done in [29]. However, this approach has substantial drawbacks. In order to achieve any significant degree of resilience against an attacker taking control of network nodes or links, one must avoid that too many paths share the same edge or internal node. Apart from being difficult to realize in practical networks, this also means that one might be forced to prefer some longer paths to avoid relying too much on few nodes and edges. In turn, this can hurt the quality of measurements, as such longer paths will have increased cumulated delay uncertainty. This suggests non-trivial trade-offs between resilience and accuracy that so far have not been studied.

What is more, algorithms designed for a known given topology or specific classes of networks might perform better by not relying on offset measurements between all pairs of nodes. In other words, simulating full connectivity might harm robustness, security, or accuracy compared to topology-aware protocols. This conjecture is corroborated by some prior work that aims at achieving fault-tolerant synchronization in very sparse networks [16, 23].

Message authentication. Cryptographic authentication needs to play a major role for synchronization in networks. For many distributed tasks, such as consensus, it can be leveraged to increase resilience. This is also true for synchronizing clocks in fully connected systems [2]. Recall that d is the maximum end-to-end communication delay and u its uncertainty, i.e., the time between commencing message transmission and the receiver completing to process is between $d-u$ and d. It has been shown that the above resilience boost is also possible with asymptotically optimal skew of $O(u)$ [56]. However, there is a catch: this imposes the additional constraint that $d - u$ is a lower bound on the end-to-end delay on links with one *faulty* endpoint. Intuitively, this follows from the need to use indirect communication for authentication to be of value, together with the fact that faulty nodes cannot be detected reliably. This entails that malicious nodes must not get access to honest nodes' messages prematurely or deliver their own much faster than honest nodes; otherwise, they can enforce a decreased precision of time offset measurements, which in turn decreases the accuracy of synchronization that can be achieved. In contrast, the classic algorithm achieving the same asymptotic skew bound of $O(u)$ without authentication (and hence smaller resilience) [84] merely requires that the end-to-end delay bounds are satisfied on links between correct nodes.

Since any lower bound for a complete network extends to arbitrary networks, this points at potential fundamental obstacles to employing authentication for increased resilience, i.e., reducing requirements on network connectivity. In particular, one must avoid that an attacker can bypass the network or otherwise significantly speed up delivery of messages to or from nodes it controls.

3.2 Algorithms for Incomplete Networks

For complete networks, the asymptotically optimal skew of $O(u)$ can be achieved under optimal resilience to Byzantine (i.e., worst-case) faults [10, 24, 56, 84]. It is known that these guarantees can also be achieved in a self-stabilizing [22] manner with small stabilization time [42, 57, 58]. Similarly, one can simultaneously achieve asymptotically optimal skew between arbitrary pairs of nodes as well as neighbors [10, 55]. These results extend to crash faults in a straightforward way, and the algorithms can also be made self-stabilizing in asymptotically optimal time [54, Sec. 12.3]. In contrast, little is known about tolerance of non-benign faults or attacks in general networks.

Network augmentation. In [16], it is shown that one can augment a given network by replacing each node with a cluster of $\Theta(f)$ nodes and increasing to number of edges by factor $\Theta(f^2)$ to simulate the algorithm from [55] in a way tolerating up to f faults in each cluster. This is resource-efficient in the sense that if the original network was just barely connected, this overhead is necessary to handle f faults in each cluster. However, networks that are already connected well might need far fewer additional resources to enable us to achieve small skews in spite of faults. It should also be stressed that one might need even fewer resources when allowing for the possibility that also a small fraction of the nodes that faithfully execute the algorithm loses synchronization to the rest of the network. Note that this is not merely a question of connectivity, but also of how well-suited the available redundant paths are for synchronization, i.e., how long they are in the distance metric induced by communication delay uncertainties. Accordingly, so far none of these issues have been addressed in the literature.

3.3 Single Points of Failure in Deployed Systems

Radio communication. Global Navigation Satellite Systems (GNSS) are, wherever available, the most convenient source of accurate time. From a global point of view, the provider of such a system constitutes a single point of failure and can manipulate the time perceived by all receivers. A simple, very cost-effective method to reduce this risk is to rely on multiple GNSS services concurrently, selecting the median time value as reference. This means that a single bad actor cannot single-handedly change the perceived time. However, given the rather short list of available systems, the potential for collusion, and the need to receive a large number of satellite signals to implement this solution, it is not universally applicable.

Moreover, on the user's side, GNSS are fairly easy to jam or, by a more sophisticated attacker, to spoof or subject to delay attacks [82]. The use of multiple systems does not protect from such attacks, necessitating different means of obtaining or verifying the time.

Similar considerations apply to the terrestrial alternative of relying on long wave radio communication (e.g. DCF77 in Germany [31]), which furthermore is much less accurate than GNSS.

National Metrology Institutes (NMIs). Obtaining the time via the phone network or possibly a dedicated link from an NMI is another option to obtain fairly accurate time. However, relying exclusively on a single such reference again renders it a single point of failure. It is worth to note that, despite numerous checks and involvement of experts, there is no official standardization or documentation of the procedure NMIs use to obtain and adjust their time, and human involvement is no protection from bad actors. In fact, it is very plausible that, from a suitable position within an NMI, a single person could meddle with the time of everyone trusting in this NMI's time. In addition, also here an attacker might manipulate the time of a recipient or a group of recipients by delaying time messages from the NMI, without the need for altering these messages.

Lack of standardized redundancy. One might think that, obviously, this would prompt standardization of the use of multiple time references and/or means of obtaining time, with the goal to improve the reliability and security of deployed solutions. Unfortunately, nothing could be further from the truth. The examples with *best* behavior are possibly the most prominent network synchronization protocols, the Precision Time Protocol (PTP) and the Network Time Protocol (NTP), when taking into account additional literature. Regarding PTP, [69] briefly mentions the *possibility* to use redundant time references and routing paths to increase reliability and security, without any further discussion on requirements or best practices for doing so. Concerning NTP, Chronos makes an effort to leverage the availability of multiple time servers to increase resilience [74], but is called out for neglecting DNS-based attacks by [41]. Arguably, these works are efforts to plug individual holes in a very leaky bucket, since they focus on defending against certain attacks against a system and protocol that were not designed based on security considerations.

The user. As pointed out in the context of stock trades, it is possible that the user might want to manipulate their local time. We conclude that there is need for solving the task of forcing selfish or malicious parties to consistently report timing of operations. For this purpose, one option of interest is to use trusted computing technology, such as Trusted Platform Modules (TPM) [83] or Intel Software Guard Extensions (SGX) [3], at the client to prevent or at least substantially hamper abuse. These approaches establish trust anchors in hardware that should provide confidentiality and integrity of crucial operations, even in light of powerful attackers that can control the entire software stack or even have physical access to the system. However, these techniques cannot provide a trusted, reliable clock by themselves. Solutions like Intel SGX must either build on shared hardware resources between isolation domains, resulting in dependence on a potentially corrupted platform clock (e.g., if the attacker manipulates voltage and frequency scaling), or rely on higher privileged, untrusted software that can arbitrarily delay operations [3]. Even distinct coprocessors such as a TPM, which feature an internal clock, are no reliable time reference. A TPM depends on a properly managed platform, implying that an attacker with privileges on the system can set the TPM clock arbitrarily into the future, or make the TPM clock run 32.5% fast or slow [83]. In [6], the authors seek to secure the clock against manipulation, but their threat model does not account for attacks on the execution speed (e.g., by manipulating supply voltage or temperature), and the clock still has no guaranteed relation to UTC.

4 Does it *Actually* Work?

The development of clock-synchronization algorithms, as well as their implementation, are challenging and complex tasks. With this complexity comes the probability of human error. While designing algorithms, researchers will usually create hand-written proofs that supposedly witness correctness of their algorithms, but may contain errors. But even assuming correctness of these hand-written proofs, they only give us theoretical guarantees that not necessarily carry over to their real-world implementations. Therefore, it is crucial that the correctness of algorithms and their implementations is fully formalized and mechanically verified.

To obtain machine-checked formal proofs, there is a range of possibilities depending on the desired level of automation: *interactive* theorem provers allow us to formalize a very large range of systems and prove that they satisfy expressive formal specifications, but they require expert users and a large manual effort. In contrast, *automated* verification techniques alleviate the burden on the human designer, but are often restricted to certain types of systems, certain levels of abstraction, and certain properties to be proved. Between these two extremes there are all kinds of intermediate solutions, be it partial automation in interactive proofs, or manual efforts to massage a given system and specification until it fits into the fragment that is supported by an automatic method. In the following, we will first consider interactive tools, then fully automated ones, and finally techniques that try to get the best of both worlds by combining human guidance with automation. Regardless of the level of automation, we are interested in verification approaches that

1. can provide guarantees that hold regardless of the size of the network (called *parameterized verification* techniques)
2. have native support for faults and attacks, and
3. support real-time systems and properties.

While there is a lot of work on these aspects separately, or a combination of two of them (in particular on the verificiation of agreement protocols without real-time properties), will see that there are few existing approaches that support all of these features, and those that do exist are usually very restricted in some other aspect.

Interactive Mechanized Proofs. A first step towards formal correctness guarantees is a precise and testable specification of the intended functionality. In contrast to conventional design documents that contain prose or pseudocode without a testable semantics, a formal specification is precise, and this precision helps to eliminate ambiguities and clarify intention. Moreover, the formal specification can be gradually refined into an implementation, and can be checked for errors at any time during this process.

To support our intended applications, a tool for formalization needs to cover complex features, including concurrency, fault-tolerance, and time. A specialized formal language that already covers a lot of this is TLA$^+$ [52, 20, 51], which can be used to formally describe the set of all legal behaviors of a system, and is essentially based on set theory and temporal logic. While TLA$^+$ started as an academic tool, it has now also been used to support large-scale system development in industry, for example at Amazon Web Services [67, 68]. TLA$^+$ directly supports refinement of abstract specifications into more and more concrete algorithms and implementations. Correctness of an algorithm or an implementation then can either be shown by interactive mathematical reasoning in the TLA$^+$ proof system (TLAPS) [18], or by discharging certain types of proof obligations to SMT solvers [63] or model checkers [88, 46]. While TLA$^+$ does not explicitly support real-time properties, it has been argued that they can easily be modeled within the existing language, but automatically discharging the resulting proof obligations is a major challenge [53]. A number of languages and techniques have been inspired by TLA$^+$ and have similar strengths and weaknesses, for example the Ironfleet [76, 34] framework.

In addition to these specialized languages, more general interactive theorem provers have been used to reason about distributed systems. In particular, the PVS system has been used to verify certain textbook clock synchronization algorithms [77]. Moreover, Coq has been used as the basis of the Verdi framework for implementing and verifying distributed systems [86]. The focus of Verdi is on the use of verified system transformers, which simplify the task of reasoning about the correctness of refinements on the way from high-level specifications to low-level implementations. Like TLA$^+$ however, real-time properties are not directly supported. Another interactive theorem prover that has been used to develop specifications and proving correctness of implementations is HOL4 [79]. It has been used to verify low-level aspects like the network stack [11] or message queues [72], but it would take significant effort to scale such efforts to more complex systems.

Automated Verification. Automated verification techniques for distributed systems can be separated into those that give rigorous guarantees for systems with a parametric number of components, and those that under-approximate the possible behaviors of a distributed system by only considering a fixed number of components.

One of the most prominent tools is Alloy [39], which is not specialized to distributed systems, but allows to model infinite-state software systems in general. Alloy uses a *bounded* verification approach that makes verification decidable and can find many bugs, but does not give reliable correctness guarantees in general. Another tool with a similar behavior is MoDIST [87]: given the code of a node in a distributed system, it instantiates it for a fixed number of processes and uses a model checking approach to check safety and liveness properties. Thus, errors that only manifest in systems with many processes will not be found, even if the search space for the chosen number of processes is explored exhaustively. However, MoDIST has two notable features that many other approaches lack: first, it works directly on unmodified executable code, simulating the OS and the network, including failures such as message reordering and machine crashes. Second, it includes some support for real-time properties, in particular timeouts, by providing a *virtual clock* mechanism that approximates the behavior of a real clock, restricting its analysis to certain parts of the code and to simple comparisons against certain program variables.

A slightly different support for obtaining correct distributed systems is provided by Mace [43], which allows the designer to specify a distributed system in a restricted and structured domain-specific language, model check this high-level specification (with a special focus on liveness properties [44]), and compile it to a C++ implementation that inherits the desired properties and includes code for failure detection and handling. Like Alloy and MoDIST, model checking is restricted to a fixed number of processes.

Another completely automated approach is implemented in MCMT, a model checker modulo theories [30]. The idea is to model distributed systems as infinite-state systems whose state variables are arrays (of unbounded length), and use SMT solvers to compute reachable sets of states. Since this reasoning naturally involves quantification, which is in general not supported in a complete way by SMT solvers, the technique relies on heuristics for quantifier instantiation that are tailored for the use case of model checking. The approach assumes that the system is given as an array-based transition system and does not provide support for automatic translation from executable code. On the other hand, it has been used to reason about distributed systems with a real-time component, such as different versions of the Fischer protocol [17]. An extension of the MCMT approach that expands its applicability and also includes an integration with a deductive verification framework has recently been introduced [19].

Finally, there are *parameterized model checking* techniques that are often restricted to a system model with certain types of communication or synchronization, but within such a fragment provide a decision procedure for properties that hold regardless of the number of components [13, 37, 38]. However, most of the existing results in this direction do not support strong attacker or fault models.

An exception is ByMC, the Byzantine Model Checker [47]: it is based on *threshold automata* that can model distributed protocols that count messages and make progress when a certain number of messages (the *threshold*) has been received, and it supports Byzantine faults in a number of nodes that is defined relative to the threshold. ByMC supports parametric verification of safety and liveness properties, i.e., a violation of the properties will be found regardless of how many processes are needed to exhibit the error, and if no violations are found, the system is provably correct for any number of processes [48]. While ByMC supports strong attackers, like most parameterized model checking approaches it does not support real-time properties.

There is a line of research in parameterized model checking that is able to give timing guarantees by modeling processes as timed automata [1], but in turn it does not support strong attacker models. However, an approach based on timed automata has recently been used to model and verify a basic gossiping clock synchronization protocol [80] (with explicit modeling of possible faulty behavior). In addition, there are approaches that support the verification of symbolic time bounds [40], but it is unclear if these can be extended to the verification of clock synchronization protocols.

Techniques with Partial Automation. Above, we mentioned the MoDist tool, which verifies an abstract algorithm and compiles it to an implementation that is guaranteed to inherit the desired properties. A more intricate variant of this approach is taken by the Civl verification framework [49, 50], which is based on an approach called layered refinement. The basic idea is that a proof of correctness does not relate an implementation to a specification in a single step, but in several refinement steps that abstract away details from the implementation, while preserving the properties that are necessary to prove the specification. In this case, the developer is responsible for designing the different refinement layers, while correctness of the refinements can then be fully automated (for a fixed number of processes). A similar approach is taken by the Armada language and tool [60], which additionally allows the developer to extend the library of proof strategies, potentially enabling the verification of a larger range of programs.

Another important recent development that merges the interactive and automated approaches is Ivy [62], a "multi-modal" verification tool that allows interactive TLA$^+$-style proofs, but also has a strong focus on automated verification in decidable fragments, which ensures its predictability and stability against small perturbations in the input.

In some of the approaches mentioned before, the hard part of verification is the identification of an inductive invariant that implies the desired safety property. The I4 approach [61] aims at automating the search for inductive invariants by efficiently identifying invariants on small instances of the system (based on symbolic model checking), and generalizing them to invariants that hold for the protocol in general, regardless of the number of processes. After generalization, the existing Ivy tool is used to check correctness of the invariant. Generalization itself is based on a number of heuristics that have proven fruitful for some applications, but may have to be extended for other cases. An extension of the approach with a special form of predicate abstraction (called syntax-guided abstraction-refinement) and word-level reasoning is implemented in the AVR tool [32]. This approach enables the verification of more complex systems, because it can automatically abstract from the domain complexity that is outside of the considered problem, for example by concentrating on control-flow details while abstracting from the processed data.

A variation of this approach is also implemented in SWISS [33]: instead of identifying invariants by model checking a small instance of the protocol, the approach relies on restrictions of the invariant itself. The idea is that, since protocols are designed by humans that have a correctness argument in mind, their correctness must be provable based on a relatively simple invariant. Therefore, search is restricted to a well-defined finite set of candidate invariants. The obvious drawback is that this restriction may be too strong, and a suitable invariant may not be found even if it exists.

Handling Faults. While some of the approaches above handle faults explicitly, most of them do not. If such an approach is used to verify distributed systems, there are two main ways to obtain correctness guarantees also in the presence of faults: either by proving that the program, as is, is fault-tolerant, or by *making* it fault-tolerant. An example of the first approach uses the regular model checking framework for verifying parameterized systems, together with a formal fault model, to completely automate verification in the presence of faults [28]. For the second approach, there exist approaches that use synthesis techniques to automatically modify existing protocols in order to ensure fault-tolerance, for example implemented in the FTSyn tool [26]. These tools support different types of faults and combinations of faults.

5 A Wish List

From the discussion above, we derive a number of specific challenges of interest to be tackled. They fall broadly in two categories: understanding clock synchronization under faults or attacks, and suitable tools for their verification. Naturally, a third crucial challenge is to actually apply these techniques in the context of the problem areas listed earlier, implementing, verifying, and deploying algorithms in the wild.

5.1 Open Problems for Synchronization under Faults

Estimating clock offsets in incomplete networks. The most basic ingredient of clock synchronization algorithms is a subroutine for estimating clock offsets between the nodes of the network. Between neighbors, this is simply done by direct communication. If algorithms need to compute such estimates between non-neighboring nodes, the best (worst-case) accuracy is achieved via communication along the shortest path with respect to edge weights given by the measurement error induced by communicating along each edge. However, when faults or corruption of internal nodes or edges on the communication path become a possibility, redundant use of (node or edge) disjoint paths can increase resilience. Unfortunately, the additional paths might provide less accurate measurements. Accordingly, we need to understand how to achieve the best possible tradeoff between accuracy and resilience. This prompts a large number of specific research questions:

- Given a fixed set of k disjoint paths between two network nodes and a target resilience f, suppose that for each path we know a worst-case bound on the accuracy when measuring the offset between the two nodes' clocks using this path, provided it is fault-free. What is the best worst-case accuracy that can be guaranteed when computing an estimate based on taking measurements from all k paths? Is there a strategy that is concurrently optimal for all values of f? If not, what are the trade-offs? Note that this problem can be used to model a client seeking to robustly obtain accurate time when multiple time servers with known communication paths of non-uniform accuracy are available, by introducing a virtual node with perfect clock. A generalized version considers the setting when not all paths are disjoint.

- Given a network with the above edge weights and a pair of nodes, how to best select k paths for a given target resilience f? Is there a uniform strategy that is good for all values of f?

- How do the answers to the above questions change if we consider multiple pairs of nodes, but allow for estimation to fail for some pairs (possibly as a function of f) entirely? A particularly important special case is the all-pairs setting, since this corresponds to simulating a complete network for the purpose of synchronization. Combining the outcome with analyses of the resilience of algorithms for complete networks to edge failures could yield improvements over the state of the art.

- Given a network and a budget for adding edges or nodes, how much can we improve the suitability of the network for the above tasks? Here, edge cost may vary depending on the uncertainty that should be guaranteed, this uncertainty could be a function of the physical network topology, or a mixture of both.
- Taking this one step further, what happens if we get to design the network from scratch? Which topologies are most suitable for the above tasks?

Damage mitigation. If faults or an attacker cannot be entirely prevented from causing a disruption, it is important to limit the impact on the functionality of the system. This is the underlying philosophy of guaranteeing synchronization at all non-faulty nodes or all but a small fraction of the non-faulty nodes. However, there are further options serving this purpose.

One such option that has been neglected in theoretical work on the subject is to harness the local clocks of nodes more effectively to mitigate the effects of temporary disruptions. If one (re-)designs algorithms to adjust the computed local output clocks only at small rates comparable to their inherent drift from UTC, even a prolonged attack temporarily compromising a majority of the network has limited impact on the time at uncompromised nodes. This can dramatically raise the cost of an attacker trying to achieve a network-wide disruption, and it could entirely prevent some faults (like the bug from [21]) from affecting the functionality of the system. Concretely, cesium standards – a type of atomic clock – are off by no more than a few nanoseconds per day. If an algorithm limits its corrections to the point where it amplifies this "natural" drift to no more than 10 nanoseconds per day, an attack would need to be maintained for several *months* to induce an error of a single microsecond. In the meantime, the attack can be detected and repelled, without causing any disruption to applications requiring microsecond (or worse) accuracy that obtain their time from a non-corrupted node.

Next, instead of trying to maintain synchronization at all times, an important and well-known complementary approach is to ensure automatic recovery from disruptions. Taking this approach to the extreme, one may ask for self-stabilization, i.e., automatic recovery to a consistent system state after arbitrary transient faults [22].

If this is too costly or makes the system more vulnerable in other regards, one might settle for automatic recovery under additional assumptions. For example, if the consequences of global system failure are so dramatic that human intervention will be triggered anyway and the potentially faster recovery due to self-stabilization provides no relevant advantage, it is justified to assume that the majority of the nodes remain synchronized. This, in turn, greatly simplifies recovery for nodes that undergo transient faults, as they merely need to resynchronize to the majority, cf. [45].

A promising candidate assumption is that even after the disruption, the synchronization error with respect to the reference time, say UTC, is bounded by some value B (which is unknown and possibly much larger than the error bound under nominal conditions). This is a natural outcome of leveraging local clocks to mitigate the impact of faults and attacks as described above. We then ask that the system converges back to nominal synchronization quality as soon as possible. Note that this results in a very appealing synergy: by bounding the impact of an attack of duration T on synchronization quality by $O((\vartheta - 1)T)$, recovery then might be possible within $O((\vartheta - 1)T)$ time, while simultaneously maintaining that the output clocks drift at rate $O(\vartheta - 1)$ relative to UTC. In other words, in addition to mitigating the impact of attacks and faults, we get the desirable property that the output clocks maintain rates that are close to 1 for free!

Gradient clock synchronization. Gradient clock synchronization (GCS) seeks, in addition to minimizing the worst-case skew between arbitrary pairs of nodes in the network, to minimize the worst-case skew between network neighbors. The latter can be kept as small as $\Theta(u \log D)$ [55], where D is the network diameter; this an exponentially smaller dependence on D than can be achieved for arbitrary pairs [10]. This makes GCS promising for settings where the skew between neighbors is what really matters. For example, when synchronizing broadcast transmitters or mobile phone cells, the goal is to avoid interference between close-by cells [21].

Accordingly, reliable GCS synchronization algorithms are of interest. Rather than requiring to add redundancy to the existing network as suggested in [16], one could exploit already existing redundancy in specific networks. For example, if we consider the power graph of a grid graph, i.e., connecting node (i, j) to all nodes (i', j') with $\max\{|i' - i|, |j' - j|\} \leq 1$, it is plausible that a variant of the algorithm from [55] could be devised that tolerates one fault in each neighborhood. If this is successful, it seems likely that the result can be generalized to similar graphs; also note that having the above graph as subgraph of the communication network is sufficient for running such an algorithm.

Moreover, the GCS algorithm from [55] allows us to bound the rate at which the output clocks can be adjusted by $O(\vartheta - 1)$. Therefore, it naturally lends itself to the damage mitigation mechanism outlined above. However, since the skew bound between neighbors is comparatively small and network cells will not all be equipped with cesium standards, the time for responding to an attack is smaller. Here, it is of interest to consider the following game. Suppose an attacker corrupts one or a several nodes in order to disrupt synchronization for additional nodes. Moreover, assume that (non-corrupted) nodes repeatedly and securely report the clock offsets they observe to their neighbors to a central authority,[4] and that the

[4]To keep in line with the goal of avoiding single points of failure, this "central" authority might

central authority can securely issue the command to logically delete links from the network. Is there a strategy that the authority can employ to prevent the attacker from causing disruption on a larger scale? Note that this is a difficult question, as an attacker could also attempt to coax the central authority into disconnecting other nodes, and the goal is to minimize the number of impacted nodes. Moreover, it is of interest to consider the same question for mobile attackers, which try to evade capture while seeking to disrupt synchronization. If there are good solutions to these tasks, one Where feasible, we will try to achieve this gold standard. can achieve accurate and reliable synchronization between neighbors in practical settings without adding substantial redundancy to existing networks.

Trusted timestamping. Another area of study are the accuracy and security guarantees that can be established for a trusted hardware providing a timestamping service that relates to UTC. Recall that existing solutions offer no such guarantee, since they do not rely on communication with devices outside of the control of the attacker. We argue that it is necessary to involve bidirectional communication with external parties to establish the veracity of any claims about timing. Otherwise, such an approach is doomed to fail due to an attacker having full control of *when* timing information reaches trusted hardware components.

Clearly, this necessitates to rely on cryptographic authentication techniques. However, this does not necessitate to trust a central server or authority, since time can be maintained collaboratively. Thus, this task blends in naturally with questions about secure distributed synchronization primitives.

5.2 Open Problems in Verification of Synchronization Protocols

As detailed before, there has been a lot of research into the verification of distributed protocols, including in particular agreement protocols. However, most of the techniques only support one or two of the three crucial features needed for the verification of practical clock synchronization protocols (parameterized verification, faults/attacks, real-time properties). Since the state of the art is rather far from being able to solve these problems we present, even partial solutions would be very much appreciated, e.g., proving *some* of the desired properties. Let us recapitulate some promising research directions, and detail the open problems they are connected to.

Automatic Methods for Clock Synchronization Protocols. There have been a number of efforts to verify clock synchronization protocols or other distributed protocols with real-time constraints, but these have been rather limited: As a completely automatic approach, Spalazzi and Spegni [80] have introduced a method

also be implemented via consensus of several monitors.

for parameterized model checking of gossiping clock synchronization protocols. This is a good starting point, but their model is limited to a certain form of gossiping communication, and has other strong restrictions that make it difficult or impossible to express practical protocols. Therefore, we believe that research into stronger fragments that still allow completely automatic forms of verification will be a fruitful research direction. Independent dimensions in which these results can be extended are (i) lifting restrictions within their framework of gossiping protocols, (ii) extending their approach to models with other communication/synchronization primitives, and (iii) extending their approach to support stronger fault models and (symbolic) cryptographic primitives.

Other Automatic Verification Methods. In addition, there are lots of completely automatic verification methods that give impressive results for certain classes of distributed protocols [47, 30], but to the best of our knowledge none of them supports parameterized verification with real-time constraints and strong fault models. Moreover, very few of them (e.g. [4]) support complex network topologies, which we have identified as being necessary for optimal robustness, security and accuracy in Section 3.1. Thus, while these techniques are certainly also worthwhile as a basis for the verification of clock synchronization protocols, on a superficial level the gap to supporting them is even bigger.

Semi-Automatic Methods. While the completely automatic methods above strictly need to be extended to handle clock synchronization protocols, the situation is a bit different with semi-automatic methods: many of them could, at least in theory, be used as they are to verify real-time constraints under strong fault models, as long as the user encodes the system and its desired properties in a specific way, and guides verification according to the capabilities of the underlying verification method. However, it is not difficult to imagine that this is a sub-optimal approach that puts most of the burden on the user, and is unlikely to succeed for any but the smallest examples. Therefore, such approaches also need specialized extensions to directly support the specification and verification of real-time constraints and strong fault models.

For example, in approaches based on TLA$^+$ [52, 51], both strong faults and real-time constraints can in theory be supported, but making its handling efficient and making verification scale to interesting protocols (or even implementations) is a challenging task. We conjecture that the addition of at least some dedicated support for such features, both in specification and in automated handling of verification conditions, would already make a large difference.

Verification of Implementations. While there are existing approaches that can formally verify abstract clock synchronization protocols (under strong restrictions), and there are approaches for the verification of *implementations of* distributed protocols (instead of high-level algorithms), we are not aware of any approach that combines both features, verifying actual implementations of clock synchronization protocols.

Some of the techniques mentioned in Section 4 work directly on implementations (e.g., in C code), others only verify the properties of a high-level model that may be implemented in different ways, making it necessary to prove that the desired properties are maintained by the implementation. Thus, even if we extend the existing techniques to support real-time properties and strong attackers or faults, we may have to additionally verify that implementations faithfully realize the respective abstract algorithms and inherit the desired properties. Some of the semi-automatic techniques support this naturally, for example the "layered" verification approaches based on TLA$^+$, or the layered refinement in Civl [49] and Armada [60]. Techniques like that, if extended to support the necessary features, could also be used to solve this problem for approaches that only verify the correctness of high-level algorithms.

For our use case, such approaches might benefit from research in adjacent areas that face similar problems, like the verification of implementations of security protocols [7], or of controllers for cyber-physical systems [70, 5]. In particular the latter area should provide important insights, as their high-level models have real-valued variables for time and other physical quantities, while implementations are usually restricted to some finite-precision abstractions of the real numbers. We face the same problem in implementations of clock synchronization algorithms, or in other distributed protocols with real-time constraints, for that matter.

References

[1] Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes (extended abstract). In *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 1998.

[2] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience. In *Financial Cryptography and Data Security (FC)*, pages 320–334, 2019.

[3] Fritz Alder, N. Asokan, Arseny Kurnikov, Andrew Paverd, and Michael Steiner. S-FaaS: Trustworthy and Accountable Function-as-a-Service Using Intel SGX. In *Cloud Computing Security Workshop (CCSW)*, pages 185–199, 2019.

[4] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.

[5] Adolfo Anta, Rupak Majumdar, Indranil Saha, and Paulo Tabuada. Automatic verification of control system implementations. In *Proceedings of the Tenth ACM Inter-*

national Conference on Embedded Software, EMSOFT '10, page 9–18, New York, NY, USA, 2010. Association for Computing Machinery.

[6] Fatima M. Anwar, Luis Garcia, Xi Han, and Mani Srivastava. Securing Time in Untrusted Operating Systems with TimeSeal. In *Real-Time Systems Symposium (RTSS)*, pages 80–92, 2019.

[7] Matteo Avalle, Alfredo Pironti, and Riccardo Sisto. Formal verification of security protocol implementations: a survey. *Formal Aspects Comput.*, 26(1):99–123, 2014.

[8] Chris Baraniuk. UK Radio Disturbance Caused by Satellite Network Bug, 2016. `https://www.bbc.com/news/technology-35463347`.

[9] Victor A. Beker. *The American Financial Crisis*, pages 45–59. Springer International Publishing, 2016.

[10] Saâd Biaz and Jennifer L. Welch. Closed Form Bounds for Clock Synchronization under Simple Uncertainty Assumptions. *Information Processing Letters (IPL)*, 80(3):151—157, 2001.

[11] Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations. In *POPL*, pages 55–66. ACM, 2006.

[12] The Cost of Blackouts in Europe, 2016. `https://cordis.europa.eu/article/id/126674-the-cost-of-blackouts-in-europe`.

[13] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

[14] Eric Budish, Peter Cramton, and John Shim. The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response. *The Quarterly Journal of Economics (Q J Econ)*, 130(4):1547–1621, 2015.

[15] Eric Budish, Robin S. Lee, and John J. Shim. A Theory of Stock Exchange Competition and Innovation: Will the Market Fix the Market? *SSRN*, 2019.

[16] Johannes Bund, Christoph Lenzen, and Will Rosenbaum. Fault Tolerant Gradient Clock Synchronization. In *Symposium on Principles of Distributed Computing (PODC)*, pages 357–365, 2019.

[17] Alessandro Carioni, Silvio Ghilardi, and Silvio Ranise. MCMT in the land of parametrized timed automata. In *VERIFY@IJCAR*, volume 3 of *EPiC Series in Computing*, pages 47–64. EasyChair, 2010.

[18] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. The tla$^+$ proof system: Building a heterogeneous verification platform. In *ICTAC*, volume 6255 of *Lecture Notes in Computer Science*, page 44. Springer, 2010.

[19] Sylvain Conchon and Mattias Roux. Reasoning about universal cubes in MCMT. In *ICFEM*, volume 11852 of *Lecture Notes in Computer Science*, pages 270–285. Springer, 2019.

[20] Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. TLA + proofs. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 147–154. Springer, 2012.

[21] Magnus Danielson. GPS Incident on Broadcast Networks. Technical report, U.S. Civil GPS Service Interface Committee (CGSIC), 2016. `https://rubidium.se/~magnus/papers/GPSincidentA6.pdf`.

[22] Edsger W. Dijkstra. Self-Stabilizing Systems in Spite of Distributed Control. *Communications of the ACM*, 17(11):643–644, 1974.

[23] Danny Dolev, Matthias Függer, Christoph Lenzen, Martin Perner, and Ulrich Schmid. HEX: Scaling Honeycombs is Easier than Scaling Clock Trees. *Journal of Computer and System Sciences (JCSS)*, 82(5):929–956, 2016.

[24] Danny Dolev, Joe Halpern, and H. Raymond Strong. On the Possibility and Impossibility of Achieving Clock Synchronization. In *Symposium on Theory of Computing (STOC)*, pages 504–511, 1984.

[25] Stephen Dominiak and Ulrich Dersch. Precise Time Synchronization of Phasor Measurement Units with Broadband Power Line Communications. Technical report, Swiss Federal Office of Energy SFOE, 2017.

[26] Ali Ebnenasir, Sandeep S. Kulkarni, and Anish Arora. Ftsyn: a framework for automatic synthesis of fault-tolerance. *Int. J. Softw. Tools Technol. Transf.*, 10(5):455–471, 2008.

[27] J.W. Feltes and Carlos Grande-Moran. Black Start Studies for System Restoration. In *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, pages 1–8, 2008.

[28] Dana Fisman, Orna Kupferman, and Yoad Lustig. On verifying fault tolerance of distributed protocols. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 315–331. Springer, 2008.

[29] Marc Frei, Jonghoon Kwon, Seyedali Tabaeiaghdaei, Marc Wyss, Christoph Lenzen, and Adrian Perrig. G-SINC: Global Synchronization Infrastructure for Network Clocks, 2022.

[30] Silvio Ghilardi and Silvio Ranise. MCMT: A model checker modulo theories. In *IJCAR*, volume 6173 of *Lecture Notes in Computer Science*, pages 22–29. Springer, 2010.

[31] Peter Glatzel. Timewarp DCF77-Testgenerator. *Elrad*, 2:88–91, 1996.

[32] Aman Goel and Karem A. Sakallah. AVR: abstractly verifying reachability. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 413–422. Springer, 2020.

[33] Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno. Finding invariants of distributed systems: It's a small (enough) world after all. In *NSDI*, pages 115–131. USENIX Association, 2021.

[34] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. Ironfleet: proving safety and liveness of practical distributed systems. *Commun. ACM*, 60(7):83–92, 2017.

[35] Internet Engineering Task Force (IETF). Network Time Protocol Version 4: Protocol and Algorithms Specification, 2010. `https://datatracker.ietf.org/doc/html/rfc5905`.

[36] Internet Engineering Task Force (IETF). Message Authentication Code for the Network Time Protocol, 2019. `https://datatracker.ietf.org/doc/html/rfc8573`.

[37] Nouraldin Jaber, Swen Jacobs, Christopher Wagner, Milind Kulkarni, and Roopsha Samanta. Parameterized verification of systems with global synchronization and guards. In *CAV (1)*, volume 12224 of *Lecture Notes in Computer Science*, pages 299–323. Springer, 2020.

[38] Nouraldin Jaber, Christopher Wagner, Swen Jacobs, Milind Kulkarni, and Roopsha Samanta. Quicksilver: modeling and parameterized verification for distributed agreement-based systems. *Proc. ACM Program. Lang.*, 5(OOPSLA):1–31, 2021.

[39] Daniel Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006.

[40] Swen Jacobs, Mouhammad Sakr, and Martin Zimmermann. Promptness and bounded fairness in concurrent and parameterized systems. In *VMCAI*, volume 11990 of *Lecture Notes in Computer Science*, pages 337–359. Springer, 2020.

[41] Philipp Jeitner, Haya Shulman, and Michael Waidner. The Impact of DNS Insecurity on Time. In *Conference on Dependable Systems and Networks (DSN)*, pages 266–277, 2020.

[42] Pankaj Khanchandani and Christoph Lenzen. Self-Stabilizing Byzantine Clock Synchronization with Optimal Precision. *Theory of Computing Systems (TOCS)*, 63(2):261–305, 2019.

[43] Charles Edwin Killian, James W. Anderson, Ryan Braud, Ranjit Jhala, and Amin Vahdat. Mace: language support for building distributed systems. In *PLDI*, pages 179–188. ACM, 2007.

[44] Charles Edwin Killian, James W. Anderson, Ranjit Jhala, and Amin Vahdat. Life, death, and the critical transition: Finding liveness bugs in systems code (awarded best paper). In *NSDI*. USENIX, 2007.

[45] Attila Kinali, Florian Huemer, and Christoph Lenzen. Fault-tolerant Clock Synchronization with High Precision. In *Symposium on VLSI (ISVLSI)*, 2016.

[46] Igor Konnov, Jure Kukovec, and Thanh-Hai Tran. TLA+ model checking made symbolic. *Proc. ACM Program. Lang.*, 3(OOPSLA):123:1–123:30, 2019.

[47] Igor Konnov and Josef Widder. Bymc: Byzantine model checker. In *ISoLA (3)*, volume 11246 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2018.

[48] Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL*, pages 719–734. ACM, 2017.

[49] Bernhard Kragl and Shaz Qadeer. Layered concurrent programs. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 79–102. Springer, 2018.

[50] Bernhard Kragl and Shaz Qadeer. The civl verifier. In *FMCAD*, pages 143–152. IEEE, 2021.

[51] Markus Alexander Kuppe, Leslie Lamport, and Daniel Ricketts. The TLA+ toolbox. In *F-IDE@FM*, volume 310 of *EPTCS*, pages 50–62, 2019.

[52] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[53] Leslie Lamport. Real-time model checking is really simple. In *CHARME*, volume 3725 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2005.

[54] Christoph Lenzen. Clock Synchronization and Adversarial Fault Tolerance, 2021. https://www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/summer21/clock-synchronization-and-adversarial-fault-tolerance.

[55] Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. Tight Bounds for Clock Synchronization. *Journal of the ACM (JACM)*, 57(2), 2010.

[56] Christoph Lenzen and Julian Loss. Optimal Clock Synchronization with Signatures. *CoRR*, abs/2203.02553, 2022.

[57] Christoph Lenzen and Joel Rybicki. Near-Optimal Self-Stabilising Counting and Firing Squads. *Distributed Computing (DC)*, 32(4):339–360, 2019.

[58] Christoph Lenzen and Joel Rybicki. Self-Stabilising Byzantine Clock Synchronisation Is Almost as Easy as Consensus. *Journal of the ACM (JACM)*, 66(5):32:1–32:56, 2019.

[59] Yao Liu, Peng Ning, and Michael K. Reiter. False Data Injection Attacks against State Estimation in Electric Power Grids. *Transactions on Information and System Security*, 14(1), 2011.

[60] Jacob R. Lorch, Yixuan Chen, Manos Kapritsos, Bryan Parno, Shaz Qadeer, Upamanyu Sharma, James R. Wilcox, and Xueyuan Zhao. Armada: low-effort verification of high-performance concurrent programs. In *PLDI*, pages 197–210. ACM, 2020.

[61] Haojun Ma, Aman Goel, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci, and Karem A. Sakallah. I4: incremental inference of inductive invariants for verification of distributed protocols. In *SOSP*, pages 370–384. ACM, 2019.

[62] Kenneth L. McMillan and Oded Padon. Ivy: A multi-modal verification tool for distributed algorithms. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 190–202. Springer, 2020.

[63] Stephan Merz and Hernán Vanzetto. Automatic verification of TLA + proof obligations with SMT solvers. In *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 289–303. Springer, 2012.

[64] Mifid ii, 2018. https://www.esma.europa.eu/policy-rules/mifid-ii-and-mifir.

[65] David L. Mills. Network Time Protocol (NTP), 1985. https://www.hjp.at/(st_a)/doc/rfc/rfc958.html.

[66] Lakshay Narula and Todd E. Humphreys. Requirements for Secure Clock Synchronization. *IEEE Journal of Selected Topics in Signal Processing (JSTSP)*, 12(4):749–762, 2018.

[67] Chris Newcombe. Why amazon chose TLA +. In *ABZ*, volume 8477 of *Lecture Notes in Computer Science*, pages 25–39. Springer, 2014.

[68] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Commun. ACM*, 58(4):66–73, 2015.

[69] Karen O'Donoghue, Dieter Sibold, and Steffen Fries. New Security Mechanisms for Network Time Synchronization Protocols. In *Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6, 2017.

[70] Junkil Park, Miroslav Pajic, Oleg Sokolsky, and Insup Lee. Automatic verification of finite precision implementations of linear controllers. In *TACAS (1)*, volume 10205 of *Lecture Notes in Computer Science*, pages 153–169, 2017.

[71] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2020. IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008).

[72] Tom Ridge. Verifying distributed systems: the operational approach. In *POPL*, pages 429–440. ACM, 2009.

[73] Alan H. Sanstad, Qianru Zhu, Benjamin Leibowicz, Peter H. Larsen, and Joseph H. Eto. Case Studies of the Economic Impacts of Power Interruptions and Damage to Electricity System Infrastructure from Extreme Events. Technical report, Berkeley Lab, 2020.

[74] Neta Rozen Schiff, Michael Schapira, Danny Dolev, and Omer Deutsch. Preventing (Network) Time Travel with Chronos. In *Applied Networking Research Workshop (ANRW)*, pages 17–31, 2018.

[75] Michael Schmidthaler and Johannes Reichl. Assessing the Socio-economic Effects of Power Outages ad hoc. *Computer Science - Research and Development*, 31:157–161, 2016.

[76] Fred B. Schneider. Technical perspective: Ironfleet simplifies proving safety and liveness properties. *Commun. ACM*, 60(7):82, 2017.

[77] Detlef Schwier and Friedrich W. von Henke. Mechanical verification of clock synchronization algorithms. In *FTRTFT*, volume 1486 of *Lecture Notes in Computer Science*, pages 262–271. Springer, 1998.

[78] M. Sforna and M. Delfanti. Overview of the Events and Causes of the 2003 Italian Blackout. In *Power Systems Conference and Exposition (PSCE)*, pages 301–308, 2006.

[79] Konrad Slind and Michael Norrish. A brief overview of HOL4. In *TPHOLs*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, 2008.

[80] Luca Spalazzi and Francesco Spegni. Parameterized model checking of networks of timed automata with boolean guards. *Theor. Comput. Sci.*, 813:248–269, 2020.

[81] Chih-Che Sun, Adam Hahn, and Chen-Ching Liu. Cyber Security of a Power Grid: State-of-the-art. *International Journal of Electrical Power & Energy Systems*, 99:45–56, 2018.

[82] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the Requirements for Successful GPS Spoofing Attacks. In *Conference on Computer and Communications Security (CCS)*, pages 75–86, 2011.

[83] TPM Working Group. Trusted Platform Module Library Specification - Part 1: Architecture, 2019. `https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.07-2014-03-13.pdf`.

[84] Jennifer L. Welch and Nancy A. Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. *Information and Computation (Inf Comput)*, 77(1):1–36, 1988.

[85] Duncan Wigan. Case Study: The Cum-Cum and Cum-Ex Schemes. Technical Report D4.5, Copenhagen Business School, 2019.

[86] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson. Verdi: a framework for implementing and formally verifying distributed systems. In *PLDI*, pages 357–368. ACM, 2015.

[87] Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long, Lintao Zhang, and Lidong Zhou. MODIST: transparent model checking of unmodified distributed systems. In *NSDI*, pages 213–228. USENIX Association, 2009.

[88] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking tla$^+$ specifications. In *CHARME*, volume 1703 of *Lecture Notes in Computer Science*, pages 54–66. Springer, 1999.