General CONGEST Compilers against Adversarial Edges

Yael Hitron
Weizmann Institute
yael.hitron@weizmann.ac.il

Merav Parter
Weizmann Institute
merav.parter@weizmann.ac.il

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.

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

Contents

Theorem 1. *blablabla*

Proof. lalala □

In Theorem 1

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 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 $\mathsf{CONGEST}$ 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 $\mathsf{CONGEST}$ model. [21] also formalized the adversarial $\mathsf{CONGEST}$ model, which is the model that we consider in this work as well.

The Adversarial $\mathsf{CONGEST}$ 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

[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].

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:

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 cover* 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 themselves 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 FT cycle covers in 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 crucial for the compilation task to reliably simulate the message exchange over these edges in the given fault-free algorithm $\mathcal{A}$. Upon computing a FT cycle cover with parameters c, d, we then present a round-by-round compiler whose overhead depends on the c, d parameters. To optimize the round overhead, we exploit (our modified) FT cycle cover in a somewhat more delicate manner compared to that of [33], leading to an improvement by a factor of $O(D^2)$ rounds for a single adversarial edge.

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 an algorithm given the FT cycle cover. We start by describing our contribution w.r.t the combinatorial characterization of FT cycle covers, and then 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 $c + d$. In order to provide reliable computation in the presence of f adversarial edges F^*, it is desired to cover each edge by multiple short cycles of small overlap. This motivates the following definition.

Definition 2 (f-FT Cycle Cover)**.** *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.*

[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.

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], we show the following in Appendix A.

Lemma 3 (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 $c = \widehat{O}(f(5fD)^f)$ and $d = \widehat{O}(fD)$.*

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 4 (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 $D^{\Omega(f)}$ of our compilers. It also provides an explanation for the natural barrier of $D^{\Omega(f)}$ rounds for handling f adversarial edges in the distributed 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 a length of $(D/f)^{\Omega(f)}$ edges. Theorem 4 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 a $(2f)$-FT cycle cover as defined by Definition 2. 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 5 ((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 contains an f-FT cycle cover, and therefore the lower bound of Theorem 4 also holds for (f, F^*)-FT cycle covers. When $F^* = \{e'\}$, we slightly abuse notation and simply write (f, e')-FT cycle cover. Our FT cycle covers should be

[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.

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.

Theorem 6. *[$(1, e')$-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 $(1, e')$-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, a full description of these cycles. The key challenge in proving Theorem 6 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 7. *(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 6). 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,

[8] This assumption can be omitted using the broadcast algorithms of [21, 22], in the case where the nodes have a designated marked leader.

we use the notion of (f, F^*)-FT cycle covers. Our first contribution is the construction of (f, F^*)-FT cycle covers in the adversarial CONGEST model. Due to technicalities that arise 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 a (f, F^*)-FT cycle cover* where the asterisk indicates the possible existence of truncated paths in the distributed output.

Definition 8 ((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 9 ((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 4, 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 7 to handle f adversarial edges.

Theorem 10 (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] of 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 of 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 algorithm 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

[9] To formalize this argument, we provide a formal definition for the cut value of a u-v walk collection.

subgraph $G' \subseteq G$), as it is required to exchange messages in a reliable manner between 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 $\text{poly}(\log n)$ factors, and the term $\widehat{O}(\cdot)$ hides $2^{O(\sqrt{\log n})}$factors[10].

Definition 11 (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 12 (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 $\widetilde{O}(r)$ $\mathsf{CONGEST}$ 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 $\mathsf{CONGEST}$ 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 13. *[[34, 33]] There is a randomized algorithm $\mathsf{ComputeCycCov}(G, D')$ 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. Alg. $\mathsf{ComputeCycCov}(G, D')$ 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 for Alg. $\mathsf{ComputeCycCov}$ 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 12, see Appendix B for the proof of the following.

Observation 14. *The algorithm $\mathsf{ComputeCycCov}(G_i, D')$ of Fact 13 can be made deterministic using the neighborhood covering algorithm of Theorem 12. 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.*

[10] The latter factors arise by the (fault-free) distributed computation of cycle covers by [34].

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 15 ((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] $\mathrm{dist}_{G \setminus F \cup \{e\}}(u,v) \leq L$, *there exists a subgraph G_i satisfying that (P1)* $\mathrm{dist}_{G_i \setminus (F \cup \{e\})}(u,v) \leq L$, *and (P2) $(F \cup \{e\}) \cap G_i = \{e\}$.*

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

Observation 16 (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* $\mathrm{dist}_{G \setminus F}(u,v) \leq 2(|F|+1) \cdot D + |F|$.

We note that by Observation 16, if G is $(t+1)$ edge-connected, then a $(5tD, t)$ covering 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 17 (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 18 ([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 19 ([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.*

[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 most L covering e, rather than any L-length u-v path.

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

Theorem 20 ([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 19 also implies a leader election algorithm. For completeness the proof of the following claim is given in Appendix A.

Claim 21. *[Byzantine Leader Election] 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* AdvBroadcast *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 in the graph. Then, we describe how to compile a single round using these cycles.

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

This section is mainly devoted to showing the following key lemma that computes a $(1,e')$-FT cycle cover, given a locally known covering family.

Lemma 22. *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 22. 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 14 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

$$e' \notin G_i \ , \ e \in G_i \text{ and } \mathrm{dist}_{G_i \setminus \{e\}}(u,v) \leq L \ .$$

Since, $\mathrm{dist}_{G \setminus \{e,e'\}}(u,v) \leq L$, 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 22).** 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. Every iteration i consists of two steps. In the first step, the nodes apply Alg. ComputeCycCov(G_i, L) of Observation 14 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 and does not contain e'. The third step of the algorithm handles the remaining adversarial edge, in case needed. We next elaborate on these steps in more detail. 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 ComputeCycCov of Observation 14 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. 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 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);

[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 $\mathcal{C}'_i(u)$.

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 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 clockwise 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 in 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 on 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 19(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 did not accept a cancellation message $cancel(i)$ via the broadcast algorithm in Step (2.5). 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 as *handled* 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 unhandled edges. Every node u and an unhandled edge (u, v), broadcasts the edge (u, v) using the deterministic broadcast algorithm of Theorem 19(1). In the analysis section, we show that if there is an unhandled 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 $w \in C$.

Correctness. We begin with showing that the tuples $\{\mathcal{C}(u)\}_{u \in V}$ computed in Step (2) induce legal cycles in G. That is, for every iteration i, a node $u \in V$, and every tuple $(ID(C), C) \in \mathcal{C}_i(u)$, we show that C is a cycle in G and $(ID(C), C) \in \mathcal{C}_i(w)$ for every $w \in C$. As we will see, the fact that the output of the algorithm induces (real) cycles will play a critical role in showing the adversarial edge is covered by a short cycle. We start with the following observation.

Observation 23. *Any maximal walk along which a message $ver(C') = (ID(C'), ID(v_{C'}), ver)$ is propagated towards some node w, either starts at $v_{C'}$ or at the adversarial edge e'. In addition, the walk must be a simple path.*

Proof of Observation 23. Since the message $ver(C')$ contains the identifier of the node $v_{C'}$, by Step (2.1) of the verification step, no other node but $v_{C'}$ initiates the verification message $ver(C')$. Hence, we can conclude that if the walk is not initiated by $v_{C'}$, it is initiated by the adversarial

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

edge e'. Let P be the maximal walk along a message $ver(C')$ is propagated towards w. We next show P is a simple path. Assume by contraction there exists a node $v \in P$ with degree at least three in P. This contradicts Step (2.2), as for every node v, the number of neighbors in $N(v)$ which communicate the message $ver(C')$ with v is at most two. □

Simple cycles. Recall that the adversarial edge is denoted by $e' = (v_1, v_2)$. A path P is denoted as *reliable* if it consists of only reliable edges (i.e., $e' \notin P$). For an $a \rightsquigarrow b$ directed path P, we denote the inverse path i.e., going from b to a, by $\bar{P}$. Fix an iteration i, and consider a node $u^* \in V$, and a tuple $(ID(C), C) \in \mathcal{C}_i(u^*)$. Since $(ID(C), C) \in \mathcal{C}_i(u^*)$, in Step (2) the node u^* received the verification message $ver(C) = (ID(C), ID(v_C), ver)$ from both neighbors on C denoted as w_1 and w_2, where v_C is the node with the highest ID in C. Let $P_1 = [u'_1, \ldots, u'_{k'} = u^*]$ and $P_2 = [u_1, \ldots, u_k = u^*]$ be the maximal directed walks on which the message $ver(C) = (ID(C), ID(v_C), ver)$ arrived to u^*. Let $\widehat{C} = P_1 \circ \bar{P}'_2$, where $P'_2 = P_2$ if $u_1 \neq v_1, v_2$, and $P'_2 = [u_2, \ldots, u_k]$ otherwise. That is, if both P_1 and P_2 start with the adversarial edge e', then e' in included in $\widehat{C}$ only once.

Our proof structure is as follows. We first show that $\widehat{C}$ is a simple cycle. Then we show that $\widehat{C} = C$ and that in addition, every vertex $w \in C$, includes $(ID(C), C)$ in the collection of its verified cycles $\mathcal{C}_i(w)$. The proof of the next claim is quite technical, and is deferred to Appendix C.

Claim 24. *$\widehat{C}$ is a simple cycle.*

Claim 25. *At the end of the first step, it holds that $(ID(C), \widehat{C}) \in \mathcal{C}'_i(w)$ for every $w \in \widehat{C}$. In particular, it holds that $\widehat{C} = C$.*

Proof. By Claim 24, $\widehat{C}$ is a simple cycle. If $e' \notin \widehat{C}$ all nodes obtain the correct cycle description, and the claim follows. Next assume $e' \in \widehat{C}$, and let $w \in \widehat{C}$. Recall that in Alg. ComputeCycCov the cycle description is obtained in both directions. As all nodes in $\widehat{C}$ are active at the beginning of the verification step, w received the same cycle description from both its neighbors on $\widehat{C}$.

Denote $\widehat{C}$ by $\widehat{C} = C_1 \circ (v_1, v_2) \circ C_2$ where C_1 is a reliable w-v_1 path, and C_2 is a reliable v_2-w path. Hence, when obtaining the cycle description during Alg. ComputeCycCov(G_i, L), w received $Q_1 \circ (v_1, v_2) \circ C_1$ over C_1, and $Q_2 \circ (v_2, v_1) \circ C_2$ over C_2, for some paths Q_1, Q_2. Since w is active at the beginning of the verification step, w received *the same* cycle description in both directions, concluding that $Q_1 = C_2$ and $Q_2 = C_1$. It follows that $(ID(C), \widehat{C}) \in \mathcal{C}'_i(w)$.

In particular, for the node u^* it holds that $(ID(C), \widehat{C}) \in \mathcal{C}'_i(u^*)$. Since u^* is active at the beginning of the verification step, it obtained a single cycle with identifier $ID(C)$, and therefore $C = \widehat{C}$. □

We now show that every node $w \in C$ adds the tuple $(ID(C), C)$ to its *verified* cycle collection $\mathcal{C}_i(w)$. Towards that goal, we start with the following auxiliary claim.

Claim 26. *Any cancellation message $cancel(C)$ received in Step (2.4) by some node $w \in C$ is dropped.*

Proof. Assume towards contradiction a node $w \in C$ received a $cancel(C)$ message during Step (2.4) that it did not drop. Consider the following cases:

- The message $cancel(C)$ received by w was initiated by v_C. In this case, we will show that also u^* (namely, the vertex that included $(ID(C), C)$ in its verified cycle set) received (and sent) the cancellation message as well, in contradiction to the assumption that the cycle is included in the output of u^*. In this case, the paths P_1 and P_2 start at v_C, that is, $P_1 = C[v_C, u^*]$ and $P_2 = C[u^*, v_C]$. Since C is simple, at least one of them, say P_2, does not contain the adversarial edge e'. According to Step (2.3), v_C sent the message $cancel(C)$ in both directions in the super-round τ_i. By Claim 25, all nodes in P_2 hold the correct cycle description of C, and the correct distances from v_C. As P_2 does not contain the adversarial edge, using an inductive argument it follows that every $v \in P_2$ at distance d^2_v from v_C on P_2, receives the message $cancel(C)$ in super-round $\tau_i + d^2_v$. According to Step (2.4), v also sends the message in super-round $\tau_i + d^2_v + 1$. It follows that u^* received the cancellation message $cancel(C)$ from the second direction on C (namely, along P_2), in super-round $\tau_i + d^2_{u^*}$, leading to a contradiction.

- The message $cancel(C)$ received by w was initiated by the adversarial edge $e' \in C$. We first show that v_C also received the message $cancel(C)$ by the end of Step (2.4). W.l.o.g assume that w received the message in direction 1 (clockwise). Since w did not drop the message, it follows that w received the message in super-round $\tau_i + d^1_w$, where d^1_w is the distance between v_C and w in direction 1. As all nodes in C hold the correct distances from v_C (Claim 25), using an inductive argument on the reliable w-v_C path $C[w, v_C]$, we conclude that v_C received the message $cancel(C)$ in super-round $\tau_i + |C| \leq \tau_i + R$ from direction 1.

  It remains to consider two cases.

  Case 1: v_C received the message $cancel(C)$ also from the second direction. In such a case, u^* also received the cancellation message, did not drop it, and sent it along the cycle. This contradicts the assumption that the cycle is included in the verified cycle set of u^*.
  Case 2: v_C received the cancellation message $cancel(C)$ only from a single direction (i.e., from one of its neighbors on C). According to Step (2.5), in this case v_C broadcasts a cancellation message $cancel(i)$ to all nodes. By the correctness of the broadcast algorithm, in Step (2.6), the node u^* cancels all its cycles defined in this iteration, leading to a contradiction.

□

Claim 27. *For every $w \in C$ it holds that $(ID(C), C) \in \mathcal{C}_i(w)$.*

Proof. Recall that a node $w \in C$ adds a tuple $(ID(C), C)$ to its cycle collection $\mathcal{C}_i(w)$, only if (i) it received the verification message $ver(C)$ from both its neighbors on C, (ii) any cancellation message $cancel(C)$ received has been dropped, and (iii) it did not accept a cancellation message $cancel(i)$ via the broadcast algorithm in Step (2.5). We will show that all these conditions hold for w and C. Condition (ii) holds by Claim 26. As for condition (iii), as shown by [21] (Corollary 14), no node accepts a false message initiated by the adversarial edge. Hence, if w accepts a message $cancel(i)$, it is initiated by some node in the graph. Since u^* did not accept such a message, by the correctness of the broadcast algorithm, w did not accept such a message as well. We are left to show condition (i) holds, and w received the verification message from both neighbors on C.

By Claim 25, it holds that $v_C \in C$. Let $P' = C[v_C, w]$ and $P'' = C[w, v_C]$ be the two v_C-w paths on C. Since C is simple, at least one of them say $P'' = C[w, v_C]$, does not contain the adversarial edge e'. By Step (2.1), at the beginning of Step (2), v_C sent $ver(C)$ to both neighbors on C. As a result, w received the message $ver(C)$ over the reliable path $\overline{P''}$. Additionally, as v_C did not send a cancellation message, v_C also received the message $ver(C)$ back over the path P''. Thus, it holds that w sent the message to its neighbor in P'', and therefore w also received the message from its other neighbor in P'. □

Coverage. We begin by noting that in case the adversarial edge e' is not contained in a subgraph G_i, the cycles $\mathcal{C}_i'$ obtained by Alg. $\mathsf{ComputeCycCov}(G_i, L)$ are legal cycles in G, with valid cycle descriptions. That is, for every node u and a cycle identifier $ID(C)$ such that $(ID(C), C) \in \mathcal{C}_i'(u)$, for every $w \in C$ it holds that $(ID(C), C) \in \mathcal{C}_i'(w)$. Additionally, we observe that all cycles in $\mathcal{C}_i'$ pass the verification step, and therefore for every node $u \in V$ and a cycle $(ID(C), C) \in \mathcal{C}_i'(u)$, it holds that $(ID(C), C) \in \mathcal{C}_i(u)$.

Observation 28. *For a subgraph G_i such that $e' \notin G_i$, for every node $u \in V$ and a tuple $(ID(C), C) \in \mathcal{C}_i'(u)$, it holds that $(ID(C), C) \in \mathcal{C}_i(u)$.*

Proof. As $e' \notin G_i$, by the properties of Alg. $\mathsf{ComputeCycCov}$, the cycle collection $\mathcal{C}_i'$ admits the desired length and edge congestion. Therefore, all nodes in G are active at the beginning of the verification step. In consequence, by Step (2.1), the node with the highest ID in C denoted as v_C initiates the verification step and sends $ver(C)$ to both neighbors on C. Next, according to Step (2.2), all nodes on C send the verification messages in both directions. As a result, all nodes in C receive the verification message $ver(C)$ from both neighbors. As v_C also receives the message $ver(C)$ back from both neighbors on C, no cancellation message is sent and u adds the tuple $(ID(C), C)$ to its cycle collection $\mathcal{C}_i(u)$. □

An edge $(u, v) \in E$ is said to be covered by the algorithm, if there exist a cycle C with ID $ID(C)$ such that $C \cap \{e, e'\} = \{e\}$, and $(ID(C), C) \in \mathcal{C}_i(w)$ for every $w \in C$. By the covering property of the graph family $\mathcal{G}$, we can now conclude that all reliable edges are covered.

Claim 29. *Every reliable edge $e \neq e'$ is covered by a cycle of length $\widehat{O}(L)$.*

Proof of Claim 29. By the promise of Lemma 22, for every reliable edge $e = (u, v)$, $\text{dist}_{G \setminus \{e, e'\}}(u, v) \leq L$. Hence, due to the covering property of $\mathcal{G}$, there exists a subgraph G_i containing all the edges of an L-length u-v path $P \subseteq G \setminus \{e, e'\}$, and in addition $G_i \cap \{e', e\} = \{e\}$. Thus, by the properties of $\mathsf{ComputeCycCov}(G_i, L)$, there exists a cycle $C \subseteq G_i$ of length $\widehat{O}(L)$, such that $(ID(C), C) \in \mathcal{C}_i'(u)$, and $e \in C$. Moreover, as $\mathsf{ComputeCycCov}(G_i, L)$ is executed correctly on G_i, for every $w \in C$ it holds that $(ID(C), C) \in \mathcal{C}_i'(w)$. By Observation 28 it then follows that every node $w \in C$ adds the tuple $(ID(C), C)$ to $\mathcal{C}_i(w)$. Hence, e is covered by the end of the i'th iteration. □

Claim 30. *The adversarial edge $e' = (v_1, v_2)$ is covered by a cycle of length $\widehat{O}(L)$.*

Proof of Claim 30. First assume that at the beginning of Step (3) at least one of the nodes v_1 or v_2, say v_1 considered the edge (v_1, v_2) as handled. That is, at the beginning of Step (3) $(ID(C), C) \in \mathcal{C}(v_1)$ for some cycle C for which $(v_1, v_2) \in C$. By Claim 24, Claim 25 and Claim 27, it follows that e' is indeed covered.

Next, assume that at the beginning of Step (3) both v_1 and v_2 consider the edge e' as unhandled, and assume v_1 has a higher ID than v_2. According to Step (3), v_1 and v_2 broadcast the identifier of the edge e' using the broadcast algorithm of [21]. By Claim 29 all reliable edges are covered during Step (2) of the algorithm. Hence, e' is the only uncovered edge, and the only message that is broadcast. By the properties of the broadcast algorithm, it must be initiated by a node and cannot be initiated by the adversarial edge. Therefore, all nodes in V accept a single message containing the identifier of e'.

Next, v_1 initiates a construction of a BFS tree T rooted at v_1 in $G \setminus \{e'\}$. Because we assumed that for every edge $e = (u, v)$ it holds that $\mathrm{dist}_{G\setminus\{e\}}(u, v) \leq L$, the tree T is of depth $O(L)$, and contains a v_1-v_2 path P of size $O(L)$. Additionally, since all edges participating in the tree construction are reliable, the cycle $C = P \circ (v_1, v_2)$ covering e', is added to the cycle collection of all nodes in C. □

Congestion. For an iteration i, let $\mathcal{C}_i$ be the cycle collection obtained during the i'th iteration. We show the congestion of the cycles in $\mathcal{C} = \bigcup_i \mathcal{C}_i$ is at most $\widehat{O}(\ell)$.

Claim 31 (Congestion)**.** *For an iteration i each edge appears on $\widehat{O}(1)$ cycles in $\mathcal{C}_i$. Consequently, each edge appears on $\widehat{O}(\ell)$ cycles in $\mathcal{C} = \bigcup_{i=1}^{\ell} \mathcal{C}_i$.*

Proof of Claim 31. For an iteration i. First note that if $e' \notin G_i$, all edges participating in the i^{th} iteration are reliable. Hence, due to the properties of Alg. ComputeCycCov, all cycles constructed during the execution of ComputeCycCov(G_i, L) are legal, and the congestion of each edge is $\widehat{O}(1)$. For an iteration i such that $e' \in G_i$, if an endpoint of an edge e detects high congestion in the cycles constructed by ComputeCycCov(G_i, L), it omits all cycles and becomes inactive. Hence, for each node u and an incident edge e, after the i'th iteration u outputs at most $\widehat{O}(1)$ cycles covering e. Because we have ℓ iterations, the total congestion of each edge is $\widehat{O}(\ell)$. □

We are now ready to prove Lemma 22.

Proof of Lemma 22. The covering property holds by Claim 29 and Claim 30. The congestion arguments hold by Claim 31. The length bound holds immediately, as all long cycles defined in bad iterations are omitted. It remains to bound the round complexity.

For each iteration i, performing Alg. ComputeCycCov(G_i, L) required $\widehat{O}(L)$ rounds. During the verification step, the congestion on each edge is kept bound by $\widehat{O}(1)$. In addition, since the length of each cycle is $\widehat{O}(L)$, steps (2.1)-(2.5) can be performed in $\widehat{O}(L)$ rounds. As for Step (2.6), in case some leaders broadcast a cancellation message *cancel*(i) during iteration i, as all nodes broadcast the same message, we can view this step as performing a single execution of the broadcast algorithm of [21] in $\widetilde{O}(D^2) = \widetilde{O}(L^2)$ rounds.

Regarding the third step, the broadcast algorithm requires $\widetilde{O}(L^2)$ rounds. Constructing a BFS tree and performing the upcast and downcast steps requires $O(L)$ rounds. We conclude that the total round complexity of the algorithm is $\widehat{O}(L^2 \cdot \ell)$. □

We also show that if our graph G is not 3 edge-connected or with bounded diameter, our cycle cover algorithm has the guarantee to cover every reliable edge that lies on a reliable short cycle in G. That is, we achieve the following.

Corollary 32. *There exists a deterministic algorithm* 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 $e' \notin C$ and $(u, v) \in C$.*

Proof of Theorem 6. Combining Lemma 22 with the deterministic construction of Fact 18 with $L = 7D$, results in a deterministic $(1, e')$-FT cycle cover algorithm, with the following parameters.

Corollary 33. *There exists an r-round* deterministic *algorithm* DetComputeOneFTCycCov *for computing a $(1, e')$-FT cycle cover with parameters $d = \widehat{O}(D), c = \widehat{O}(D^2)$ and $r = \widehat{O}(D^4)$.*

As for the randomized algorithm, we begin with construction a $(L, 1)$ covering family using the following lemma that follows by [21] and [24], see Appendix C for the proof.

Lemma 34. *Given a 3 edge-connected graph G of diameter D with a fixed unknown adversarial edge e', one can compute in $\widetilde{O}(D^2)$ rounds a $(L = 7D, 1)$ covering family $\mathcal{G}$ of size $\ell = O(D \log n)$. The family $\mathcal{G}$ satisfies the covering properties w.h.p.*

To reduce the round complexity of Alg. ComputeOneFTCycCov, in Step (2.6) of the verification step, we use the nearly optimal randomized broadcast algorithms of [21] that work given that all nodes share a random seed of size $\widetilde{O}(1)$. In particular, we use the broadcast algorithm of Theorem 19(2). To share this random seed, the algorithm begins with applying the randomized leader election algorithm of Claim 21 which takes $\widetilde{O}(D^2)$ rounds. Given a leader s, it shares a random seed r of $\widetilde{O}(1)$ bits by using the *deterministic* broadcast algorithm of Theorem 19(1) within $\widetilde{O}(D^2)$ rounds. From that point, the nodes have a shared seed and the future broadcast procedures will be based on that. We note that since the shared seed is only used to define the covering family, we can re-use the same shared seed in all future applications of the broadcast algorithm.

From now on, the randomized algorithm is exactly the same as Alg. ComputeOneFTCycCov, only that thanks to the shared seed, it uses the randomized broadcast algorithm of Theorem 19(2). This broadcast algorithm works in $\widetilde{O}(D)$ rounds. As it is applied ℓ many times, the round complexity is bounded by $\widehat{O}(\ell \cdot L + \ell \cdot D + L^2)$. The proof of Theorem 6 is completed by $\ell = \widetilde{O}(D)$ (see Lemma 34) and $L = 7D$.

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 7). 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

3.1 (f, F^*)-FT cycle cover*

At the heart of the compilers lies the construction of a (f, F^*)-FT cycle cover* in the adversarial CONGEST model that we describe in this section. Our main result is a deterministic construction of (f, F^*)-FT cycle covers* in the adversarial CONGEST model.

Lemma 35. *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 9 follows by combining Lemma 35 and Fact 18. Alg. ComputeFTCycCov uses an $(L, 1)$ covering family $\mathcal{G}$ with slightly different properties than those provided in Definition 15. Specifically, we use the following fact from [24].

Claim 36 ([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) Alg. ComputeCycCov from Fact 13 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 5. 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 edge $e' \in F^*$: 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 Corollary 32) 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 3 edge-connected and might not even be connected. In the single edge case, Alg. DetComputeOneFTCycCov is indeed applied on the input graph that 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 a radius of $O(fD)$).

Algorithm ComputeFTCycCov (Proof of Lemma 35). 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 15). 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 $E_i = \{e = (u, v) \in G_i \mid \text{dist}_{G_i \setminus \{e\}}(u, v) \leq L\}$ 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 applies the deterministic $(1, e)$-FT cycle cover Alg. DetComputeOneFTCycCov(G_i, L') of Corollary 32 on the subgraph G_i with diameter estimate $L' = O(L \cdot \log^c n)$, where c is the constant of Definition 11 (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 20). 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

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

of Alg. $\mathsf{ComputeOneFTCycCov}(G_i, L')$), 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 $\mathsf{ComputeOneFTCycCov}(G_i, L')$. 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. $\mathsf{ComputeOneFTCycCov}$ 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 a tuple $(ID(C), C) \in \mathcal{C}_i(u)$ 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 36 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 12, 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 $T_{i,j,q}$. If during the execution of this step, a node u detects an incident edge with 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.

Correctness. We first show the output collection admits the desire congestion and length bounds. The algorithm proceeds in ℓ phase. In each phase, all cycles obtained in Step (1) are of length $\widehat{O}(L)$, and the cycles obtained in Step (2) are of length $\widetilde{O}(L)$. Since every node u that detects an edge e with congestion above the limit omits the cycles computed in that iteration, each edge participates in at most $\widehat{O}(L^2)$ cycles. Hence, the total edge congestion is $\widehat{O}(\ell \cdot L^2)$.

Coverage. For an edge $e \in E$ and a subset $E' \subseteq E$ of size $|E'| \leq (f-1)$, the tuple (e, E') is *covered*, if there exists a cycle C with a cycle identifier $ID(C)$ satisfying $C \cap (F^* \cup E' \cup \{e\}) = \{e\}$, and $(ID(C), C) \in \mathcal{C}(w)$ for every $w \in C$. An edge $e \in E$ is covered, if for every subset $E' \subseteq E$ of size $|E'| \leq (f-1)$, the tuple (e, E') is covered.

By the definition of an (f, F^*)-FT cycle cover*, we are left to show that all edges in E are covered. Given an edge $e = (u, v)$ and a subset $E' \subseteq E$ of size $|E'| \leq (f-1)$, a subgraph $G_i \in \mathcal{G}$ is *good* for the tuple (e, E') if (1) $(E' \cup F^* \cup \{e\}) \cap G_i = \{e\}$, and (2) $\mathrm{dist}_{G_i \setminus \{e\}}(u, v) \leq L'$. We start with showing that for every tuple (e, E') there exists a good subgraph $G_i \in \mathcal{G}$.

Claim 37. *For every edge $e = (u, v)$ and a subset $E' \subseteq E$ of size $|E'| \leq (f-1)$, there exists a good subgraph $G_i \in \mathcal{G}$. Moreover, if a subgraph G_i is good for some tuple (e, E'), all cycles obtained during the execution of* $\mathsf{ComputeOneFTCycCov}(G_i, L')$ *in Step (1) are legal cycles in G with length $\widehat{O}(L)$ and edge congestion $\widehat{O}(L^2)$.*

Proof. By Observation 16, for a $(2f+1)$ edge-connected graph G with diameter D and $L = 7fD$, for every edge $(u, v) = e \in E$, and every set $F \subseteq E$ of size $|F| \leq (f-1)$ it holds that $\mathrm{dist}_{G \setminus \{F \cup F^* \cup \{e\}\}}(u, v) \leq L$. Since $|E'| \leq (f-1)$, it holds that $\mathrm{dist}_{G \setminus (E' \cup F^* \cup \{e\})}(u, v) \leq L$. Hence, as

$|F^* \cup E'| \leq (2f-1)$, by the covering property of the $(2f, L)$-covering family $\mathcal{G}$, there exists a good subgraph G_i satisfying properties (1) and (2). Next, consider a subgraph $G_i \in \mathcal{G}$ that is good for some tuple (e, E'). Since $F^* \cap G_i \subseteq \{e\}$, the subgraph G_i contains at most one adversarial edge.

Recall that in Step (2.6) of the execution ComputeOneFTCycCov(G_i, L'), if a leader receives a cancellation message only from one direction, it broadcasts a cancellation message $cancel(i)$ using the broadcast algorithm of [21] against f adversarial edges, over the graph G. By Theorem 20 it follows that in such a case, all nodes in G will accept the message. The claim then follows from Theorem 6 and the properties of Alg. ComputeOneFTCycCov. □

We next show that all the reliable edges in G are covered. Towards that goal, we show that for an edge $e \notin F^*$ and a subset E', if a subgraph G_i is good for a tuple (e, E'), then (e, E') is covered by the end of the i'th phase.

Claim 38. *For an edge $e \notin F^*$ and a subset $E' \subseteq E$ of size $|E'| \leq (f-1)$, let $G_i \in \mathcal{G}$ be a good subgraph for (e, E'). Then after Step (1) of the i'th phase (e, E') is covered.*

Proof. Since G_i is good for (e, E'), $\text{dist}_{G_i \setminus \{e\}}(u, v) \leq L'$. Additionally, by Claim 37 the execution of ComputeOneFTCycCov(G_i, L') is performed correctly, and no node in V cancels the i'th phase. As e is reliable, by Corollary 32 we conclude that there exists a cycle C with cycle identifier $ID(C)$, such that $e \in C$, and $(ID(C), C) \in \mathcal{C}_i(w)$ for every $w \in C$. As no node cancels the i'th phase, w also adds the tuple $(ID(C), C)$ to its output set $\mathcal{C}(w)$. Finally, as $(E' \cup F^* \cup \{e\}) \cap G_i = \{e\}$ and $C \subseteq G_i$, then $(E' \cup F^* \cup \{e\}) \cap C = \{e\}$. The claim follows. □

Corollary 39 (Covering the Reliable Edges)**.** *Every edge $e \notin F^*$ is covered.*

Proof. For an edge $e \notin F^*$, and a subset $E' \subseteq E$ of size $|E'| \leq (f-1)$, by Claim 37 there exists a subgraph $G_i \in \mathcal{G}$ that is good for (e, E'). Hence, by Claim 38 the tuple (e, E') is covered by the end of the i'th phase. □

Claim 40 (Covering the Adversarial Edges)**.** *Every* adversarial *edge $e = (u, v) \in F^*$ is covered.*

Proof. Fix a subset $E' \subseteq E$ of size $|E'| \leq (f-1)$. We will show that (e, E') is covered. By Claim 37 there exists a subgraph $G_i \in \mathcal{G}$ that is good for the tuple (e, E'). First assume that for at least one of the nodes u or v, say v, at the end of Step (1) the edge e is i-handled. By Claim 37 the execution of ComputeOneFTCycCov(G_i, L') is performed correctly, and therefore (e, E') is covered by the end of Step (1) of the i'th phase.

Next, assume that at the end of Step (1), both v and u considers the edge e as not i-handled. Let $G_{i,j} \in \mathcal{G}_i$ be a subgraph of G_i satisfying that (P1) $\text{dist}_{G_{i,j} \setminus \{e\}}(u, v) \leq L$, and (P2') $e \notin G_{i,j}$. By Claim 36, there exists such a subgraph $G_{i,j}$. Since $e \notin G_{i,j}$ the subgraph $G_{i,j}$ contains no adversarial edges. Therefore, the computation of Step (2.1) is performed successfully, resulting in an L-neighborhood-cover $\mathcal{S}_{i,j}$. Since $\text{dist}_{G_{i,j} \setminus \{e\}}(u, v) \leq L$, there exists a cluster $S_{i,j,k} \in \mathcal{S}_{i,j}$ such that $u, v \in S_{i,j,q}$. Hence, the edge $e = (u, v)$ is a short bridgeless edge. Therefore, according to Step (2.2), a cycle C_e with identifier $ID(C_e)$ of length $\widetilde{O}(L)$ such that $(u, v) \in C_e$ is added to the cycle collection. That is, every node $w \in C_e$ adds the tuple $(ID(C_e), C_e)$ to its output set $\mathcal{C}(w)$.

We next show that during the j'th iteration of Step (2), the edge (u, v) is the only short bridgeless edge. It will then imply that during iteration j, a single cycle is added to the cycle collection, and therefore the iteration is not canceled due to large congestion by any of the nodes in V. For every reliable edge $(w_1, w_2) \in G_i$, if there exists a tree $T_{i,j,k}$ containing both w_1 and w_2, by the definition of neighborhood covers it holds that $\mathrm{dist}_{G_{i,j}\setminus\{(w_1,w_2)\}}(w_1,w_2)O(L\cdot\log^c n) \leq L'$. Since $e \notin G_{i,j}$, it implies that $\mathrm{dist}_{G_i\setminus\{e,(w_1,w_2)\}}(w_1,w_2) \leq L'$. Hence, by Claim 38 and Corollary 32, the edge (w_1,w_2) is covered in Step (1) of the i'th phase and therefore both w_1 and w_2 consider the edge as i-handled. Thus, (w_1,w_2) is not a short bridgeless edge. It follows that no node cancels the j'th iteration in Step (2.2). Additionally, as $C_e \subseteq G_i$, it holds that $C_e \cap (E' \cup F^*) = \{e\}$, and therefore the tuple (e,E') is covered. □

Finally, for the sake of the implementation of the general compilers in the next section, we note that all the cycles and the paths in the (f,F^*)-FT cycle cover* are simple.

Observation 41. *All cycles and paths in an (f,F^*)-FT cycle cover* are simple.*

Proof of Observation 41. In every phase i, by the properties of Alg. $\mathsf{ComputeOneFTCycCov}(G_i,L')$, all cycles and paths obtained in Step (1) are simple. Additional, in Step (2), for every short bridgeless edge e, for which the algorithm adds a cycle $C_e = \pi(u,v)\circ e$ to the cycle collection, every node $w \in C_e$ adds a single tuple $(ID(C_e),C_e)$ to its final output set $\mathcal{C}(w)$. Hence the degree of every node in the cycle (or truncated cycle) obtained in Step (2) is at most two and therefore the cycle (or path) is simple. □

Round complexity. The algorithm proceeds in ℓ phases. Consider phase i. Applying Alg. $\mathsf{DetComputeOneFTCycCov}(G_i,L')$ in Step (1) takes $\widehat{O}(L^4)$ rounds. Applying the broadcast alg. of Theorem 20 used during the verification step (Step (2.6)) takes $O(fD\log n)^{O(f)}$ rounds. In Step (2), the algorithm iterates over $\ell_i = \widetilde{O}(L^2)$ subgraphs $\mathcal{G}_i$. In each iteration, applying the L-neighborhood cover algorithm of Theorem 12 takes $\widetilde{O}(L)$ rounds. The total round complexity is bounded by $\widehat{O}(\ell\cdot(fD\log n)^{O(f)})$. Note that one can also reduce the edge congestion by using the randomized algorithm for $(1,e')$-FT cycle cover of Theorem 6.

3.2 General Compilers Given (f,F^*)-FT cycle cover*

We next show that our (f,F^*)-FT cycle cover* yields a general compiler that translates any r-round distributed algorithm $\mathcal{A}$ into an equivalent algorithm $\mathcal{A}'$, that works in the presence of f adversarial edges with round complexity $r' = O(r\cdot c\cdot d^3)$. Throughout, we use the following definition for a minimum s-v cut defined over a collection of s-v *paths*.

Definition 42 (Minimum (Edge) Cut of a Path Collection)**.** *Given a collection of s-v paths $\mathcal{P}$, the minimum s-v cut of $\mathcal{P}$, denoted as* $\mathrm{MinCut}(s,v,\mathcal{P})$*, is the minimal number of edges appearing on all the paths in $\mathcal{P}$. I.e.,* $\mathrm{MinCut}(s,v,\mathcal{P}) = x$ *implies that there exists a collection of x edges E' such that for every path $P \in \mathcal{P}$, it holds that $E' \cap P \neq \emptyset$.*

General compiler given an (f,F^*)-FT cycle cover* (Proof of Theorem 10). The compiler works in a round-by-round fashion, where every round of $\mathcal{A}$ is implemented within a phase of $O(c\cdot d^3)$ rounds.

Compilation of round i. Let $\mathcal{C}$ be the given (f,F^*)-FT cycle cover*. Fix a round i in $\mathcal{A}$, and an edge $e=(u,v)\in E$, and let $M_{u\to v}$ be the message sent from u to v in round i. Let $\mathcal{P}_{u,v} = \{C\setminus\{e\} \mid C \in \mathcal{C}, e \in C\}$ be the collection of u-v paths obtained from the cycles covering e in $\mathcal{C}$. In the i'th phase of Alg. $\mathcal{A}'$, the node u sends v the message $M_{u\to v}$ through the direct edge e, as well as through *all* the u-v paths in $\mathcal{P}_{u,v}$. The message $M_{u\to v}$ is augmented with the cycles ID and the path description, where every node in the path augments the message with its own ID [15]. On every path (i.e., for every cycle ID), each node sends at most one such message.

At the end of phase i, every node v computes the message $\widetilde{M}_{u,v}$ as its estimate for the message $M_{u\to v}$ for every $u \in N(v)$. Let M' be the message v received over the directed edge (u,v). For a message M received by v, let $\mathcal{P}_M$ be the path collection on which v received this message during the i'th phase, such that $(u,v) \notin P$ for every path $P \in \mathcal{P}_M$. If $\mathrm{MinCut}(u,v,\mathcal{P}_{M'}) \geq f$, then v sets $\widetilde{M}_{u,v} = M'$. Otherwise, v sets $\widetilde{M}_{u,v} = M$, for some message M satisfying $\mathrm{MinCut}(u,v,\mathcal{P}_M) \geq f$, breaking ties arbitrarily.

Correctness. Fix a phase i and an edge $e = (u,v) \in E$. We show that at the end of the i'th phase, it holds that $\widetilde{M}_{u,v} = M_{u\to v}$. We first show that $\mathrm{MinCut}(u,v,\mathcal{P}_{M_{u\to v}}) \geq f$.

Claim 43. *At the end of the i'th phase it holds that* $\mathrm{MinCut}(u,v,\mathcal{P}_{M_{u\to v}}) \geq f$.

Proof. By the definition of the (f,F^*)-FT cycle cover* $\mathcal{C}$, for every subset $E' \subseteq E$ of size $|E'| \leq f-1$, there exists a cycle $C \in \mathcal{C}$ such that $C \cap (E' \cup F^* \cup \{e\}) = \{e\}$. It then follows that for the set $\mathcal{P} = \{P \in \mathcal{P}_{u,v} \mid P \cap F^* = \emptyset\}$, it holds that $\mathrm{MinCut}(u,v,\mathcal{P}) \geq f$. Since $P \cap F^* = \emptyset$ for every path $P \in \mathcal{P}$, for the correct message $M_{u\to v}$ it holds that $\mathcal{P} \subseteq \mathcal{P}_{M_{u\to v}}$ and therefore $\mathrm{MinCut}(u,v,\mathcal{P}_{M_{u\to v}}) \geq f$. □

If the edge (u,v) is reliable, then $M' = M_{u\to v}$, and due to Claim 43 it also holds that $\mathrm{MinCut}(u,v,\mathcal{P}_{M_{u\to v}}) \geq f$. Therefore v sets $\widetilde{M}_{u,v} = M' = M_{u\to v}$. Next, assume $(u,v) \in F^*$ is adversarial. We claim that for any message $M \neq M_{u\to v}$, it holds that $\mathrm{MinCut}(u,v,\mathcal{P}_M) \leq f-1$. For a path $P \in \mathcal{P}_M$, since $M \neq M_{u\to v}$, the message M which propagated over the path P, was initiated by some adversarial edge in F^*. We claim that the path description of P contains an edge $e' \in F^*$, as for the last adversarial edge in P, the node v will receive its ID as part of the path description. Additionally, since the edge (u,v) does not appear in any path in $\mathcal{P}_M$, it follows that $P \cap (F^* \setminus \{e\}) \neq \emptyset$ for every path $P \in \mathcal{P}_M$. Hence, $\mathrm{MinCut}(u,v,\mathcal{P}_M) \leq |F^* \setminus \{e\}| = f-1$. We can then conclude that $M_{u\to v}$ is the only message for which $\mathrm{MinCut}(u,v,P_M) \geq f$ and therefore $\widetilde{M}_{u,v} = M_{u\to v}$ as desired.

Round complexity. All cycles (and paths) in the (f,F^*)-FT cycle cover* have length at most d, and edge congestion at most c. Since each edge belongs to c cycles of length d, the number of messages sent over an edge in a given phase is bounded by $c \cdot d$. Since each message sent over a path P is augmented with the path description of P, the messages have size $O(d \log n)$. Hence, in every phase, each edge needs to send $c \cdot d$ messages, of size $O(d \log n)$ along paths of size d. Therefore each phase requires $O(c \cdot d^3)$ rounds. As a result, an r-round algorithm $\mathcal{A}$ is simulated in $\mathcal{A}'$ by phase of $O(r \cdot c \cdot d^3)$ rounds.

3.3 Lower Bounds (Proof of Theorem 4)

In this section, we present a lower bound for the quality of the FT-cycle covers with parameters c,d, where the quality is measured by the summation $c+d$ (congestion + dilation). At the heart of our lower bound argument lies a graph-theoretical characterization of the tradeoff between

[15] We note that in the compilers against a single edge, it was not needed to send the path information.

congestion and dilation in a collection of s-v paths with a sufficiently large flow. Given an (unweighted) graph $G = (V, E)$, and a pair of nodes s, v, a *route set* is a collection of s-v paths in G of sufficiently large flow. Our goal is then to characterize the tension between the length of these paths and their overlap subject to some threshold on the flow of these paths.

Congestion vs. dilation tradeoff in route-sets. Given a graph $G = (V, E)$, a pair of nodes s, v and a route-set $\mathcal{P}$ of s-v paths, define the *dilation* of $\mathcal{P}$ by $\text{dilation}(\mathcal{P}) = \max_{P \in \mathcal{P}} |P|$, and the congestion of $\mathcal{P}$ by $\text{congestion}(\mathcal{P}) = \max_{e \in E} |\{P \in \mathcal{P} \mid e \in P\}|$. The next lemma shows that if the flow of the route-set is required to be sufficiently large, i.e., at least t, then there are graphs in which the congestion + dilation of these paths *must* be exponentially large in t.

Lemma 44. *For any parameters $t \geq 1$, $\rho \geq 1$ and $D = \Omega(t)$, there exists a $(t+1)$ edge-connected graph $G = (V, E)$ with diameter D, and two nodes $u, v \in V$ with the following property. For* any *collection of u-v paths $\mathcal{P}$ such that* $\text{MinCut}(u, v, \mathcal{P}) \geq t + 1$ *it holds that if* $\text{dilation}(\mathcal{P}) = \rho$ *then* $\text{congestion}(\mathcal{P}) = \Omega((\frac{D-1}{t+1})^{t+1}/(t \cdot \rho))$.

The Graph construction. We start with describing the lower bound graph denoted as G^*. The graph G^* consists of $t+1$ layers of edges $E_0, E_1, \dots, E_t$ described described in an inductive manner. Let $\widehat{D} = \frac{D-1}{t+1}$.

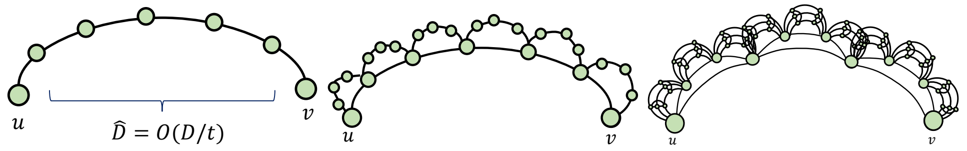

Figure 1: An illustration of the graph construction for $t = 2$. Left: The first layer consists of a simple u-v path of length $\widehat{D}$. Middle: Illustration of the first two layers E_0 and E_1. Every layer consists of $\widehat{D}$ lengths paths covering the edges of the previous layer. Right: The constructed graph. The last layer E_t covers each of the edges in E_{t-1} with $t+1$ edge disjoint paths that are connected via cliques.

Layer 0. We introduce two designated nodes $u, v \in V$. The nodes u and v are connected via a path of length $\widehat{D} + 1$, denoted as $P_0 = (u, u_1, \dots u_{\widehat{D}}, v)$. See Figure 1 (Left) for an illustration of the first layer E_0.

Layer i **for** $1 \leq i \leq t - 1$**.** The i'th layer E_i consists of $\widehat{D}$-length paths covering the edges in E_{i-1}. For every edge $e = (w_1, w_2) \in E_{i-1}$, we connect w_1 and w_2 by a path $P_e = (w_1, u_{e,1} \dots u_{e,\widehat{D}-1}, w_2)$ of length $\widehat{D}$. We then define $E_i = \bigcup_{e \in E_{i-1}} E(P_e)$. See Figure 1 (Middle) for an illustration of the construction after adding the first layer of edges E_1 to the basic construction.

Layer t**.** The last layer E_t covers each of the edges in E_{t-1} with $t + 1$ edge disjoint paths that are connected connected via cliques. For every edge $e = (w_1, w_2) \in E_{t-1}$, we introduce $t + 1$ edge-disjoint w_1-w_2 paths of length $\widehat{D}$ denoted as $\mathcal{P}_e = \{(w_1, u^i_{e,1}, \dots, u^i_{e,\widehat{D}}, w_2)\}^t_{i=1}$. Additionally, each set of nodes $u^1_{e,j}, \dots u^{k-1}_{e,j}$ for $j \in [1, \widehat{D} - 1]$ form a clique. This completes the description of the lower bound graph, as illustrated in Figure 1 (Right).

Correctness. We start with showing the constructed graph G^* is indeed $(t+1)$ edge-connected, with diameter at most D.

Observation 45. *The graph G^* is $(t+1)$ edge-connected and with diameter at most D.*

Proof. For every two nodes $w_1, w_2 \in V$ by te definition of the last layer E_t, there exists $t+1$-edge disjoint w_1-w_2 paths in E_t. Next, we bound the diameter of G^*. Consider two nodes $w_1, w_2 \in V$. By the definition of the graph G^*, the distance between w_1 and w_2 from the path P_0 is at most $t\widehat{D}/2$. Since the length of P_0 is $\widehat{D}+1$, by the triangle inequality it holds that:

$$\text{dist}_{G^*}(w_1, w_2) \leq t\widehat{D}/2 + t\widehat{D}/2 + \widehat{D} + 1 = (t+1)\widehat{D} + 1 = D$$

□

The next claim completes the proof of Lemma 44.

Claim 46. *For any u-v route-set $\mathcal{P}$ with* $\text{MinCut}(u, v, \mathcal{P}) \geq t+1$ *and* $\text{dilation}(\mathcal{P}) = \rho$, *it holds that* $\text{congestion}(\mathcal{P}) = \Omega((\frac{D-1}{t+1})^{t+1}/(\rho \cdot t))$.

Proof. Let $\mathcal{P}$ be a collection of u-v paths such that $\text{MinCut}(u, v, \mathcal{P}) \geq t+1$ with $\text{dilation}(\mathcal{P}) = \rho$. Consider a collection of t–tuples $\mathcal{F} \subseteq E_0 \times E_1 \cdots \times E_t$, defined as follows. A tuple $(e_0, \ldots e_{t-1})$ is in $\mathcal{F}$ if for every $i \in [0, t-1]$ it holds that (i) $e_i \in E_i$, and (ii) e_i is on the unique path covering e_{i-1}, added in the i^{th} layer. Note that by the definition of the graph G^*, the size of $\mathcal{F}$ is $\widehat{D}^t$. In order to bound the congestion of $\mathcal{P}$ we begin with bounding the size of the route set $\mathcal{P}$ using the cardinality of $\mathcal{F}$.

For a u-v path $P' \in \mathcal{P}$, let $S(P') = \{F \in \mathcal{F} \mid P' \cap F = \emptyset\}$ be the tuples in $\mathcal{F}$ which do not intersect with P'. We next show that $|P'| = \Omega(\widehat{D} \cdot |S(P')|)$. For every $F = (e_0, \ldots, e_{t-1}) \in S(P')$, because P' is a u-v path and $P' \cap F = \emptyset$, P' contains a sub-path P_F between the two endpoints of the edge e_{t-1}, such that $P_F \subseteq E_t$. By the definition of E_t, it holds that $|P_F| \geq \widehat{D}$. Additionally, by the definition of $\mathcal{F}$, for every two *different* tuples $F_1 = (e_0, \ldots, e_{t-1})$ and $F_2 = (e'_0, \ldots, e'_{t-1})$ in $S(P')$, it holds that $e_{t-1} \neq e'_{t-1}$, and therefore $P_{F_1} \cap P_{F_2} = \emptyset$. Hence, for every tuple $F \in S(P')$ we can account P' with $\widehat{D}$ unique edges. As a result, $|P'| = \Omega(\widehat{D} \cdot |S(P')|)$.

Next, as $\text{MinCut}(u, v, \mathcal{P}) \geq t+1$, for every tuple $F \in \mathcal{F}$ of t edges, there exists a path $P' \in \mathcal{P}$ such that $F \cap P' = \emptyset$ (i,e,. $F \in S(P')$). Since the length of each path $P' \in \mathcal{P}$ is at most ρ, it holds that $|S(P')| \leq \rho/\widehat{D}$.

As $|\mathcal{F}| = \widehat{D}^t$, we conclude that the collection $\mathcal{P}$ contains at least $\widehat{D}^t \cdot \widehat{D}/\rho$ different paths. Because the degree of the node v is $2t+1$, there exists an edge (w, v) that participates in at least $\widehat{D}^{t+1}/(\rho \cdot (2t+1))$ paths in $\mathcal{P}$. Hence,

$$\text{congestion}(\mathcal{P}) \geq \widehat{D}^{t+1}/(\rho \cdot (2t+1)) = \left(\frac{D-1}{t+1}\right)^{t+1} / (\rho \cdot (2t+1)) \,.$$

□

From Lemma 44 it follows that there exists a graph G^* with diameter D such that any collection of $t+1$ disjoint paths (i.e., with $\text{congestion}(\mathcal{P}) = 1$), must contain a long path.

Corollary 47. *For any parameters $t \geq 1$, and $D = \Omega(t)$, there exists a $(t+1)$ edge-connected graph $G = (V, E)$ with diameter D, and two nodes $u, v \in V$ such that* any *collection of u-v* disjoint *paths $\mathcal{P}$ of size $|\mathcal{P}| = t+1$, contains a path $P \in \mathcal{P}$ of length $|P| = \Omega((\frac{D-1}{t+1})^{t+1}/t)$.*

Finally, we prove Theorem 4 and show a lower bound on the quality of f-FT cycle covers using Lemma 44.

Proof of Theorem 4

Proof. Let $G^* = (V, E)$ be the n-node $(f+1)$ edge-connected graph with diameter D, with two designated nodes $u, v \in V$ of Lemma 44. Consider the graph $G' = G^* \cup \{(u,v)\}$, obtained by adding the edge (u,v) to the graph G^*. Let $\mathcal{C}$ be an f-FT cycle cover for G' with parameters c, d, and let $\mathcal{P} = \{C \setminus \{(u,v)\} \mid C \in \mathcal{C}, \text{ and } (u,v) \in C\}$ be the collection of u-v paths induced by the cycles in $\mathcal{C}$. By the definition of the f-FT cycle cover $\mathcal{C}$, for every subset $E' \subseteq E$ of size $|E'| \leq f-1$, there exists a cycle $C \in \mathcal{C}$ such that $C \cap (E' \cup \{(u,v)\}) = \{(u,v)\}$. It then follows that $\text{MinCut}(u, v, \mathcal{P}) \geq f$. As $(u,v) \notin P$ for every $P \in \mathcal{P}$, the set $\mathcal{P}$ is also a u-v rout-set in G^*. Hence, by Lemma 44 it holds that $\text{dilation}(\mathcal{P}) + \text{congestion}(\mathcal{P}) = (\frac{D}{f})^{\Omega(f)}$, and therefore $c + d = (\frac{D}{f})^{\Omega(f)}$. □

Acknowledgments 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 22.

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 $n > 3$t 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.

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

[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.

A Missing Proofs of Section 1

Proof of Lemma 3. Let $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ be a collection of $\ell = O(f(5fD)^f \log n)$ subgraphs obtained by sampling each edge into G_i independently with probability of $p = 1 - 1/(5fD)$. For each graph G_i, let $\mathcal{C}_i$ be the cycles computed in G_i by applying the algorithm of Fact 13 with $G = G_i, D' = 5fD$. Let $\mathcal{C} = \bigcup_i \mathcal{C}_i$. The length of all cycles is bounded by $\widehat{O}(fD)$ and the total congestion is bounded by $\widehat{O}(\ell)$.

It remains to show that $\mathcal{C}$ is an f-FT cycle cover $\mathcal{C}$. Fix an edge e and $E' \subseteq E$ of size at most $(f-1)$. The probability that a given G_i contains a short cycle C_e of length at most $5fD$ and that $E' \cap G_i = \{e\}$ is given by $\Theta(1/(fD))$. Therefore, with probability of $1 - 1/n^{3f}$, (e, E') have a good subgraph G_i. The claim follows by applying the union bound over all n^{2f} pairs (e, E'). □

Proof of Claim 21. Let $N' = \lceil \log N \rceil$, where $N = \text{poly}\, n$ is the polynomial estimate on n known to all nodes in G. The nodes apply $\ell = O(\log n)$ iterations of Alg. AdvBroadcast until they receive the identifier of at least one node. Each iteration $i \in \{0, \ldots, \ell\}$ consists of $\widetilde{O}(D'^2)$ rounds. Specifically, iteration i starts by letting each node u being sampled independently with probability $1/N_i$ where $N_i = 2^{N'-i}$. A sampled node u initiates Alg. AdvBroadcast(m_u) with a message $m_u = ID(u)$. For every node v, let S_v be the set of all messages that v *accepted* during this iteration. If $|S_v| \neq \emptyset$, then v outputs $LE(v) = \{\max(ID(u)) \mid m_u \in S_v\}$ and stops. I.e., it will not sample itself in the future experiments $j \geq i + 1$ and would not send messages. This concludes the description of the i'th iteration. Each iteration lasts for a fixed number of $\widetilde{O}(D'^2)$ rounds, even if no node gets sampled.

We now analyze the algorithm. We first claim that $LE(v) = LE(u)$ for all nodes u, v, and that $LE(v)$ corresponds to an identifier of a real node in G. Consider the first iteration i in which at least one node got successfully sampled, and let V_i be the set of all nodes that got sampled in this iteration. By a simple application of Chernoff, it is easy to see that w.h.p. there exists such an iteration i, and $|V_i| = O(\log n) = O(\log N)$. Each node $u \in V_i$ initiates Alg. AdvBroadcast(m_u). Since $|V_i| = O(\log n) = O(\log N)$, all these algorithms can be run in parallel by paying a logarithmic factor in the number of rounds[16]. Hence, by Theorem 19 at the end of the i^{th} iteration all nodes obtain the IDs of all nodes in V_i. Thus, the output $LE(u)$ of every node u corresponds to the node with the largest ID in V_i. The claim follows. □

B Missing Details for Cycle Cover Constructions

Claim 48. *The low-congestion cycle cover algorithm* ComputeCycCov *can be modified such that each cycle C has a unique identifier $ID(C)$ of $\widehat{O}(1)$ bits. In the distributed output of the algorithm, each node u knows $ID(C)$ for every cycle C that passes through u.*

Proof. The reader is referred to the centralized constructions of the cycle covers in [33] and their distributed implementation in [35] for complete details. We will next explain the minor modifications required to assign cycles unique IDs. Let T be a BFS tree rooted at an arbitrary node. The algorithm has two main procedures, for covering the non-tree edges in $E \setminus T$ and for covering T. We start with the simpler case of assigning unique IDs for the cycles covering the non-tree edges. By Claim 5 (2) in [35], each non-tree edge appears on a unique cycle. In addition, by construction

[16] That is, each round of the broadcast algorithm AdvBroadcast will be implemented now over a $O(\log n)$ rounds.

each cycle has $\widehat{O}(1)$ non-tree edges[17]. Thus the identifier of each such cycle C is made by the list of $\widehat{O}(1)$ non-tree edges ordered in increasing IDs. (The ID of an edge $e = (u, v)$ is the $O(\log n)$ string $ID(u), ID(v)$ where w.l.o.g. $ID(u) < ID(v)$).

We next describe how to define unique IDs for the cycles computed in the tree covering procedure. The algorithm of [33] starts by defining for every edge $e = (u, p(u))$ a swap edge $e' = (x, y)$ where $p(u)$ is the parent of u in the tree. The algorithm then defines a virtual edge $(u, s(u))$ where $s(u) \in \{x, y\}$ is carefully chosen, and each node u knows its $s(u)$ mate. See Fig. 7 in [33]. The edges $(u, s(u))$ are covered by applying a recursive balanced partitioning of T into two trees T_1, T_2. The edge $(u, s(u))$ is handled at the point where u and $s(u)$ belong to different trees, namely, T'_1 and T'_2. At this point, the algorithm considers all nodes u in T'_1 that have their second endpoint $s(u')$ in T'_2 and vice-versa. All these nodes in T'_1 get matched by applying the TreeEdgeDisjointPath Alg. This process defines virtual edges internal to T'_2, where each virtual edge corresponds to exactly two $(u, s(u))$ edges. The ID of each such virtual edge will be the concatenation of these two edge IDs $(u, s(u))$ and $(u', s(u'))$. The virtual edges are then covered by applying the non-tree covering algorithm in the tree T'_2. By the description above each cycle contains $\widehat{O}(1)$ non-tree edges. In our case, the non-tree edge is a virtual edge whose ID correspond to two edge IDs. Thus, the ID of the cycle can again be represented using $\widehat{O}(1)$ many bits. □

Claim 49. *The low-congestion cycle cover algorithm* ComputeCycCov *can be modified such that for every node v and a cycle $(ID(C), C)$ in its output collection, v obtain a full description of C, received from both directions over C.*

Proof. At the end of the cycle cover construction, the nodes obtain full cycle descriptions in a phase by phase manner, where each phase consists of $\widehat{O}(1)$ many rounds (i.e., the congestion bound of Observation 14). Every round corresponds to a cycle $ID(C)$ that an edge participates in. For every node u on a cycle C obtained by ComputeCycCov, first u sends the message $(ID(C), u)$ to both neighbors on C. Then for $\widehat{O}(D)$ many phases, every message $(ID(C), v)$ received by u from one neighbor on C is sent to the other neighbor. For every edge $(u, w) \in G$, the messages are sent from u to w as follows. For each cycle $ID(C)$ that an edge (u, w) participate in, u sends w at most one message in every phase.

After $\widehat{O}(D)$ many phases, u output the cycle description of C, where the i'th node in C is set to be the node received by u at the i'th phase.

□

In addition, since the only randomized part in Alg. ComputeCycCov is in the computation of the neighborhood covers, using the deterministic algorithm of Theorem 12 provides Observation 14.

C Missing Proofs for Section 2

Observation 50. *Let $Q_1 = [a_1, a_2, \ldots, a_\ell = w]$ and $Q_2 = [b_1, b_2, \ldots, b_k = w]$ be two walks along which a message $ver(C') = (ID(C'), ID(v_{C'}), ver)$ is propagated. Then if $a_1 = b_1$ and $a_2 = b_2$, it holds that $Q_1 = Q_2$.*

[17] It is $2^{1/\epsilon}$ in the description in [33], but ϵ is chosen to be $1/\sqrt{\log n}$

Proof. We show that for every $i \geq 1$, $a_i = b_i$ by induction on i. For $i = 1, 2$, the claim follows from the assumption that $a_1 = b_1$ and $a_2 = b_2$. Assume that the claim holds for $i \geq 2$ and consider the nodes a_{i+1}, b_{i+1}. By the induction assumption, $a_i = b_i$. By the description of the walks Q_1 and Q_2, the node $w = a_i = b_i$ sent the message $ver(C)$ received from a_{i-1} to both a_{i+1} and b_{i+1}. By Step (2.2), w sends the message $ver(C)$ received from a_{i-1} to exactly one neighbor i.e., its second neighbor on the cycle C_w, where C_w is the unique cycle satisfying that $(ID(C), C_w) \in \mathcal{C}'_i(w)$. This implies that $a_{i+1} = b_{i+1}$. □

Proof of Claim 24. By the definition of P_1 and P_2, every node $w \in \widehat{C}$ sent the message $ver(C) = (ID(C), v_C, ver)$ towards u^*. Therefore, all nodes in P_1 and P_2 are active at the beginning of the verification step, and according to Step (2.2) for every $w \in \widehat{C}$, there exists a unique cycle, denoted as C_w, such that $(ID(C), C_w) \in \mathcal{C}'_i(w)$. We start with showing that $\widehat{C}$ is a cycle. By Observation 23, we consider the following cases.

Case P_1 and P_2 start with v_C. In this case $\widehat{C}$ forms a cycle.

Case P_1 and P_2 start with e'. Without loss of generality, assume that P_1 starts with the (directed) edge (v_1, v_2). Assume toward contradiction that P_2 starts with (v_1, v_2) as well. By Observation 50, it then holds that $P_1 = P_2$ and $w_1 = w_2$, leading to a contradiction. Therefore P_1 and P_2 use e' in opposite directions, concluding that $\widehat{C}$ is a cycle.

Case P_1 starts with v_C and P_2 starts with the adversarial edge e'. We show that this case cannot occur. Assume without loss of generality that the path P_2 starts with the directed edge (v_1, v_2). We divide the analysis to two sub-cases.

Subcase $e' \notin P_1$. As P_1 is a reliable path and every node $w \in P_1$ sent the message $ver(C)$ towards u^*, it follows that the cycle description u^* received from its neighbor on P_1 (i.e, from $u'_{k'-1}$) during Alg. ComputeCycCov, is of the form $Q_1 \circ P_1$. Similarly, as all nodes on $P_2[v_2, u^*]$ are reliable, the cycle description u^* received from its neighbor on P_2 (i.e, from u_{k-1}), is of the form $Q_2 \circ P_2$. Since u^* is active at the beginning of the verification step, it follow that $C = Q_1 \circ P_1 = Q_2 \circ P_2$, and therefore $C = \overline{P_1} \circ P_3 \circ P_2$ for some simple v_C-v_1 path P_3. Moreover, since the path $\overline{P_1} \circ P_3$ is reliable, we can conclude that every $w \in \overline{P_1} \circ P_3$ received the path $\overline{P_1} \circ P_3$ as part of the cycle with identifier $ID(C)$, obtained by Alg. ComputeCycCov.

Next, since $(ID(C), C) \in \mathcal{C}_i(u)$, the node u^* did not receive a cancellation message $cancel(C)$ from v_C over the reliable path P_1. As P_1 is a reliable path, all nodes on P_1 holds the correct distance from v_C, and therefore we can conclude that v_C did not initiate a cancellation message. Hence, according to Step (2.3), v_C received the message $ver(C)$ from both its neighbors on the cycle - its neighbor in P_1 and its neighbor in $\overline{P_3}$.

Hence, according to Steps (2.1) and (2.2), the message $ver(C)$ propagated over the path P_3 towards v_1. As a result, during Step (2) the message $ver(C)$ is received by u^* over the concatenated path $\overline{P_3} \circ P_2$, in contradiction to the maximality of P_2. See Figure 2(a) for an illustration.

Subcase $e' \in P_1$. First, assume that $(v_1, v_2) \in P_1$. Since P_2 starts with the same edge (v_1, v_2), by Observation 50 it holds that $P_1[v_1, u] = P_2$. This contradicts the assumption that u^* received $ver(C)$ from two *different* neighbors over P_1 and P_2.

Next, assume that P_1 contains the directed edge (v_2, v_1). Let $w \in N(v_2)$ be the neighbor of v_2 such that w and v_1 are the neighbors of v_2 obtained by Alg. ComputeCycCov. As v_2 communicates the message $ver(C)$ only with its two neighbors in the cycle (i.e., v_1 and w), P_2 starts with the sub-path $[v_1, v_2, w] \subseteq P_2$, and P_1 contains the reveres sub-path $[w, v_2, v_1] \subseteq P_1$. As a result, by Observation 50 it follows that $P_2 = \overline{P_1}[v_1, u^*]$. In particular, in holds that $u^* \in P_1[v_C, v_1]$, leading to a contradiction, as P_1 is a *simple* v_C-u^* path. See Figure 2(b) for an illustration.

Finally, we show that $\widehat{C}$ is also simple. Assume toward contradiction that $\widehat{C}$ is not simple. Let v be the closest node to u^* on $\widehat{C}$ such that $v \in P_1 \cap P_2 \setminus \{v_C, u^*\}$. Note that because P_1 and P_2 are simple paths, and $P_1 \neq P_2$, if $\widehat{C}$ is not simple there exists such a node v. Since u^* received the message $ver(C)$ from two *different* neighbors w_1, w_2, it follows that $P_1[v,u] \neq P_2[v,u]$. As we choose v to be the closest node to u^* such that $v \in P_1 \cap P_2 \setminus \{v_C, u^*\}$, we conclude that v has two different neighbors in P_1 and P_2 denoted as a_1, a_2. Additionally, v received the message $ver(C)$ from a different node a_3 over the path P_1. This contradicts Step (2.2), as the number of neighbors in $N(v)$ that send the message $ver(C)$ to v is at most two. $\square$

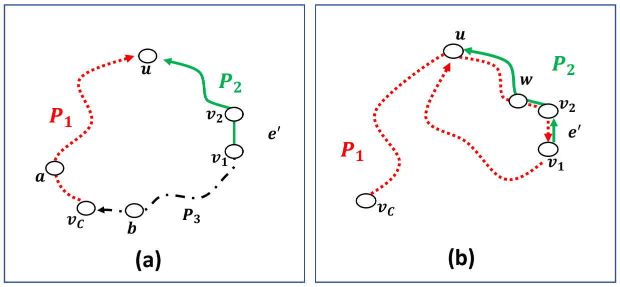

Figure 2: Proof of Claim 24: Illustrations of the case where P_1 (dotted red) starts with v_C, and P_2 (simple line green) starts with the adversarial edge (v_1, v_2). Figure (a) illustrates the case where $e' \notin P_1$. Since u added the cycle to $\mathcal{C}_i(u)$, v_C did not send a cancellation message $cancel(C)$, and v_C received the message $ver(C)$ from an additional path P_3 (shown in dashed). In this case, the message $ver(C)$ propagated over the path $\overline{P_3} \circ P_2$ towards u, in contradiction to the maximality of P_2. Figure (b) illustrates the case where $e' \in P_1$. As $P_1 \neq P_2$ the path P_1 contains the directed edge (v_2, v_1). Since v_2 send (or receive) the message $ver(C)$ only with its two neighbors v_1, w on the cycle C_{v_2}, it holds that $P_1[u, v_1] = \overline{P_2}$, in contradiction to the assumption that P_1 is a simple v_C-u path.

Proof of Lemma 34. We note that since G is 3 edge-connected, from Observation 16 it follows that for every pair of edges $e = (u,v), e'$ it holds that $\text{dist}_{G \setminus \{e,e'\}}(u,v) \leq 7D$. To apply the randomized construction from Fact 18(1), the nodes need to first compute a shared seed of $\widetilde{O}(1)$ random bits. To do that, the nodes pick a leader s using the leader election algorithm of Claim 21 within $\widetilde{O}(D^2)$ rounds. The leader s picks a random seed r of length $\widetilde{O}(1)$, and sends it to all the nodes by applying the deterministic broadcast protocol of Theorem 19(1). Given the random seed r, each node apply the randomized construction of Fact 18 to compute a $(7D, 1)$-covering family $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ for $\ell = O(D \log n)$. $\square$