General CONGEST Compilers against Adversarial Edges

Yael Hitron
Weizmann Institute of Science, Rehovot, Israel

Merav Parter
Weizmann Institute of Science, Rehovot, Israel

Abstract

We consider the *adversarial* CONGEST model of distributed computing in which a fixed number of edges (or nodes) in the graph are controlled by a computationally unbounded adversary that corrupts the computation by sending malicious messages over these (a-priori unknown) controlled edges. As in the standard CONGEST model, communication is synchronous, where per round each processor can send $O(\log n)$ bits to each of its neighbors.

This paper is concerned with distributed algorithms that are both time efficient (in terms of the number of rounds), as well as, robust against a fixed number of adversarial edges. Unfortunately, the existing algorithms in this setting usually assume that the communication graph is complete (n-clique), and very little is known for graphs with arbitrary topologies. We fill in this gap by extending the methodology of [Parter and Yogev, SODA 2019] and provide a compiler that simulates any CONGEST algorithm $\mathcal{A}$ (in the reliable setting) into an equivalent algorithm $\mathcal{A}'$ in the adversarial CONGEST model. Specifically, we show the following for every $(2f+1)$ edge-connected graph of diameter D:

- For $f = 1$, there is a general compiler against a single adversarial edge with a compilation overhead of $\widehat{O}(D^3)$ rounds[1]. This improves upon the $\widehat{O}(D^5)$ round overhead of [Parter and Yogev, SODA 2019] and omits their assumption regarding a fault-free preprocessing phase.
- For any constant f, there is a general compiler against f adversarial edges with a compilation overhead of $\widehat{O}(D^{O(f)})$ rounds. The prior compilers of [Parter and Yogev, SODA 2019] were limited to a single adversarial edge.

Our compilers are based on a new notion of fault-tolerant cycle covers. The computation of these cycles in the adversarial CONGEST model constitutes the key technical contribution of the paper.

2012 ACM Subject Classification Networks → Network algorithms

Keywords and phrases CONGEST, Cycle Covers, Byzantine Adversaries

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.24

Related Version *Full Version*: `https://www.weizmann.ac.il/math/parter/sites/math.parter/files/uploads/Adversarial-CONGEST-Compilers.pdf`

Funding *Merav Parter*: This project is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 949083).

Acknowledgements We are very grateful to Dan Mikulincer, David Peleg and Eylon Yogev for many useful discussions. More specifically, we thank Eylon Yogev for suggesting the high-level framework for the 1-FT cycle cover. We also thank the unanimous referee for the useful comments on the proof of Lemma 21.

1 Introduction

As communication networks grow in size, they become increasingly vulnerable to failures and byzantine attacks. It is therefore crucial to develop fault-tolerant distributed algorithms that work correctly despite the existence of such failures, without knowing their location. The

[1] The notation $\widehat{O}(.)$ hides factors of $2^{O(\sqrt{\log n})}$ which arises by the distributed algorithms of [34, 35].

area of fault-tolerant distributed computation has attracted a lot of attention over the years, especially since the introduction of the byzantine agreement problem by [Pease, Shostak and Lamport, JACM'80] [37]. The vast majority of these algorithms, however, assume that the communication graph is the complete graph [12, 13, 16, 8, 43, 41, 6, 42, 5, 15, 18, 17, 27, 38, 25, 14, 29, 23, 10, 26]. For the latter, one can provide time efficient algorithms for various distributed tasks that can tolerate up to a *constant* fraction of corrupted edges and nodes [12, 4, 6, 14]. Very little is known on the complexity of fault-tolerant computation for general graph topologies. In a seminal work, Dolev [12] showed that any given graph can tolerate up to f adversarial nodes iff it is $(2f+1)$ vertex-connected. Unfortunately, the existing distributed algorithms for general $(2f+1)$ connected graphs, usually require a polynomial number of rounds in the CONGEST model of distributed computing [39].

In this paper, we present a general compiler that translates any given distributed algorithm $\mathcal{A}$ (in the fault-free setting) into an equivalent algorithm $\mathcal{A}'$ that performs the same computation in the presence of f adversarial edges. Our primary objective is to minimize the **compilation overhead**, namely, the ratio between[2] the round complexities of the algorithms $\mathcal{A}'$ and $\mathcal{A}$. We take the gradual approach of fault-tolerant network design, and consider first the case of a single adversarial edge, and later on the case of multiple adversarial edges. We note that, in general, such compilers might not be obtained for adversarial nodes[3] and thus we focus on edges.

1.1 Model Definition and the State of the Art

Very recently, [21] presented the first round-efficient broadcast algorithms against adversarial edges in the CONGEST model. [21] also formalized the adversarial CONGEST model, which is the model that we consider in this work as well.

The Adversarial CONGEST Model. The network is abstracted as an n-node graph $G = (V, E)$, with one processor on each node. Each node has a unique identifier of $O(\log n)$ bits. Initially, the processors only know the identifiers of their incident edges[4], as well as a polynomial estimate on the number of nodes n.

There is a computationally unbounded adversary that controls a fixed set of edges F^* in the graph. The set of F^* edges are denoted as *adversarial*, and the remaining edges $E \setminus F^*$ are denoted as *reliable*. The nodes do not know the identity of the adversarial edges in F^*, but they do know the bound f on the cardinality of F^*. We consider the full information model where the adversary knows the graph, the messages sent through the graph edges in each round, and the internal randomness and the input of the nodes. On each round, the adversary can send $O(\log n)$ bits along each of the edges in F^*. The adversary is adaptive as it can determine its behavior in round r based on the overall communication up to round r.

We focus on $(2f+1)$ edge-connected graphs, which can tolerate up to f adversarial edges. The problem of devising general round-by-round compilers in the adversarial CONGEST model boils down into the following distributed task:

[2] Note that we use the term compilation overhead to measure the time it takes to simulate a single fault-free round of algorithm $\mathcal{A}$ in the adversarial setting. This should not be confused with the time required to set up the compiler machinery (e.g., computing the cycle cover).

[3] Such compilers might still be obtained under the stronger KT2 model where nodes know their two-hop neighbors.

[4] This is known as the standard KT1 model [3].

Single Round Compilation in the Adversarial CONGEST Model: Given is a $(2f+1)$ edge-connected graph $G=(V,E)$ with a fixed set $F^* \subseteq G$ of at most f adversarial edges. Let $\mathcal{M}=\{M_{u\to v} \mid (u,v)\in E\}$ be a collection of $O(\log n)$-bit messages that are required to be sent over (potentially) all graph edges. I.e., for each (directed) edge (u,v), the node u has a designated $O(\log n)$-bit message for v.

The single round compilation algorithm is required to exchange these messages in the adversarial CONGEST model, such that at the end of the algorithm, each node v holds the correct message $M_{u\to v}$ for each of its neighbors u, while ignoring all remaining (corrupted) messages.

The main complexity measure is the round complexity of the single-round compilation algorithm which corresponds to the *compilation overhead* of the compiler. The compilation of CONGEST algorithms under various adversarial settings, has been recently studied by [33]. We next explain their methodology and discuss our contribution with respect to the state-of-the-art.

The simulation methodology of [33]. Motivated by various applications for *resilient distributed computing*, Parter and Yogev [33] introduced the notion of *low-congestion cycle covers* as a basic communication backbone for reliable communication. Formally, a (c,d)-cycle cover of a two edge-connected graph G is a collection of cycles in G in which each cycle is of length at most d, and each edge participates in at least one cycle and at most c cycles. The quality of the cycle cover is measured by $c+d$. Using the beautiful result of Leighton, Maggs and Rao [28] and the follow-up by Ghaffari [19], a (c,d)-cycle cover allows one to route $O(\log n)$ bits of information on all cycles simultaneously in $\widetilde{O}(c+d)$ CONGEST rounds.

Low-congestion cycle covers with parameters c,d give raise to a *simulation* methodology that transforms any distributed algorithm $\mathcal{A}$ and compile it into a *resilient* one; the compilation overhead is $g(c,d)$, for some function g. The resilient simulation exploits the fact that a cycle covering an edge $e=(u,v)$ provides *two*-edge-disjoint paths for exchanging messages from u to v. Parter and Yogev [33] showed that any n-node two edge-connected graph with diameter D has a (c,d)-cycle covers with $c=O(1)$ and $d=\widetilde{O}(D)$. These bounds are existentially tight. [34, 35] also presented an r-round CONGEST algorithm for computing (c,d) cycles covers for $r,d=\widehat{O}(D)$ and $c=\widehat{O}(1)$.

Our simulation methodology in the adversarial CONGEST model extends the work of [33] in several aspects. First, the cycle covers of [33] are limited to handle at most *one* edge corruption. To accommodate a large number of adversarial edges, we introduce the notion of *fault-tolerant (FT) cycle covers* which extends low-congestion cycle cover to handle multiple adversarial edges. Informally, a FT cycle cover with parameters c,d is a cycle collection $\mathcal{C}$ that covers each edge e by multiple cycles (instead of one): For every sequence of at most f faults F, there is a cycle C in $\mathcal{C}$ that covers[5] e without visiting any of the edges in $F\setminus\{e\}$. All cycles in $\mathcal{C}$ are required to be of length at most d, and with an overlap of at most c, to allow an efficient information exchange over all these cycles in parallel.

A key limitation of the compilers provided by [33] is that they assume the cycle covers are computed in a (fault-free) preprocessing phase. These cycles are then used by the compilers in the adversarial CONGEST model. Our main goal in this paper is to omit this assumption and provide efficient algorithms for computing the FT cycle covers in

[5] A stricter requirement is to cover each edge by f edge-disjoint cycles, however, this definition leads to a larger compilation overhead compared to the one obtained with our definition.

the adversarial CONGEST model. The computation of these cycles in the presence of the adversarial edges is quite intricate. The key challenge is in computing cycles for covering the adversarial edges themselves. The latter task requires some coordination between the endpoints of the adversarial edges, which seems to be quite hard to achieve. Note that the covering of the adversarial edges by cycles is indeed crucial for the compilation task, in order to reliably simulate the message exchange over these edges in the given fault-free algorithm. Upon computing FT cycle covers with parameter c, d, we then present a round-by-round compiler whose overhead depends on the c, d parameters. To optimize for the round overhead, we exploit (our modified) FT cycle covers in a somewhat more delicate manner compared to that of [33], leading to an improvement by factor of $O(D^2)$ rounds.

1.2 Contributions and Key Results

We consider the design of compilers that can simulate every given distributed algorithm in the adversarial CONGEST model. The compilers are based on a new notion of *FT cycle cover*, an extension of the low-congestion cycle cover [31] to the adversarial setting. We also provide a new method to compile the algorithm given the FT cycle cover. We start by describing our contribution w.r.t the combinatorial characterization of FT cycle covers, and then turn to consider the computational aspects in the adversarial CONGEST model.

1.2.1 Combinatorial Properties of Fault Tolerant Cycle Covers

We provide first the standard definition of low congestion cycle covers of [33], and then introduce their extension to the fault-tolerant setting. A (c, d) *low-congestion cycle cover* of a two edge-connected graph G is a collection of cycles in G in which (i) each cycle is of length at most d (dilation), and (ii) each edge participates in at least *one* cycle (covering), and at most c cycles (congestion). The *quality* of the cycle cover is measured by $c + d$. To provide reliable computation in the presence of f adversarial edges F^*, it is desired to cover each edge by multiple short cycles with small overlap. This motivates the following definition.

▶ **Definition 1** (f-FT Cycle Covers). *Given an $(f+1)$ edge-connected graph G an f-FT cycle cover with parameters (c, d) is a collection of cycles $\mathcal{C}$ such that for any set $E' \subseteq E$ of size $(f-1)$ and every edge $e \in E$, there exists a cycle $C \in \mathcal{C}$ such that $C \cap (E' \cup \{e\}) = \{e\}$. The length of every cycle in $\mathcal{C}$ is at most d, and each edge participates in at most c cycles.*

In other words, the f-FT cycle cover $\mathcal{C}$ provides for each edge $e = (u, v)$ a subgraph G'_e (consisting of all cycles covering e), such that the minimum u-v cut in G'_e is at least $f + 1$. Using the FT sampling technique from [44, 11], in the full version we show the following:

▶ **Lemma 2** (Upper bound on FT Cycle-Covers). *For every $(f+1)$ edge-connected graph G with diameter D, there is a randomized construction for computing f-FT cycle cover $\mathcal{C}$ with parameters (c, d) where $c = f(5fD)^f \cdot \mathrm{poly}(\log n)$ and $d = 5fD$.*

One of our technical contributions is an almost *matching* lower bound for the quality of FT cycle covers. This is done by a careful analysis of the congestion and dilation parameters of replacement paths in faulty graphs. We believe that the following graph theoretical theorem should be of independent interest in the context of fault-tolerant network design and distributed minimum cut computation.

▶ **Theorem 3** (Lower Bound on the Quality of FT Cycle Covers). *For every $f \geq 1$, $D \geq f$ and $n = \omega(D^f)$, there exists an n-node $(f+1)$ edge-connected graph $G^* = (V, E)$ with diameter D, such that any f-FT cycle cover with parameters c, d must satisfy that $c + d = (D/f)^{\Omega(f)}$.*

This theorem provides an explanation for the compilation overhead of $D^{O(f)}$ of our compilers. It also provides an explanation for the natural barrier of $D^{O(f)}$ rounds for handling f adversarial edges in the distribued setting. Specifically, the lower bound implies that there exists at least one pair of nodes u, v in the graph G^* such that for any selection of $f+1$ edge-disjoint u-v paths $\mathcal{P}$ in G^*, the longest path in $\mathcal{P}$ must have length of $(D/f)^{\Omega(f)}$ edges. Theorem 3 also proves that the collection of all $V \times V \times E^f$ replacement paths[6] avoiding f faults, obtained by the FT sampling technique, are optimal in terms of their congestion + dilation bounds. It also shows that the analysis of the distributed minimum cut algorithm of [30] is nearly *optimal*[7].

A relaxed notion of FT cycle covers. In a setting where a fixed set of edges F^* are adversarial for $|F^*| = f$, it might not be possible to compute $(2f)$-FT cycle cover as defined by Definition 1. This is despite the fact that we require the edge connectivity of the graph to be at least $2f+1$. To see this, consider the scenario where the adversarial edges F^* are completely idle throughout the distributed computation. In such a case, the communication graph becomes $G \setminus F^*$, which is no longer guaranteed to have an edge-connectivity of $2f+1$. For this reason, we consider a more relaxed notion of FT cycle covers, that on the one hand can be computed in the adversarial setting, and on the other hand is strong enough for our compilers.

▶ **Definition 4** ((f, F^*)-FT Cycle Cover)**.** *Given an $(2f+1)$ edge-connected graph G, and a fixed set of unknown adversarial edges $F^* \subseteq E$ of size at most f, an (f, F^*)-FT cycle cover with parameters (c, d) is a collection of cycles $\mathcal{C}$ such that for every edge $e \in E$ (possibly $e \in F^*$), and every set $E' \subseteq E$ of size $|E'| \leq f-1$, there exists a cycle $C \in \mathcal{C}$ such that $C \cap (E' \cup F^* \cup \{e\}) = \{e\}$. The length of each cycle is bounded by d, and every edge appears on at most c cycles in $\mathcal{C}$.*

Note that for every $F \subseteq E$, $|F| \leq f$, an (f, F)-FT cycle cover $\mathcal{C}$ *contains* an f-FT cycle cover, and therefore the lower bound of Theorem 3 also holds for (f, F^*)-FT cycle cover. When $F^* = \{e'\}$, we slightly abuse notation and simply write (f, e')-FT cycle covers Our FT cycle covers should be useful for many other adversarial settings. Specifically, they provide an immediate extension of the compilers of [33] to handle adversaries that corrupt multiple edges, such as eavesdroppers [33] and semi-honest adversaries [32].

We next turn to consider the computational aspects of FT cycle covers, and their applications. In the distributed setting, we assume throughout that the nodes of the graph obtain a linear estimate[8] on the diameter of the graph D. This assumption (also applied in e.g., [9]) is needed as the compilation overhead is a function of D.

1.2.2 Handling a Single Adversarial Edge

We start by considering an adversarial setting with a *single* fixed unknown adversarial edge e'. At the heart of the compiler lies an efficient construction of a $(1, e')$ FT cycle cover in the adversarial CONGEST model.

[6] A replacement path is a shortest path in some graph $G \setminus F$.
[7] This algorithm computes the minimum cut by computing for each vertex v the collection of all replacement paths w.r.t a fixed source node s.
[8] This assumption can be omitted using the broadcast algorithms of [21, 22], in the case where the nodes have a designated marked leader.

▶ **Theorem 5** ($(1,e')$-FT Cycle Cover). *Consider a 3 edge-connected n-node graph G of diameter D, and a fixed adversarial edge e'.*

- *There is an r-round deterministic algorithm for computing a $(1,e')$-FT cycle cover with congestion and dilation $c = \widehat{O}(D^2), d = \widehat{O}(D)$, and $r = \widehat{O}(D^4)$ in the adversarial* CONGEST *model.*
- *There is an r-round randomized algorithm for computing a $(1,e')$-FT cycle cover, w.h.p., with congestion and dilation $c, d = \widehat{O}(D)$, and $r = \widehat{O}(D^2)$ in the adversarial* CONGEST *model.*

In the distributed output format of the $(1,e')$-FT cycle cover computation, the endpoints of every edge $e = (u,v)$ hold the unique identifiers of all the cycles $\mathcal{C}_e$ covering e ,as well as, their neighbors on each of these cycles. The key challenge in proving Theorem 5 is in covering the adversarial edge e'. For that purpose we provide a delicate cycle verification procedure that allows the endpoints of each edge $e = (u,v)$ to correctly identify if e is currently covered by a (legal) cycle. This verification is robust to the behavior of the adversarial edge. Using these cycle covers, we obtain general compilers against e'.

▶ **Theorem 6** (Compiler against a Single Adversarial Edge). *Given is a 3 edge-connected D–diameter graph G with a fixed adversarial edge e', and a $(1,e')$-FT cycle cover $\mathcal{C}$ with parameters (d,c) for G (e.g., as obtained by Theorem 5). Then any distributed algorithm $\mathcal{A}$ can be compiled into an equivalent algorithm $\mathcal{A}'$ against e' with an overhead of $O(c \cdot d^2)$ rounds (in the adversarial* CONGEST *model).*

This improves the compilation overhead of Parter and Yogev [33] by a factor of $\widetilde{O}(D^2)$ rounds. The compilers of [33] are based on exchanging the $M_{u \to v}$ messages of Alg. $\mathcal{A}$ along 3 edge-disjoint u-v paths. In our compilation scheme, instead of insisting on edge-disjoint paths, the messages are exchanged over a collection u-v paths of a sufficiently large *flow*. This leads to improvement in the compilation overhead.

1.2.3 Handling Multiple Adversarial Edges

We next consider $(2f+1)$ edge-connected graphs of diameter D with a fixed set $F^* \subseteq E$ of adversarial edges, $|F^*| \leq f$. To handle f adversarial edges F^* in $(2f+1)$ edge-connected graphs, we use the notion of (f,F^*)-FT cycle covers. Our first contribution is the construction of the (f,F^*)-FT cycle covers in the adversarial CONGEST model. Due to technicalities arises in this adversarial setting, our final output contains the desired cycles required by (f,F^*)-FT cycle cover, but might include in addition, also truncated paths which are quite "harmless" in the compilation process later on. Formally, our distributed construction computes (f,F^*)-FT cycle cover* where the asterisk indicates the possible existence of truncated paths in the distributed output.

▶ **Definition 7** ((f,F^*)-FT Cycle Cover*). *Given a $(2f+1)$ edge-connected graph G and a fixed set of adversarial edges $F^* \subseteq E$ where $|F^*| \leq f$, a (f,F^*)-FT cycle cover* with parameters (c,d) is a collection of cycles and paths $\mathcal{C}$ such that $\mathcal{C}$* contains *a (f,F^*)-FT cycle cover for G. The length of each cycle and path in $\mathcal{C}$ is at most d and every edge $e \in E$ appears in at most c cycles and paths.*

▶ **Theorem 8** ((f,F^*)-FT Cycle Cover*). *Let G be a $(2f+1)$ edge-connected graph G of diameter D, and a fixed set of f adversarial edges F^*. Then, there exists an r-round deterministic algorithm, in the adversarial* CONGEST *model, for computing a (f,F^*)-FT cycle cover* for G with parameters $d = \widehat{O}(f \cdot D)$ and $r, c = \widehat{O}((Df \log n)^{O(f)})$.*

Note that by the lower bound result of Theorem 3, the quality of the FT cycle covers must be $(D/f)^{\Omega(f)}$. Given a (f, F^*)-FT cycle cover* for a graph G, we extend the general compiler of Theorem 6 to handle f adversarial edges.

▶ **Theorem 9** (Compilers against f Adversarial Edges). *Given a $(2f + 1)$ edge-connected D–diameter graph G with a fixed set of f adversarial edges F^*, and a (f, F^*)-FT cycle cover* with parameters (d, c) for G. Then any distributed algorithm $\mathcal{A}$ can be compiled into an equivalent algorithm $\mathcal{A}'$ against F^*, with a compilation overhead of $O(c \cdot d^3)$ rounds.*

The high level intuitive idea of our compiler is as follows. Fix a round i of algorithm $\mathcal{A}$, and consider the message $M_{u\to v}$ sent over the edge (u, v) in that round. Our compiler lets u send the message $M_{u\to v}$ through all cycles covering e in the (f, F^*)-FT cycle cover*. The node v can then recover $M_{u\to v}$ by exploiting the following property. On the one hand, the (f, F^*)-FT cycle cover* covers e by sufficiently many cycles that avoid $F^* \setminus \{e\}$. Consequently, the correct message $M_{u\to v}$ is received by v over a path collection with a u-v flow[9] at least $f + 1$. On the other hand, any corrupted message $M' \neq M_{u\to v}$ must be propagated along a walk that contains at least *one* adversarial edge. Consequently, a corrupted message M' is propagated over a walk collection with a u-v flow at most f.

Technical comparison with [21]. The recent work of [21] provides broadcast algorithms in the adversarial CONGEST model. This paper is concerned with a general compiler that translates any CONGEST algorithm into an adversarial CONGEST algorithms provided that the adversary controls at most f edges in the graph. The common tool used by both of the works is the covering family obtained by the FT sampling and its recent derandomization [24, 7]. Besides this, each paper handles different types of challenges. In the broadcast task the goal is to send the broadcast message m_0 through a collection of sufficiently many reliable paths. In contrast, in the compiler setting, given a (fault-free) algorithm $\mathcal{A}$, the goal is to exchange messages of $\mathcal{A}$ over (potentially) *all* graph edges in a reliable manner. Specifically, unlike the broadcast setting, one cannot simply ignore the adversarial edges (e.g., by exchanging messages over a reliable subgraph $G' \subseteq G$), as it is required to exchange messages in a reliable manner over the endpoints of the *adversarial* edges as well. The heart of this simulation is in the computation of fault-tolerant cycle covers.

1.3 Preliminaries

Notations. Throughout, the diameter of the given graph G is denoted by D, and the number of nodes by n. For a graph $G = (V, E)$, a subgraph $G' \subseteq G$, and nodes $u, v \in V(G')$, let $\pi(u, v, G')$ be the unique u-v shortest path in G' where shortest-path ties are decided arbitrarily in a consistent manner. Let $N(u, G)$ be the neighbors of node u in the graph G. When the graph G is clear from the context we may omit it and write $N(u)$. For a path $P = [u_1, \ldots, u_k]$ and an edge $e = (u_k, v)$, let $P \circ e$ denote the path obtained by concatenating e to P. Similarly, for two paths $P_1 = [u_1, \ldots, u_k], P_2 = [u_k, u_{k+1}, \ldots, u_\ell]$ denote the concatenated path $[u_1, \ldots, u_k, u_{k+1}, \ldots, u_\ell]$ by $P_1 \circ P_2$. Given a path $P = [u_1, \ldots, u_k]$ denote the sub-path from u_i to u_ℓ by $P[u_i, u_\ell]$. The term $\widetilde{O}(\cdot)$ hides poly$(\log n)$ factors, and the term $\widehat{O}(\cdot)$ hides $2^{O(\sqrt{\log n})}$factors[10].

[9] To formalize this argument, we provide a formal definition for the cut value of a u-v walk collection.
[10] The latter factors arise by the (fault-free) distributed computation of cycle covers by [34].

▶ **Definition 10** (Neighborhood Covers, [2]). *The r-neighborhood cover of the graph G is a collection of vertex subsets, denoted as, clusters $\mathcal{N} = \{S_1, \ldots, S_\ell\}$ where $S_i \subseteq V$ such that: (i) every node v has a cluster that contains its entire r-radius neighborhood in G, (ii) the diameter of each $G[S_i]$ is $O(r \log^c n)$ for some constant c, and (iii) every node belongs to $\widetilde{O}(1)$ clusters in $\mathcal{N}$.*

We use the deterministic construction of neighborhood covers by Rohzon and Ghaffari [40] .

▶ **Theorem 11** (Corollary 3.5 [40]). *There is a deterministic distributed algorithm that for any radius $r \geq 1$, computes an r-neighborhood cover $\mathcal{N}$ within $\widetilde{O}(r)$* CONGEST *rounds.*

Low-congestion cycle covers. The construction of FT cycle covers is based on the distributed construction of (c, d) cycle covers in the standard CONGEST model. In particular, we use the construction from [34, 33] that covers each edge $e = (u, v)$ by a cycle C_e such that $|C_e| = \widetilde{O}(\text{dist}_{G \setminus \{e\}}(u, v))$.

▶ **Fact 12** ([34, 33]). *There is a randomized algorithm* ComputeCycCov(G, D') *that for any n-node input graph $G = (V, E)$ and an input parameter D', computes, w.h.p., a cycle collection $\mathcal{C}$ with the following properties: (1) every edge $e \in E$ that lies on a cycle of length at most D' in G is covered by a cycle in $\mathcal{C}$ of length $\widehat{O}(D')$, and (2) each edge appears on $\widehat{O}(1)$ cycles. Algorithm* ComputeCycCov(G, D') *runs in $\widehat{O}(D')$ rounds. In the output format, each node knows the edges of the cycles that cover each of its incident edges.*

Note that Alg. ComputeCycCov does not require the graph G to be connected. This will be important in our context. This algorithm can also be made deterministic using the neighborhood covers of Theorem 11.

▶ **Observation 13.** *The algorithm* ComputeCycCov(G_i, D') *of Fact 12 can be made deterministic using the neighborhood covering algorithm of Theorem 11. Additionally, in the output format of the algorithm, each node u knows a $\widehat{O}(1)$-bit unique identifier for each of the cycles it belongs to, as well as a full description of the cycle, obtained from both directions.*

Covering families. Our distributed algorithms in the adversarial CONGEST model are based on communication over a collection of G-subgraphs that we denote as covering family. These families are used extensively in the context of fault-tolerant network design [1, 44, 11, 20, 30, 36, 9, 7, 24, 21].

▶ **Definition 14** ((L, t) Covering Families). *For a given graph G, a family of G-subgraphs $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ is a (L, t)* covering family, *if for every edge $e = (u, v) \in E$ and every set $F \subseteq E$ where $|F| \leq t - 1$, such that*[11] $\text{dist}_{G \setminus F \cup \{e\}}(u, v) \leq L$, *there exists a subgraph G_i satisfying that (P1) $\text{dist}_{G_i \setminus (F \cup \{e\})}(u, v) \leq L$, and (P2) $(F \cup \{e\}) \cap G_i = \{e\}$.*

Throughout, we use the following observation from [21].

▶ **Observation 15** (Observation 8 from [21]). *Consider a D-diameter graph $G = (V, E)$ and assume that $u, v \in V$ are connected in $G \setminus F$ for some $F \subseteq G$. It then holds that* $\text{dist}_{G \setminus F}(u, v) \leq 2(|F| + 1) \cdot D + |F|$.

[11] We note that our definition slightly differs from that of [21], in the sense that for a pair $e = (u, v), F$, we require the graph G_i (see below) to contain a cycle of length at least L covering e, rather than an L-length u-v path.

We note that by Observation 15, if G is $(t+1)$ edge-connected, then a $(5tD,t)$ family satisfies (P1) and (P2) for any edge $(u,v) \in E$, and an edge set F of size at most $(t-1)$. For our purposes, it is required for the nodes to know the covering family in the following sense.

▶ **Definition 16** (Local Knowledge of a Subgraph Family). *A family of ordered subgraphs $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ where each $G_i \subseteq G$, is* locally known *if given the identifier of an edge $e = (u,v)$ and an index i, u and v can locally determine if $e \in G_i$.*

▶ **Fact 17** ([24]). *Given a graph G and an integer parameter L, the following holds.*

1. *Given that all nodes share a seed $\mathcal{S}$ of $\widetilde{O}(1)$ random bits, there exists a 0-round randomized algorithm for locally computing a $(L,1)$-covering (ordered subgraph) family $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ such that $\ell = \widetilde{O}(L)$, where the covering property holds w.h.p. Given the seed $\mathcal{S}$, index $i \in \{1, \ldots, \ell\}$ and an edge identifier (u,v), each node can locally determine if $(u,v) \in G_i$.*
2. *For every $t \geq 1$, there exists a 0-round deterministic algorithm for computing a (L,t) covering family $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ such that $\ell = ((Lt\log n)^{t+1})$. This covering family is locally known.*

Broadcast against adversarial edges. Our algorithms for constructing FT-cycle covers make use of the broadcast algorithms of Hitron and Parter [21], which are resilient to adversarial edges. We will use the following facts.

▶ **Theorem 18** ([21] Broadcast against a Single Adversarial Edge). *Given a D–diameter, 3 edge-connected graph G and an unknown adversarial edge e', the following holds.*

1. *There exists a deterministic broadcast algorithm which delivers a message m_0 from a designated node s to all nodes in V within $\widetilde{O}(D^2)$ rounds. In addition, at the end of the algorithm, all nodes obtain a linear estimate for the* diameter *of the graph.*
2. *There exists a randomized broadcast algorithm which delivers a message m_0 from a designated node s to all nodes in V within $\widetilde{O}(D)$ rounds, provided that all nodes share $\widetilde{O}(1)$ random bits.*

In addition, the same bounds hold in the case where there are multiple sources holding the same broadcast message m_0.

▶ **Theorem 19** ([21] Broadcast against f Adversarial Edges). *There exists a deterministic broadcast algorithm against f adversarial edges, for D-diameter, $(2f+1)$ edge-connected graphs, with round complexity of $(tD\log n)^{O(t)}$. In addition, the same bound holds in the case where there are multiple sources holding the same broadcast message m_0.*

The broadcast algorithm of Theorem 18 also implies a leader election algorithm. For completeness the proof of the following claim is given in the full version.

▷ Claim 20. [Leader Election against an Adversarial Edge] Given a D–diameter, 3 edge-connected graph G and an adversarial edge e', assuming a linear upper bound $D' = cD$ on the diameter (for some constant $c \geq 1$), there exists a randomized algorithm AdvLeaderElection that w.h.p elects a single leader known to all nodes in the graph within $\widetilde{O}(D^2)$ rounds.

2 Compilers against a Single Adversarial Edge

We first describe the construction of $(1,e')$-FT cycle covers where e' is the adversarial edge. Then, we describe how to compile a single round using these cycles.

2.1 $(1, e')$-FT Cycle Covers

In this section, we prove Theorem 5. This section is devoted for showing the following key lemma that computes a $(1, e')$-FT cycle cover given a locally known covering family.

▶ **Lemma 21.** *Given is a* 3 *edge-connected graph G, with a fixed unknown adversarial edge e'. Let L be an integer satisfying that for every edge $e = (u, v)$ it holds that $\text{dist}_{G\setminus\{e,e'\}}(u, v) \leq L$. Assuming that all nodes locally know a $(L, 1)$ covering family $\mathcal{G}$ of size ℓ, there exists a deterministic algorithm* ComputeOneFTCycCov *for computing a $(1, e')$ FT-cycle cover $\mathcal{C}$ with parameters $c = \widehat{O}(\ell), d = \widehat{O}(L)$ within $\widehat{O}(L^2 \cdot \ell)$ rounds.*

Since the computation of the $(L, 1)$ covering family is straightforward using known tools, we focus on proving Lemma 21. As a warm-up, we describe the construction assuming a reliable setting (with no adversarial edges). Then, we handle the real challenge of the $(1, e')$-FT cycle cover computation in the presence of an adversarial edge.

Warm-up: $(1, e')$-FT cycle covers in a reliable communication graph. The construction is based on applying the cycle cover algorithm of [34] on every subgraph G_i in the covering family $\mathcal{G}$. Specifically, given a locally known covering family $\mathcal{G} = \{G_1, \ldots, G_\ell\}$, the algorithm proceeds in ℓ iterations. In each iteration i it applies the cycle cover algorithm ComputeCycCov(G_i, L) from Observation 13 on the graph G_i with a diameter estimation L, resulting in a cycle collection $\mathcal{C}_i$. The final cycle collection is given by $\mathcal{C} = \bigcup_{i=1}^{\ell} \mathcal{C}_i$, that is, the union of all cycles computed in the ℓ iterations. We next analyze the construction.

Correctness. The round complexity, the cycle length, and the edge congestion bounds follow immediately by the construction. It remains to show that the cycle collection $\mathcal{C}$ is indeed a $(1, e')$-FT cycle cover. To see this, consider a fixed pair of edges $e = (u, v), e'$. We will show that $\mathcal{C}$ contains a cycle $C_{e,e'}$ that contains e and does not contain e'. An iteration i is defined to be *good* for the edge pair e, e' if $e' \notin G_i$, $e \in G_i$ and $\text{dist}_{G_i\setminus\{e\}}(u, v) \leq L$. Since, $\text{dist}_{G\setminus\{e,e'\}}(u, v) \leq L$, due to the covering property of $\mathcal{G}$, there exists a good iteration i^* for every pair e, e'. We next show that e is successfully covered in iteration i^* by some cycle C_e. By the properties of Alg. ComputeCycCov, in iteration i^* the edge e is covered by a cycle C of length $\widehat{O}(L)$. In addition, as $e' \notin G_{i^*}$ this cycle does not contain e' as required.

Algorithm ComputeOneFTCycCov (Proof of Lemma 21). Given is a locally known covering family $\mathcal{G} = \{G_1, \ldots, G_\ell\}$. The algorithm works in ℓ iterations, where in iteration i it performs the computation over the subgraph G_i. Since $\mathcal{G}$ is locally known, every node knows its incident edges in G_i, and ignores messages from other edges in that iteration. Iteration i then consists of two steps. In the first step, the nodes apply Alg. ComputeCycCov(G_i, L) of Observation 13 over the graph G_i with diameter estimate L. This results in a cycle collection $\mathcal{C}'_i(u)$ for every node u. In the output format of Alg. ComputeCycCov, every cycle in $\mathcal{C}'_i(u)$ is presented by a tuple $(ID(C), C)$, where $ID(C)$ is the unique identifier of the cycle of size $\widehat{O}(1)$ bits, and C is the collection of the cycle edges[12]. Since e' might be in G_i, the cycles of $\mathcal{C}'_i(u)$ can be totally corrupted.

In the second step of iteration i, the nodes apply a verification procedure for their cycles in $\mathcal{C}'_i(u)$. Only verified cycles will then be added to the set of cycles $\mathcal{C}_i(u)$. In the analysis section, we show that for every reliable edge $e = (u, v) \neq e'$, there exists at least one cycle in $\mathcal{C}(u) = \bigcup_i^{\ell} \mathcal{C}_i(u)$ that covers e. The third step of the algorithm handles the remaining

[12] Recall that in Alg. ComputeCycCov(G_i, L), each node receives the cycle description C from both directions, i.e., from its two neighbors on C. In case a node u obtained distinct cycle descriptions from its two neighbors, it omits the cycle from its cycle collection $\mathcal{C}'_i(u)$.

adversarial edge, in case needed. We next elaborate on these steps in more details. We focus on iteration i where the nodes communicate over the graph $G_i \in \mathcal{G}$.

Step (1) of iteration i: Cycle cover computation. The (fault-free) cycle cover algorithm $\mathsf{ComputeCycCov}$ of Observation 13 is applied over the subgraph $G_i \in \mathcal{G}$, with parameter L. Since the graph G_i is locally known, each node can verify which of its incident edges lie on G_i and ignore the messages from the remaining edges. During the execution of Alg. $\mathsf{ComputeCycCov}(G_i, L)$, if a node u receives an illegal message, or different cycle descriptions with the same cycle ID, these messages are ignored, as well as future messages in that iteration. At the end of the execution of $\mathsf{ComputeCycCov}(G_i, L)$, each node u performs the following verification step on its output cycle set $\mathcal{C}'_i(u)$. The goal of this verification is to ensure each cycle in $\mathcal{C}'_i(u)$ corresponds to a legal cycle.

Step (2) of iteration i: Cycle verification. First, each node u performs a *local* inspection of its cycles in $\mathcal{C}'_i(u)$, and declares the iteration to be *faulty* if $\mathcal{C}'_i(u)$ contains at *least* one of the following:

1. A cycle of length $\widehat{\omega}(D)$;
2. An edge appearing in $\widehat{\omega}(1)$ cycles in $\mathcal{C}'_i(u)$;
3. A partial cycle (i.e., a walk rather than a cycle);
4. Inconsistency in a cycle description $(ID(C), C) \in \mathcal{C}'_i(u)$ as obtained through the two neighbors of u on C.

In the case where $\mathcal{C}'_i(u)$ is found to be faulty, u sets $\mathcal{C}_i(u) = \emptyset$, and will remain silent throughout this verification step. We will call such a node an *inactive* node. A node whose local inspection is successful is called *active*.

We now describe the global verification procedure for an active node u. The verification step is performed in *super-rounds* in the following manner. Each super-round consists of $c = \widehat{O}(1)$ rounds, which sets the upper bound on the number of cycles that an edge (u, v) participates in. A single super-round has the sufficient bandwidth to exchange a single message through an edge (u, v) for each of the cycles on which (u, v) lies. We then explicitly enforce that in each super-round, each node u sends over an edge (u, v) at most *one* message per cycle $(ID(C), C) \in \mathcal{C}'_i(u)$ for which $(u, v) \in C$.

For a cycle $(ID(C), C) \in \mathcal{C}'_i(u)$, let v_C be the node with largest ID in the cycle description C obtained by u during Alg. $\mathsf{ComputeCycCov}(G_i, L)$. We note that the cycle description C is not necessarily correct, and in particular, it could be that $(ID(C), C) \notin \mathcal{C}'_i(v_C)$. For each cycle $(ID(C), C) \in \mathcal{C}'_i(u)$ such that $u = v_C$ (the cycle's leader), it initiates the following verification steps.

(2.1) A leader v_C of a cycle $(ID(C), C) \in \mathcal{C}'_i(v_C)$ sends the verification message $ver(C) = (ID(C), ID(v_C), ver)$ along its two incident edges on this cycle (i.e., in the clock-wise and counter clockwise directions).

(2.2) The verification messages are then propagated over the cycles for $R = \widehat{O}(L)$ super-rounds, where $\widehat{O}(L)$ is the upper bound on the maximal cycle length. Upon receiving a verification message $ver(C) = (ID(C), ID(v_C), ver)$, an active node u sends $ver(C)$ to a neighbor $w \in N(u)$ if the following conditions hold: (1) $(ID(C), C_u) \in \mathcal{C}'_i(u)$ for some cycle C_u, (2) v_C is the node with the highest ID in C_u, (3) w is a neighbor of u in C_u, and (4) u received the message $ver(C)$ from its second neighbor on the cycle C_u.

(2.3) A leader v_C of a cycle C such that $(ID(C), C) \in \mathcal{C}_i(v_C)$, which did not receive the verification message $ver(C)$ from both its neighbors in C within R super-rounds, initiates a *cancellation message*, $cancel(C) = (ID(C), ID(v_C), cancel)$, and sends it to both its neighbors in C. This indicates to the nodes on this cycle that the cycle should be omitted from their cycle collection.

(2.4) The cancellation messages $cancel(C)$ are propagated over the cycle C for R super-rounds in the following manner. Let τ_i be the first super-round of Step (2.3). In this super-round, v_C may start propagating the message $cancel(C)$ (if the conditions of 2.3 hold). Note, however, that the cancellation messages might also originate at the adversarial edge e'. Step (2.4) handles the latter scenario by augmenting the cancellation messages $cancel(C)$ with distance information. For every node u let d^1_u, d^2_u be the u-v_C distance on C along the first (second) u-v_C path in C. Note that u can locally compute d^1_u, d^2_u using the cycle description of C. A vertex u upon receiving a $cancel(C)$ message from its neighbor v on C acts as follows. Let d^j_u be the length of the v_C-u path on C that passed through v. Then, if the message $cancel(C)$ is received at u from v in super-round $r_j = \tau_i + d^j_u$, u *accepts* the cancellation message and sends it to its other neighbor on C. All other cancellation messages received by u in later or prior super-rounds are dropped.

(2.5) A leader v_C of a cycle $(ID(C), C) \in \mathcal{C}_i(v_C)$ that received a cancellation message $cancel(C)$ that it did not initiate from only *one* direction (i.e., from exactly one of its neighbors on C), broadcasts a cancellation message $cancel(i)$, i.e., canceling iteration i, to all the nodes in the graph by using the broadcast algorithm of Theorem 18(1) over the graph G. Since there is only one broadcast message $cancel(i)$ to be sent on that iteration, possibly by many cycle leaders, this can be done in the same time as a single broadcast operation (i.e., within $\widetilde{O}(D^2)$ rounds).

(2.6) A node that *accepts* a cancellation message $cancel(i)$ via the broadcast algorithm, omits all cycles obtained in this iteration i.

At the end of the i'th iteration, every node u defines a verified cycle set $\mathcal{C}_i(u)$. A cycle $(ID(C), C) \in \mathcal{C}'_i(u)$ is defined as *verified* by u if the following conditions hold (i) it received a verification message $ver(C)$ from both neighbors in C, (ii) any cancellation message $cancel(C)$ received by u has been dropped, and (iii) u accepted no cancellation message $cancel(i)$. Every verified cycle $(ID(C), C) \in \mathcal{C}'_i(u)$ is added to the set $\mathcal{C}_i(u)$. Thus, $\mathcal{C}_i(u)$ consists of all verified cycles passing through u computed in iteration i. This concludes the description of the i'th iteration. The output of each node u is $\mathcal{C}(u) = \bigcup_{i=1}^{\ell} \mathcal{C}_i(u)$.

Step (3): Covering the adversarial edge. For a node u, an incident edge (u, v) is considered by u to *covered* if there exists a tuple $(ID(C), C) \in \mathcal{C}(u)$ such that $(u, v) \in C$. The goal of the third and final step is to cover the remaining uncovered edges. Every node u and an uncovered edge (u, v), broadcasts the edge (u, v) using the deterministic broadcast algorithm of Theorem 18(1). In the analysis section, we show that if there is an uncovered edge then it must be the adversarial edge. The reason for broadcasting the edge (u, v) by its endpoints is to prevent the adversarial edge from initiating this step (despite the fact that all edges are covered). To cover (u, v), the endpoint with the larger identifier, say u, initiates a construction of a BFS tree T rooted at u in $G \setminus \{(u, v)\}$. Within $O(L)$ rounds, u and v learn the u-v tree path P. Then the cycle covering (u, v) is given by $C = (v, u) \circ P$. The cycle $(ID(C), C)$ is then[13] added to the cycle collection $\mathcal{C}(w)$ of every $w \in C$.

The correctness is deferred to the full version. We also show that the graph G is not required to be 3 edge-connected or with a bounded diameter. Our cycle cover algorithm has the

[13] The ID of the cycle C can obtained by appending the maximum ID vertex on C with a special tag indicating that the cycle is added in Step (3).

guarantee to cover every reliable edge that lies on a reliable short cycle in G. That is, we achieve the following.

▶ **Lemma 22.** *There exists a deterministic algorithm* $\mathsf{DetComputeOneFTCycCov}(G, L)$ *that given a graph G containing a single adversarial edge e' and a parameter L, returns a collection of cycles and paths $\mathcal{C}$ with the following property. Every reliable edge $(u, v) \neq e'$ for which* $\mathrm{dist}_{G \setminus \{e', (u,v)\}}(u, v) \leq L$ *is covered by a reliable $\widehat{O}(L)$-length cycle $C \in \mathcal{C}$ such that $e' \notin C$.*

2.2 General Compilers Given $(1, e')$-FT Cycle Cover

We next show that our $(1, e')$-FT cycle cover with parameters (c, d) yields a general compiler that translates any r-round distributed algorithm $\mathcal{A}$ into an equivalent algorithm $\mathcal{A}'$ that works in the presence e'.

Compiler against a single adversarial edge (Proof of Theorem 6). The compiler works in a round-by-round fashion, where every round of $\mathcal{A}$ is implemented in $\mathcal{A}'$ using a phase of $O(c \cdot d^2)$ rounds. At the end of the i'th phase, all nodes will be able to recover the original messages sent to them in round i of algorithm $\mathcal{A}$.

Compilation of round i. Let $\mathcal{C}$ be the cycle collection of the $(1, e')$-FT cycle cover. Fix a round i of algorithm $\mathcal{A}$, and let $M_{u \to v}$ be the message sent from u to v for every pair of neighbors $e = (u, v) \in E$ during the i'th round. In the i'th phase of $\mathcal{A}'$, the node u sends v the message $M_{u \to v}$ through e and *all* u-v paths $\mathcal{P}_{u,v} = \{C \setminus \{e\} \mid C \in \mathcal{C}, e \in C\}$. When sending the messages, each node on a path $P \in \mathcal{P}_{u,v}$ sends at most one message targeted from u to v. If a node w is requires to send at least two different messages from u to v, it omits both messages and sends a null message Φ over the cycle.

At the end of phase i, each node v sets the message $\widetilde{M}_{u,v}$ as its estimate for the message $M_{u \to v}$ sent by u in round i of $\mathcal{A}$. The estimate $\widetilde{M}_{u,v}$ is defined by applying the following protocol. In the case that v receives an identical message $M \neq \Phi$ from u through all the paths in $\mathcal{P}_{u,v}$, then $\widetilde{M}_{u,v} \leftarrow M$. Otherwise, $\widetilde{M}_{u,v} \leftarrow M'$ where M' is the message v received over the direct edge (u, v).

Correctness. We show that at the end of phase i for every edge $(u, v) \in E$ it holds that $\widetilde{M}_{u,v} = M_{u \to v}$. Consider the following two cases.

Case $e = e'$ is the adversarial edge. Since all u-v paths in $\mathcal{P}_{u,v}$ are reliable, all messages received by v over these paths must be identical. Thus, all the messages that v receives through the paths are identical, and equal to $M_{u \to v}$. By the definition of the $(1, e')$-FT cycle cover, $\mathcal{P}_{u,v} \neq \emptyset$. Hence, v accepts the correct message.

Case $e \neq e'$ is reliable. The message that u receives from v through the direct edge e is $M' = M_{u \to v}$. By the definition of the $(1, e')$-FT cycle cover there exists a cycle $C \in \mathcal{C}$ covering e that does not contain e'. Hence, if all edges on C deliver the same message from u to v, it must be the message sent by u. Thus, if all messages v received through the paths $\mathcal{P}_{u,v}$ are identical and differ from Φ, they are equal to $M_{u \to v}$. Otherwise, v accepts the correct message $M' = M_{u \to v}$ delivered through the reliable edge (u, v).

Round complexity. Since each edge belongs to at most c cycles in the $(1, e')$-FT cycle cover $\mathcal{C}$, and as all cycles are of length at most d, the number of messages sent over an edge in a given phase is bounded by $c \cdot d$. Hence, each phase is implemented in $O(c \cdot d^2)$ rounds.

3 Compilers against Multiple Adversarial Edges

At the heart of the compilers lies the construction of (f, F^*)-FT cycle covers in the adversarial CONGEST model, that we describe in this section. The description of the compilers that exploit these cycles are deferred to the full version. Our main result is a deterministic construction of (f, F^*)-FT cycle covers* in the adversarial CONGEST model.

▶ **Lemma 23.** *Given is an $(2f+1)$ edge-connected graph G with a fixed subset of unknown adversarial edges F^* of size f. Assuming all nodes locally know an $(L = 7fD, 2f)$-covering family $\mathcal{G}$ of size ℓ, there exists an r-round algorithm* ComputeFTCycCov *for computing an (f, F^*)-FT cycle cover* with parameters $d = \widehat{O}(L)$, $c = \widehat{O}(\ell \cdot L^2)$, and $r = \widehat{O}(\ell \cdot (fD \log n)^{O(f)})$ in the adversarial* CONGEST *model.*

The proof of Theorem 8 follows by combining Lemma 23 and Fact 17. The algorithm ComputeFTCycCov uses an $(L, 1)$ covering family $\mathcal{G}$ with slightly different properties than those provided in Definition 14. Specifically, we use the following fact from [24].

▷ Claim 24 ([24]). Given a graph G and an integer parameter L, there exists a (deterministic) 0-round algorithm that allows all nodes to locally know a family of subgraphs $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ of size $\ell = \widetilde{O}(L^2)$ where for every edge $e = (u, v) \in G$ such that $\text{dist}_{G \setminus \{e\}}(u, v) \leq L$ there exists a subgraph G_i satisfying that (P1) $\text{dist}_{G_i \setminus (\{e\})}(u, v) \leq L$, and (P2') $e \notin G_i$.

Our Approach. Before presenting the algorithm, we provide the high level approach. Consider the following natural algorithm for computing an (f, F^*)-FT cycle cover. Let $\mathcal{G}$ be an $(L, 2f)$ covering family for $L = O(fD)$. By applying the (fault-free) algorithm ComputeCycCov from Fact 12 on each subgraph $G_i \in \mathcal{G}$, we have the guarantee that all the reliable edges $E \setminus F^*$ are covered successfully as required by Definition 4. The key challenge is in determining whether the adversarial edges are covered as well. In particular, it might be the case that an edge $e \in F^*$ mistakenly deduces that it is covered, leading eventually to an illegal compilation of the messages sent through this edge. Note that unlike $(1, e')$-FT cycle covers, here an edge is covered only if it is covered by cycles of sufficiently large "flow".

Our approach is based on reducing the problem of computing an (f, F^*)-FT cycle cover into the problem of computing $(1, e')$-FT cycle covers in *multiple* subgraphs for every $e' \in F^*$. Specifically, we define a covering family $\mathcal{G}$ with the following guarantee for each adversarial edges $e' \in F^*$: for every $F \subseteq G$, $|F| \leq f$, there exists a subgraph G_i containing a short cycle covering e' such that $G_i \cap (F^* \setminus \{e'\} \cup F) = \emptyset$. Since the covering guarantees for every $e' \in F^*$ are based on such "good" subgraphs G_i, it is safe to apply Alg. DetComputeOneFTCycCov (from Lemma 22) on these subgraphs (as they contain at most one adversarial edge). This approach also has a major caveat which has to do with the fact that the subgraph G_i is not necessarily two-edge connected, and might not even be connected. In the single edge case, Alg. DetComputeOneFTCycCov is indeed applied on the input graph which is 3 edge-connected.

Recall that Alg. DetComputeOneFTCycCov is based on performing a verification step of the cycles, at the end of which we have the guarantee that at most one edge, corresponding to the adversarial edge, might not be covered. The third step of that algorithm then covers this edge, in case needed, using its fundamental cycle in the BFS tree. When applying Alg. DetComputeOneFTCycCov on the subgraph G_i the situation is quite different. Since G_i is not necessarily connected, there might be potentially a large number of edges in G_i that are uncovered by cycles. Broadcasting the identities of these edges is too costly. For this reason, our algorithm applies the reduction in a more delicate manner.

Specifically, the algorithm applies Step (3) of Alg. DetComputeOneFTCycCov on the neighborhood cover of G_i (with radius of $O(fD)$). In addition, it attempts to cover with Alg. DetComputeOneFTCycCov only edges that lie on short cycles in G_i (i.e., to fix the issue that G_i is not two edge-connected). This guarantees that at most one edge activates the brute-force covering procedure in which the edge e get covered by a fundamental cycle of a BFS tree in $G \setminus \{e\}$. We next describe the algorithm in details.

Algorithm ComputeFTCycCov (Proof of Lemma 23). Let $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ be a $(L, 2f)$-covering subgraph family that is locally known to all the nodes (from Definition 14). The algorithm iterates over the subgraphs in $\mathcal{G}$. In phase i, the algorithm considers the subgraph $G_i \in \mathcal{G}$ and applies two major steps. Let $E_i = \{e = (u,v) \in G_i \mid \mathrm{dist}_{G_i \setminus \{e\}}(u,v) \leq L\}$ be the set of edges in G_i that are covered by a short cycle (of length at most $L+1$) in G_i [14]. During the i'th phase, the goal is to cover the edges in E_i. The first step considers covering the reliable edges in $E_i \setminus F^*$, and the second step considers the adversarial edges $F^* \cap E_i$. Note that the endpoints of an edge e does not necessarily know if it belongs to E_i.

Step (1): Covering non-adversarial edges in G_i. The algorithm employs the deterministic $(1,e)$-FT cycle cover algorithm DetComputeOneFTCycCov(G_i, L') of Lemma 22 on the subgraph G_i with diameter estimate $L' = O(L \cdot \log^c n)$, where c is the constant of Definition 10 (in the analysis part, it will be made clear why L' is set in this manner). When executing Alg. ComputeOneFTCycCov(G_i, L'), Step (3) of that algorithm which covers the adversarial edge is omitted. In addition, in the verification step of Alg. ComputeOneFTCycCov(G_i, L') (Step 2.6), instead of using the broadcast algorithm of [21] against a single adversarial edge, we use the broadcast algorithm of [21] against f adversarial edges over the original graph G (see Theorem 19). If during the execution of Alg. ComputeOneFTCycCov(G_i, L'), a node u receives an illegal message or that it needs to send too many messages through its incident edges (i.e., that exceeds the allowed $\widehat{O}(L'^2)$ congestion bound of Alg. ComputeOneFTCycCov(G_i, L')), it cancels the i'th iteration in the following sense. It omits all its cycles computed in the i'th phase, and remains silent until the next phase.

For a node u, let $\mathcal{C}_i(u)$ be the cycle collection obtained by u during ComputeOneFTCycCov(G_i, L'). Every node u that did not cancel the i'th phase, adds the cycles in $\mathcal{C}_i(u)$ to its final cycle collection $\mathcal{C}(u)$. Recall that the output of Alg. ComputeOneFTCycCov is given by a collection of tuples $\mathcal{C}_i(u) = \{(ID(C), C)\}$. At the end of Step (1), a node u considers its incident edge (u,v) as *i-handled* if there exists $(ID(C), C) \in \mathcal{C}_i(u)$ such that $(u,v) \in C$.

Step (2): Covering the adversarial edges in G_i. The goal of this step is to cover the adversarial edges of $E_i \cap F^*$. At the beginning of the step, the nodes locally compute a family of subgraphs $\mathcal{G}_i = \{G_{i,1}, \ldots, G_{i,\ell_i}\}$ of size $\ell_i = \widetilde{O}(L^2)$ using Claim 24 with parameter L. The algorithm then proceeds in ℓ_i iterations, where in each iteration j the nodes perform the following sub-steps over the communication subgraph $G_{i,j} \in \mathcal{G}_i$.

(2.1) Compute an L neighborhood-cover $\mathcal{S}_{i,j} = \{S_{i,j,1}, \ldots, S_{i,j,k_{i,j}}\}$ by applying Theorem 11, and let $T_{i,j,q}$ be the spanning tree of each node subset $S_{i,j,q}$.

(2.2) An edge (u,v) is *short bridgeless* if (i) (u,v) is not i-handled in Step (1), and (ii) there exists a tree $T_{i,j,q}$ containing u and v. For every short bridgeless edge e, the algorithm adds a cycle $C_e = \pi(u,v) \circ e$ to the cycle collection, where $\pi(u,v)$ is a u-v path in $T_{i,j,q}$.

[14] Note that the set E_i is unknown to the nodes in G.

If during the execution of this step, a node u detects an incident edge with a congestion above the limit, it omits all the cycles obtained in this step from its cycle collection $\mathcal{C}(u)$ and proceeds to the next sub-iteration.

The correctness analysis is deferred to the full version.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

2 Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Fast distributed network decompositions and covers. *J. Parallel Distributed Comput.*, 39(2):105–114, 1996.

3 Baruch Awerbuch, Oded Goldreich, David Peleg, and Ronen Vainish. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.

4 Piotr Berman, Krzysztof Diks, and Andrzej Pelc. Reliable broadcasting in logarithmic time with byzantine link failures. *Journal of Algorithms*, 22(2):199–211, 1997.

5 Piotr Berman and Juan A. Garay. Cloture votes: $n/4$-resilient distributed consensus in $t+1$ rounds. *Math. Syst. Theory*, 26(1):3–19, 1993.

6 Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October – 1 November 1989*, pages 410–415. IEEE Computer Society, 1989.

7 Greg Bodwin, Michael Dinitz, and Caleb Robelle. Optimal vertex fault-tolerant spanners in polynomial time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 2924–2938, 2021.

8 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

9 Diptarka Chakraborty and Keerti Choudhary. New extremal bounds for reachability and strong-connectivity preservers under failures. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 25:1–25:20, 2020.

10 Ran Cohen, Iftach Haitner, Nikolaos Makriyannis, Matan Orland, and Alex Samorodnitsky. On the round complexity of randomized byzantine agreement. In *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, pages 12:1–12:17, 2019.

11 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 169–178. ACM, 2011.

12 Danny Dolev. The byzantine generals strike again. *J. Algorithms*, 3(1):14–30, 1982.

13 Danny Dolev, Michael J. Fischer, Robert J. Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982.

14 Danny Dolev and Ezra N. Hoch. Constant-space localized byzantine consensus. In *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*, pages 167–181, 2008.

15 Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

16 Michael J Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *International Conference on Fundamentals of Computation Theory*, pages 127–140. Springer, 1983.

17 Mattias Fitzi and Ueli Maurer. From partial consistency to global broadcast. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 494–503, 2000.

18 Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM J. Comput.*, 27(1):247–290, 1998.

19 Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21–23, 2015*, pages 3–12, 2015.

20 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Transactions on Algorithms (TALG)*, 16(1):1–25, 2019.

21 Yael Hitron and Merav Parter. Broadcast CONGEST algorithms against adversarial edges. In *Distributed Computing – 29th International Symposium, DISC 2021*, 2021.

22 Yael Hitron and Merav Parter. Broadcast CONGEST algorithms against adversarial edges. *CoRR*, abs/2004.06436, 2021. arXiv:2004.06436.

23 Damien Imbs and Michel Raynal. Simple and efficient reliable broadcast in the presence of byzantine processes. *arXiv preprint*, 2015. arXiv:1510.06882.

24 Karthik C. S. and Merav Parter. Deterministic replacement path covering. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 704–723. SIAM, 2021.

25 Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

26 Muhammad Samir Khan, Syed Shalan Naqvi, and Nitin H. Vaidya. Exact byzantine consensus on undirected graphs under local broadcast model. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 – August 2, 2019*, pages 327–336, 2019.

27 Chiu-Yuen Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 275–282, 2004.

28 Frank Thomson Leighton, Bruce M Maggs, and Satish B Rao. Packet routing and job-shop scheduling ino (congestion+ dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

29 Alexandre Maurer and Sébastien Tixeuil. On byzantine broadcast in loosely connected networks. In *Distributed Computing – 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, pages 253–266, 2012.

30 Merav Parter. Small cuts and connectivity certificates: A fault tolerant approach. In *33rd International Symposium on Distributed Computing (DISC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

31 Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

32 Merav Parter and Eylon Yogev. Distributed algorithms made secure: A graph theoretic approach. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1693–1710, 2019.

33 Merav Parter and Eylon Yogev. Low congestion cycle covers and their applications. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1673–1692, 2019.

34 Merav Parter and Eylon Yogev. Optimal short cycle decomposition in almost linear time. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 89:1–89:14, 2019.

35 Merav Parter and Eylon Yogev. Optimal short cycle decomposition in almost linear time. http://www.weizmann.ac.il/math/parter/sites/math.parter/files/uploads/main-icalp-cycles-full.pdf, 2019.

36 Merav Parter and Eylon Yogev. Secure distributed computing made (nearly) optimal. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 – August 2, 2019*, pages 107–116, 2019.

37 Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

38 Andrzej Pelc and David Peleg. Broadcasting with locally bounded byzantine faults. *Inf. Process. Lett.*, 93(3):109–115, 2005.

39 David Peleg. *Distributed Computing: A Locality-sensitive Approach.* SIAM, 2000.

40 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363. ACM, 2020.

41 Nicola Santoro and Peter Widmayer. Time is not a healer. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 304–313. Springer, 1989.

42 Nicola Santoro and Peter Widmayer. Distributed function evaluation in the presence of transmission faults. In *Algorithms, International Symposium SIGAL '90, Tokyo, Japan, August 16-18, 1990, Proceedings*, pages 358–367, 1990.

43 Sam Toueg, Kenneth J Perry, and TK Srikanth. Fast distributed agreement. *SIAM Journal on Computing*, 16(3):445–457, 1987.

44 Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Transactions on Algorithms (TALG)*, 9(2):1–13, 2013.