Pulse Propagation, Graph Cover, and Packet Forwarding

Dissertation zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

vorgelegt von
Ben Wiederhake

Saarbrücken, 2022

Dean:	Univ.-Prof. Dr. Thomas Schuster
Chair:	Prof. Dr. Markus Blaeser
Reporters:	Dr. Christoph Lenzen
	Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn
Academic Assistant:	Dr. Rohaani Sharma
Colloquium:	February 14, 2022

Acknowledgments

Thank you, Christoph Lenzen, for supervising my years-long journey to my graduation, and providing so many opportunities to learn and explore. Especially when I took the scenic route down some rabbit hole[1]. And last but not least, thank you for organizing so many and great Board Games Nights!

I would also like to thank Antonios Antoniadis, for having an open ear, giving great feedback, and introducing me to the topic and fine points of Packet Scheduling.

Thank you so much Attila Kinali for introducing me to the world of Swiss chocolate – without you I would weigh significantly less. Also, I thoroughly enjoyed talking with you about all the (un)important things in life; including but not limited to technical details of IRC, transistors, Quartz oscillators, GPS, the Charly and Dorothy effect, and several ski huts.

I'm very grateful to Saeed Amiri. It was pure joy to develop and analyze our distance-r minimum dominating set algorithm and lower bound.

I'm also thankful to Matthias Függer, for all the productive discussions, his unique perspective on life, and excellent recommendations on Viennese cuisine.

I would like to thank Will Rosenbaum, for demonstrating that not all which is JavaScript is evil, and introducing me to the topic of Packet Forwarding. I am also thankful to him (and Christoph and Attila and Matthias) for fruitful discussions about the Metastability-Containing Frequency Adaption Module, which greatly expanded my (still limited) understanding of electrical engineering.

Also, thank you Corinna Coupette, Nick Fischer, Johannes Bund, Cosmina Croitoru, André Nusser, and Bhaskar Ray Chaudhury for all the wonderful discussions on Computer Science, Law, Jazz, Pizza, Jam sessions, hats, and cake.

Next, I would like to thank our Servicedesk, especially Maik Muschter and Andreas Alzano, for keeping all the infrastructure running, handling the countless tickets I opened, and keeping calm even after I found the umpteenth way to crash our computing cluster or bringing in a laptop battery that was about to explode.

Außerdem will ich mich besonders bei Tim, Doris und Wilm bedanken, für all die Unterstützung, Hilfestellungen, und auch die sprichwörtlichen Arschtritte. Ohne euch wäre ich nie so weit gekommen.

I would also like to thank Julia, Ferdinand, Max, Nora, and Pascal; it has been a Long Road with you. I am grateful to Christian, who helped us Find our Path.

Finally, a special thank you to Julius, Andreas, Vladislav, and the silk painting youth group, who each showed me wonderful aspects of life that I would never want to miss.

[1] Exercise for the reader: Let $X \sim \mathcal{N}(\mu, \sigma^2)$, show that $\lim_{\sigma\to\infty} \mathrm{Var}(\lceil X \rfloor) = \sigma^2 + 1/12$, where $\lceil \cdot \rfloor$ is the function that rounds to the nearest integer.

Abstract

We study distributed systems, with a particular focus on graph problems and fault tolerance.

Fault-tolerance in a microprocessor or even System-on-Chip can be improved by using a fault-tolerant pulse propagation design. The existing design TRIX achieves this goal by being a distributed system consisting of very simple nodes. We show that even in the typical mode of operation without faults, TRIX performs significantly better than a regular wire or clock tree: Statistical evaluation of our simulated experiments show that we achieve a skew with standard deviation of $O(\log \log H)$, where H is the height of the TRIX grid.

The distance-r generalization of classic graph problems can give us insights on how distance affects hardness of a problem. For the distance-r dominating set problem, we present both an algorithmic upper and unconditional lower bound for any graph class with certain high-girth and sparseness criteria. In particular, our algorithm achieves a $O(r \cdot f(r))$-approximation in time $O(r)$, where f is the expansion function, which correlates with density. For constant r, this implies a constant approximation factor, in constant time. We also show that no algorithm can achieve a $(2r + 1 - \delta)$-approximation for any $\delta > 0$ in time $O(r)$, not even on the class of cycles of girth at least $5r$. Furthermore, we extend the algorithm to related graph cover problems and even to a different execution model.

Furthermore, we investigate the problem of packet forwarding, which addresses the question of how and when best to forward packets in a distributed system. These packets are injected by an adversary. We build on the existing algorithm OED to handle more than a single destination. In particular, we show that buffers of size $O(\log n)$ are sufficient for this algorithm, in contrast to $O(n)$ for the naive approach.

Zusammenfassung

Wir untersuchen verteilte Systeme, mit besonderem Augenmerk auf Graphenprobleme und Fehlertoleranz.

Fehlertoleranz auf einem System-on-Chip (SoC) kann durch eine fehlertolerante Puls-Weiterleitung verbessert werden. Das bestehende Puls-Weiterleitungs-System TRIX toleriert Fehler indem es ein verteiltes System ist das nur aus sehr einfachen Knoten besteht. Wir zeigen dass selbst im typischen, fehlerfreien Fall TRIX sich weitaus besser verhält als man naiverweise erwarten würde: Statistische Analysen unserer simulierten Experimente zeigen, dass der Verzögerungs-Unterschied eine Standardabweichung von lediglich $O(\log\log H)$ erreicht, wobei H die Höhe des TRIX-Netzes ist.

Das Generalisieren einiger klassischer Graphen-Probleme auf Distanz r kann uns neue Erkenntnisse bescheren über den Zusammenhang zwischen Distanz und Komplexität eines Problems. Für das Problem der dominierenden Mengen auf Distanz r zeigen wir sowohl eine algorithmische obere Schranke als auch eine bedingungsfreie untere Schranke für jede Klasse von Graphen, die bestimmte Eigenschaften an Umfang und Dichte erfüllt. Konkret erreicht unser Algorithmus in Zeit $O(r)$ eine Annäherungsgüte von $O(r \cdot f(r))$. Für konstante r bedeutet das, dass der Algorithmus in konstanter Zeit eine Annäherung konstanter Güte erreicht. Weiterhin zeigen wir, dass kein Algorithmus in Zeit $O(r)$ eine Annäherungsgüte besser als $2r + 1$ erreichen kann, nicht einmal in der Klasse der Kreis-Graphen von Umfang mindestens $5r$.

Weiterhin haben wir das Paketweiterleitungs-Problem untersucht, welches sich mit der Frage beschäftigt, wann genau Pakete in einem verteilten System idealerweise weitergeleitet werden sollten. Die Pakete werden dabei von einem Gegenspieler eingefügt. Wir bauen auf dem existierenden Algorithmus OED auf, um mehr als ein Paket-Ziel beliefern zu können. Dadurch zeigen wir, dass Paket-Speicher der Größe $O(\log n)$ für dieses Problem ausreichen, im Gegensatz zu den Paket-Speichern der Größe $O(n)$ die für einen naiven Ansatz nötig wären.

Contents

Chapter 1

Introduction

This thesis covers distributed systems, with a particular focus on graph problems and fault tolerance. First, we will analyze a particular family of distributed systems (TRIX) that provides fault-tolerant pulse propagation, which is useful for real-world hardware. Second, we examine a generalization of a well-known graph cover problem to gain insights into how distance affects the difficulty of a graph cover problem. Third, we revisit an existing packet forwarding algorithm (OED) that can only deliver packets to a single destination, and use it as a building block to serve two destinations.

Many modern systems run on synchronous hardware, where a single clock signal is supplied to the entire circuit, and computation progresses in lock-step. One way to increase reliability is by introducing or improve fault-tolerance. In other words, we want to make sure that the circuit is still fully operational if any fault of a certain kind occurs. Fault-tolerant clock generation schemes already exist, i.e., designs that can create a clock signal even if any single component fails. However, even assuming that these generation schemes address all potential requirements (e.g., synchronization, type of behavior in the presence of faults, area and power consumption), a separate pulse propagation mechanism is still necessary, in order to get the clock signal efficiently to each circuit element. In particular, neighboring elements must receive a tightly synchronized clock signal. If the difference in timing, also called skew, is too large, then communication becomes too difficult or requires expensive and lossy re-synchronization steps, defeating the point of a common clock signal.

The first part of this thesis is about pulse propagation, beginning with an introduction to the state of the art, as well as a review of HEX [Dol+16], an existing pulse propagation scheme. We then introduce TRIX, which uses the same basic idea for fault-tolerance (using a median signal), but chooses a fundamentally different approach to its topology. We analyze TRIX' properties and improvements, and in particular provide strong experimental evidence that TRIX achieves remarkably low skew (mean 0, standard deviation roughly $O(\log \log H)$). This means that TRIX is significantly better than HEX at delivering a tightly synchronized clock signal.

This contribution has been presented in a brief announcement [LW20a] at the SSS 2020 conference. A full version [LW20b] is available.

The analysis of classic graph problems enables fascinating insights into how the structure of a problem or the graph class affects complexity. Recently, the study of the corresponding distance-r generalizations has received attention, as it seems to reveal how distance and sparsity influence complexity.

The second part of this thesis considers the distance-r generalization of well-known graph cover problems, most prominently the dominating set problem. We introduce an algorithm that computes an approximation, assuming a restricted graph class. In the analysis, we define a candidate set based on Voronoi cells; this approach may generalize to further graph classes. This implies the upper bound that a $O(r \cdot f(r))$-approximation can be achieved in time $O(r)$, where f is the expansion function, which correlates with density. We also prove the unconditional lower bound that no algorithm can achieve a $(2r + 1 - \delta)$-approximation for any $\delta > 0$ in time $O(r)$. Finally, we provide simple extensions of the distance-r dominating set algorithm to other distance-r graph cover problems, such as connected dominating set, vertex cover, and connected vertex cover, as well as the port numbering model.

We find that the presented upper and lower bound do not quite match, and leave a gap. We are forced to leave this as an open question whether the lower bound can be raised, or whether a different algorithm finds a good approximation more quickly.

These contributions have been published at CIAC 2021 [AW21] and TCS Special Issue on Algorithms and Complexity 2021 [AW22].

Many digital protocols are packet-based, and bandwidth-limitations prevent instant re-transmission. This necessitates packet buffers, in order to minimize packet loss. Infinitely large buffers would prevent this type of packet loss entirely, but are impractical. But how large do packet buffers need to be? Even assuming that the routing problem is already solved, i.e. each packet already knows which route it has to take, the timing and ordering of the packets has a large impact on the required packet buffer size. This is called the packet forwarding problem, and the existing OED algorithm (Odd-Even-Downward) guarantees that small packet buffers (of size $O(\log n)$, where n is the number of nodes) suffice, under a certain packet injection and forwarding model. However, OED assumes that all packets have all a single, shared destination, and depends crucially on it.

The third part of this thesis addresses the problem of packet forwarding. We introduce a new algorithm that uses OED as a building block, requiring a slight relaxation of the communication model, allowing to exchange packets instead of only forwarding them. OED breaks if even a second destination is introduced. We break this seeming barrier, as our algorithm can handle two separate destinations, which is a step towards a general packet forwarding strategy. The fundamental approach is to consider cumulative destination amounts, instead of individual destination amounts.

This contribution has not yet been published elsewhere.

Part I.

Pulse Propagation

Chapter 2

Introduction

This chapter introduces the concepts surrounding pulse propagation, and reviews how it is traditionally solved. We also explain the shortcomings that create a need for a distributed, fault-tolerant pulse propagation mechanism. Our proposed mechanism TRIX is a good candidate to fill this need, and we present an outline of how we demonstrate this in Chapter 3. Finally, we introduce all the notation and precise mathematical definitions needed to do so.

2.1. Motivation

The vast majority of computational hardware is clocked, meaning that it relies on a single clock pulse being distributed simultaneously down to all individual building blocks, like latches, flip-flops, buffers, etc.

In the pursuit of fault-tolerant systems, distributed and fault-tolerant pulse *generation* has been studied extensively [SSS07; ST87; WL88]. In the following, we will assume that this part of the system is already solved.

However, it would be wasteful to scale up the number of generated clocks (which run a full-blown clock generation algorithm) to the vast amount of building blocks. Traditional clock-trees cannot accomplish this task, as a single point of failure can permanently disturb the system.

In this work, we present a simple grid structure as a more reliable clock propagation method and study it by means of simulation experiments. Fault-tolerance is achieved by forwarding clock pulses on arrival of the second of three incoming signals from the previous layer.

A key question is how well neighboring grid nodes are synchronized, even without faults. Analyzing the clock skew under typical-case conditions is highly challenging. Because the forwarding mechanism involves taking the median, standard probabilistic tools fail, even when modeling link delays just by unbiased coin flips. This is precisely what we will study in this and the following chapter.

2.2. Related Work

When designing high reliability systems, any critical subsystem susceptible to failure must exhibit sufficient redundancy. Traditionally, clocking of synchronous systems is performed by clock trees or other structures that cannot sustain faulty components [Xan09]. This imposes limits on scalability on the size of clock domains; for instance, in multiprocessor systems typically no or only very loose synchronization is maintained between different processors [CSG98; PH90]. Arguably, this suggests that fault-tolerant clocking methods that are competitive – or even better – in terms of synchronization quality and other parameters (ease of layouting, amount of circuitry, energy consumption, etc.) would be instrumental in the design of larger synchronous systems.

To the best of our knowledge, at least until 20, or perhaps even 10 years ago, there is virtually no work on fault-tolerance of clocking schemes beyond production from the hardware community; due to the size and degree of miniaturization of systems at the time, clock trees and their derivatives were simply sufficiently reliable in practice. That this has changed is best illustrated by an upsurge of interest in single event upsets of the clocking subsystem in the last decade [Abo+15; Chi+11; Chi+12; CK14; Guj+15; Mal+16; Wan+16; Wis+09]. However, with ever larger systems and smaller components in place, achieving acceptable trade-offs between reliability, synchronization quality, and energy consumption requires to go beyond these techniques.

On the other hand, there is a significant body of work on fault-tolerant synchronization from the area of distributed systems. Classics are the Srikanth-Toueg [ST87] and Lynch-Welch [WL88] algorithms, which maintain synchronization even in face of a large minority (strictly less than one third) of *Byzantine* faulty nodes.[1] Going beyond this already very strong fault model, a line of works [DDP03; DH07; Dol+14; DW04; LR19] additionally consider *self-stabilization*, the ability of a system to recover from an unbounded number of transient faults. The goal is for the system to stabilize, i.e., recover nominal operation, after transient faults have ceased. Note that the combination makes for a very challenging setting and results in extremely robust systems: even if some nodes remain faulty, the system will recover from transient faults, which is equivalent to recover synchronization when starting from an arbitrary state despite interference from Byzantine faulty nodes.

While these fault-tolerance properties are highly desirable, unsurprisingly they also come at a high price. All of the aforementioned works assume a fully connected system, i.e., direct connections between each pair of nodes. Due to the strong requirements, it is not hard to see that this is essentially necessary [DHS86]: in order to ensure that each non-faulty node can synchronize to the majority of correct nodes, its degree must exceed the number of faults, or it might become effectively disconnected. In fact, it actually must have more correct neighbors than faulty ones, or a faulty majority of neighbors might falsely appear to provide the correct time. Note that emulating full connectivity using

[1] Distributed systems are typically modeled by network graphs, where the nodes are the computational devices and edges represent communication links. A Byzantine faulty node may deviate from the protocol in an arbitrary fashion, i.e., it models worst-case faults and/or malicious attacks on the system.

a crossbar or some other sparser network topology defeats the purpose, as the system then will be brought down by a much smaller number of faults in the communication infrastructure connecting the nodes. Accordingly, asking for such extreme robustness must result in solutions that do not scale well.

A suitable relaxation of requirements is proposed in [Dol+16]. Instead of assuming that Byzantine faults are also *distributed* across the system in a worst-case fashion, the authors of this work require that faults are "spread out." More specifically, they propose a grid-like network they call HEX, through which a clock signal can be reliably distributed, so long as for each node at most one of its four neighbors from which a signal is received is faulty. Note that for the purposes of this paper, we assume that the problem of fault-tolerant clock signal *generation* has already been sufficiently addressed (e.g. using [Dol+14]), but the signal still needs to be *distributed*. Provided that nodes fail independently, this means that the probability of failure of individual nodes that can likely be sustained becomes roughly $1/\sqrt{n}$, where n is the total number of nodes; this is to be contrasted with a system without fault-tolerance, in which components must fail with probability at most roughly $1/n$. The authors also show how to make HEX self-stabilizing. Unfortunately, however, the approach has poor synchronization performance even in face of faults obeying the constraint of at most one fault in each in-neighborhood. While it is guaranteed that the clock signal propagates through the grid, nodes that fail to propagate the clock signal cause a "detour" resulting in a clock skew between neighbors of at least one maximum end-to-end communication delay d.[2] This is much larger than the *uncertainty* u in the end-to-end delay: As clocking systems can be (and are) engineered for this purpose, the end-to-end delay will vary between $d - u$ and d for some $u \ll d$. To put this into perspective, in a typical system u will be a fraction of a clock cycle, while d may easily be half of a clock cycle or more.

This is inherent to the structure of the HEX grid, see Fig. 2.1. It seeks to propagate the clock signal from layer to layer, where each node has two in-neighbors on the preceding and two in-neighbors on their own layer. Because the possibility of a fault requires nodes to wait for at least two neighbors indicating a clock pulse before doing so themselves, a faulty node refusing to send any signal implies that its two out-neighbors on the next layer need to wait for at least one signal from their own layer. This adds at least one hop to the path along which the signal is propagated, causing an additional delay of at least roughly d.

[2] d includes the wire delay as well as the time required for local computations. As the grid is highly uniform and links connect close-by nodes, the reader should expect this value to be roughly the same for all links.

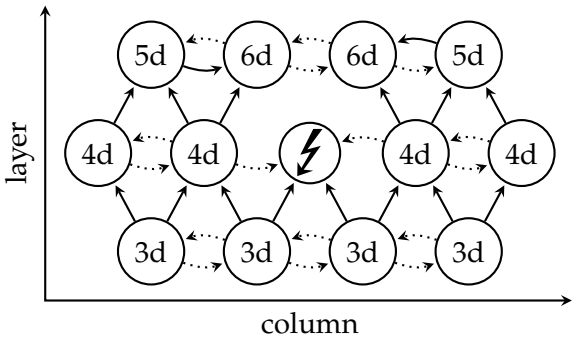

Figure 2.1.: A crashing node in a HEX grid causes a large skew between neighbors in the same layer, even with all links having exactly the same delay. Thick links cause nodes to pulse, dotted links mean that the transmitted pulse was too late to be considered. Observe that the faulty node fails to transmit any pulse at all.

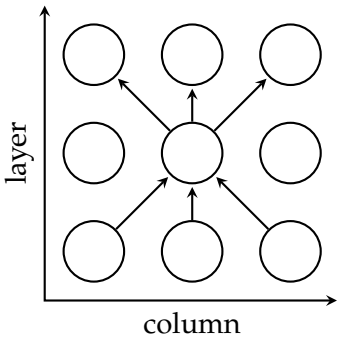

Figure 2.2.: The basic topology of TRIX. Only wires incident to the central node are shown; the same pattern is applied at other nodes.

2.3. Contribution: Empirical Proof of Quality

We propose a novel clock distribution topology that overcomes the above shortcoming of HEX. As in our topology nodes have in- and out-degrees of 3 and it is inspired by HEX, we refer to it as TRIX; see Fig. 2.2 for an illustration of the grid structure. Similar to HEX, the clock signal is propagated through layers, but for each node, all of its three in-neighbors are on the preceding layer. This avoids the pitfall of faulty nodes significantly slowing down the propagation of the signal. If at most one in-neighbor is faulty, each node still has two correct in-neighbors on the preceding layer, as demonstrated in Fig. 2.3. Hence, we can now focus on fault-free executions, because single isolated faults only introduce an additional uncertainty of at most $u \ll d$. Predictions in this model are therefore still meaningful for systems with rare and non-malicious faults.

The TRIX topology is acyclic, which conveniently means that self-stabilization is trivial to achieve, as any incorrect state is "flushed out" from the system.

Despite its apparent attractiveness and even greater simplicity, we note that this choice of topology should not be obvious. The fact that nodes do not check in with their neighbors on the same layer implies that the worst-case clock skew between neighbors grows as uH, where H is the number of layers and (for the sake of simplicity) we assume that the skew on the first layer (which can be seen as the "clock input") is 0, see Fig. 2.4. However, reaching the skew of d between neighbors on the same layer, which is necessary to give purpose to any link between them, takes many layers, at least $d/u \gg 1$ many.

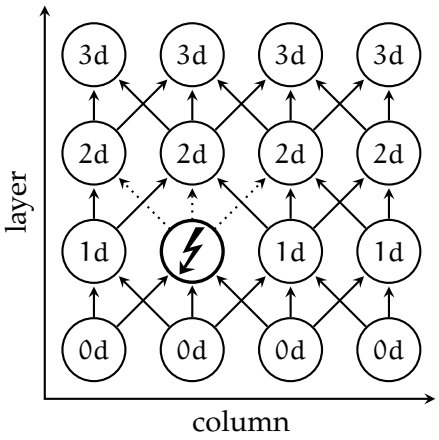

Figure 2.3.: A crashing node in a TRIX grid causes no significant skew, compared to Fig. 2.1. In fact, in absence of uncertainty, isolated crashes can be ignored entirely.

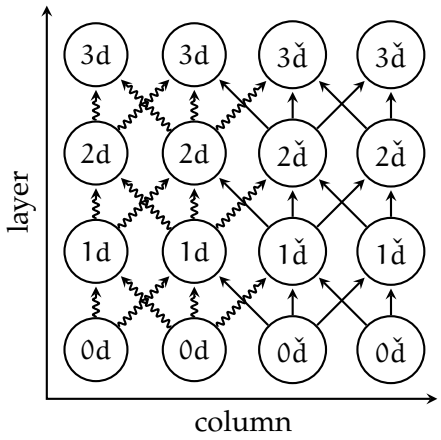

Figure 2.4.: Worst-case assignment of wire delays causing large skew for TRIX. Squiggly lines indicate slow wires, straight ones fast wires. The symbol $\breve{\mathrm{d}}$ stands for $\mathrm{d} - \mathrm{u}$.

This is in contrast to HEX, where the worst-case skew is bounded, but more easily attained. When assuming that delays are chosen in a typical-case manner, our statistical experiments show that it likely takes much more than d/u layers before this threshold of is reached. Accordingly, TRIX has no need for links within the same layer, for any practical height H, resulting in smaller in- and out-degrees compared to HEX.

The main focus of this work lies on statistical experiments with the goal of estimating the performance of a TRIX grid as clock distribution method[3]. Note that this is largely[4] dominated by the skew between adjacent nodes in the grid, as these will drive circuitry that needs to communicate. While the worst-case behavior is easy to understand, it originates from a very unlikely configuration, where one side of the grid is entirely slow and the other is fast, see Fig. 2.4. In contrast, correlated but gradual changes will also result in spreading out clock skews – and any change that affects an entire region in the same way will not affect local timing differences at all. This motivates to study the extreme case of independent noise on each link in the TRIX grid. Choosing a simple abstraction, we study the random process in which each link is assigned either delay 0 or delay 1 by an independent, unbiased coin flip.

[3]The topology itself was invented by Christoph Lenzen, with whom the corresponding paper [LW20a] was co-authored. The implementation, simulation, double-checking, statistical and stochastic analysis are done by Ben Wiederhake.

[4]Skews over longer distances are relevant for long-range communication, but have longer communication delays and respective uncertainties. This entails larger buffers even in absence of clock skew. We briefly show that the TRIX grid appears to behave well also in this regard.

For simplicity, we assume "perfect" input, i.e., each node on the initial layer signals a clock pulse at time 0, and that the grid is infinitely wide.[5] By induction over the layers, each node is then assigned the second largest value out of the three integers obtained by adding the respective link delay to the pulse time of each of its in-neighbors. We argue that this abstraction captures the essence of (independent) noise on the channels.

Due to the lack of applicable concentration bounds for such processes, we study this random process by extensive numerical experiments. Our results provide evidence that TRIX behaves surprisingly well in several regards, exhibiting better concentration than one might expect. First and foremost, the skew between neighbors appears to grow extremely slowly with the number of layers H. Even for 2000 layers, we never observed larger differences than 7 between neighbors on the same layer. Plotting the standard deviation of the respective distribution as a function of the layer, the experiments show a growth that is slower than *doubly logarithmic*, i.e. $\log\log H$. Second, for a fixed layer, the respective skew distribution exhibits an exponential tail falling as roughly $e^{-\lambda|x|}$ for $\lambda \approx 2.9$. Third, the distribution of pulsing times as a function of the layer (i.e., when a node pulses, not the difference to its neighbors) is also concentrated around its mean (which can easily be shown to be $H/2$), where the standard deviation grows roughly as $H^{1/4}$.

To support that these results are not simply artifacts of the simulation, i.e., that we sampled sufficiently often, we make use of the Dvoretzky-Kiefer-Wolfowitz (DKW) inequality [DKW56] to show that the underlying ground truth is very close to the observed distributions. In addition, to obtain tighter error bounds for our asymptotic analysis of standard deviations as function of H, we leverage the Chernoff bound (as stated in [MU05]) on individual values. We reach a high confidence that the qualitative assertion that the TRIX distribution exhibits surprisingly good concentration is well-founded. We conclude that the simple mechanism underlying the proposed clock distribution mechanism results in a fundamentally different behavior than existing clock distribution methods or naive averaging schemes.

2.4. Preliminaries and Notation

Model We model TRIX in an abstract way that is amenable to very efficient simulation in software. In this section, we introduce this model and discuss the assumptions and the resulting restrictions in detail.

The network topology is a grid of height H and width W. For finite W, the left- and rightmost column would be connected, resulting in a cylinder. To simplify, we choose $W = \infty$, because we aim to focus on the behavior in large systems. Note that a finite width will work in our favor, as it adds additional constraints on how skews can evolve over layers; in the extreme case of $W = 3$, in absence of faults skews could never become larger than 1. We refer to the grid nodes by integer coordinates (x, y), where $x \in \mathbb{Z}$ and $y \in \mathbb{N}_0$. Layer $0 \leqslant \ell \leqslant H$ consists of the nodes (x, ℓ), $x \in \mathbb{Z}$.

[5] Experiments with grids of bounded width suggest that reducing width only helps, while our goal here is to study the asymptotic behavior for large systems.

Layer 0 is special in that its nodes represent the clock source; they always *pulse* at time 0. Again, there are implicit simplifications here. First, synchronizing the layer 0 nodes requires a suitable solution – ideally also fault-tolerant – and cannot be done perfectly. However, our main goal here is to understand the properties of the clock distribution grid, so the initial skew is relevant only insofar as it affects the distribution. Some indicative simulations demonstrate that the grid can counteract "bad" inputs to some extent, but some configurations do not allow for this in a few layers, e.g. having all nodes with negative x coordinates pulse much earlier than those with positive ones. Put simply, this would be a case of "garbage-in garbage-out," which is not the focus of this study. A more subtle point is the unrealistic assumption that all layers of the grid have the same width. If the layer 0 nodes are to be well-synchronized, they ought to be physically close; arranging them in a wide line is a poor choice. Accordingly, arranging the grid in concentric rings (or a similar structure) would be more natural. This would, however, entail that the number of grid nodes per layer should increase at a constant rate, in order to maintain a constant density of nodes (alongside constant link length, etc.). However, adding additional nodes permits to distribute skews *better,* therefore our simplification acts against us.

All other nodes (x, ℓ) for $\ell > 0$ are TRIX nodes. Each TRIX node propagates the clock signal to the three nodes "above" it, i.e., the vertices $(x-1, y+1)$, $(x, y+1)$, and $(x+1, y+1)$. In the case of the clock generators, the signal is just the generated clock pulse; in case of the TRIX nodes, this signal is the forwarded clock pulse. Each node (locally) triggers the pulse, i.e., forwards the signal, when receiving the second signal from its predecessors; this way, a single faulty in-neighbor cannot cause the node's pulse to happen earlier than the first correct in-neighbor's signal arriving or later than the last such signal.

Pulse propagation over a comparatively long distance involves delays, and our model focuses on the uncertainty on the wires. Specifically, we model the wire delays using i.i.d. random variables that are fair coin flips, i.e., attain the values 0 or 1 with probability $1/2$ each.[6] This reflects that any (known expected) absolute delay does not matter, as the number of wires is the same for any path from layer 0 (the clock generation layer) to layer $\ell > 0$; also, this normalizes the uncertainty to 1.

Formalizing the above, for each wire from the nodes $(x + c, y)$ with $c \in \{-1, 0, +1\}$ to node $(x, y + 1)$, we define w_c to be the wire delay. We further define the time $t_c := d(x+c, y) + w_c$ at which node $(x, y+1)$ receives the clock pulse. Then node $(x, y+1)$ fires a clock pulse at the median time $t := \text{median}\{t_{-1}, t_0, t_{+1}\}$. As we assume that all clock generators (i.e., nodes with $y = 0$) fire, by induction on y all $d(x, y)$ are well-defined and finite.

This model may seem idealistic, especially our choice of the wire delay distribution. However, in Appendix A.1 we argue that this is not an issue.

[6] This model choice is restrictive in that it deliberately neglects correlations. A partial justification here is the expectation that (positive) correlations are unlikely to introduce local "spikes" in pulse times, i.e., large skews. However, we acknowledge that this will require further study, which must be based on realistic models of the resulting physical implementations.

We concentrate on two important metrics to analyze this system: absolute delay and relative skew. The total delay $d(x, y)$ of a node (x, y) (usually with $y = H$) is the time at which this node fires. The relative skew $s^\delta(x, y)$ in horizontal distance δ is the difference in total delay; i.e., $s^\delta(x, y) := d(x + h, y) - d(x, y)$. Our main interest lies in the behavior of the random variables $d(H) := d(0, H)$, i.e. the delay at the top of the grid, and the random variable $s(H) := s^1(0, H)$, i.e. the relative skew between neighboring nodes.

Further Notation We write $\mathcal{N}(\mu, \sigma^2)$ to denote the normal distribution with mean μ and standard deviation σ.

Given the average sample value $\bar{v} = \sum_{i=1}^{n} v_i/n$ over a set of n sample values v_i, we compute the *empiric* standard deviation as $(\sum_{i=1}^{n}(v_i - \bar{v})^2/(n-1))^{1/2}$.

We denote the cumulative distribution function of a random variable X as $C[X]$; e.g. the term $C[X](x)$ denotes the probability that $X \leqslant x$. We denote the inverse function by $C^{-1}[X]$.

A quantile-quantile-plot relates two distributions A and B. The simplest definition is to plot the domain of A against the domain of B, using the function $C^{-1}[B](C[A](x))$.

We define the sample space Ω as the set of possible specific assignments of wire delays. If we need to refer to the value of some random variable X in a sample $s \in \Omega$, we write $X[s]$.

Chapter 3

TRIX: Low-Skew, Fault-Tolerant

In this chapter we show experimental data and the theory behind it in order to demonstrate that the TRIX is not only a good candidate pulse propagation mechanism, but also behaves better than one might reasonably expect.

In particular, we will show that the delay is identically and normally distributed, but not independently: In fact, the skew (difference of neighboring delays) is surprisingly small.

This chapter is an extended version of the brief announcement [LW20a] at SSS 2020, with full proofs, more details about the methodology, and more explanations of the data. A stand-alone version [LW20b] is available.

3.1. Delay is Tightly Concentrated

3.1.1. Empiric Analysis

In this subsection, we examine $\mathrm{d}(2000)$, which we formally defined in Sect. 2.4. Recall that $\mathrm{d}(2000)$ is the delay at layer 2000. The results are similar for other layers and, as we will show in Sect. 3.1.3, do change slowly with increasing H.

For reference, consider a simpler system consisting of a sequence of nodes arranged in a line topology, where each node transmits a pulse once receiving the pulse from its predecessor. In this system, the delay at layer H would follow a binomial distribution with mean $H/2 = 1000$ and standard deviation $\sqrt{H}/2 = \sqrt{2000}/2 \approx 22$. Recall that by the central limit theorem, for large H this distribution will be very close to a normal distribution with the same mean and standard deviation. In particular, it will have an exponential tail.

Fig. 3.1 shows the estimated probability mass function of $\mathrm{d}(2000)$. The data was gathered using 25 million independent simulations; in Appendix A.1 we discuss how we ensured that the simulations are correct.

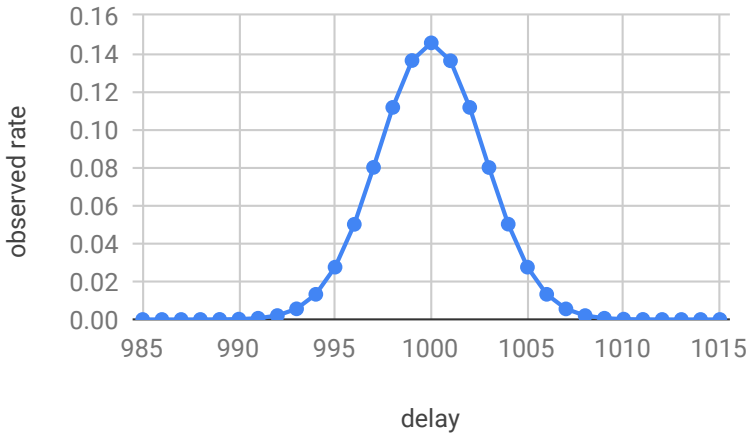

Figure 3.1.: Estimated probability mass function of $\mathrm{d}(2000)$.

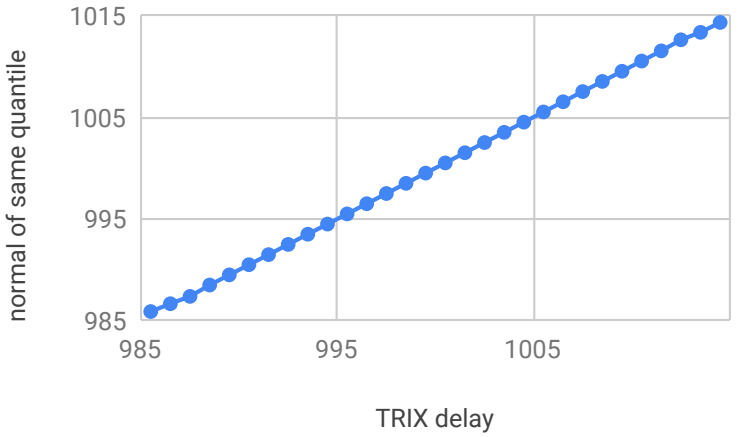

Figure 3.2.: Quantile-quantile plot of $\mathrm{d}(2000)$ against $\mathrm{N}(1000, 2.741^2)$.

Observe that the shape looks like a normal distribution, as one might expect. However, it is concentrated much more tightly around its mean than for the simple line topology considered above: The empiric standard deviation of this sample is only 2.741.

Fig. 3.2 uses a quantile-quantile-plot to show that $\mathrm{d}(2000)$ and $\mathcal{N}(1000, 2.741^2)$ seem to be close to identical, as indicated by the fact that the plot is close to a straight line. The extremes are an exception, where numerical and uncertainty issues occur. Of course, they cannot be truly identical, even if our guess 2.741 was correct: $\mathrm{d}(2000)$ is discrete and has a bounded support, in contrast to $\mathcal{N}(1000, 2.741^2)$. However, this indicates some kind of connection we would like to understand better.

The Dvoretzky-Kiefer-Wolfowitz inequality [DKW56] implies that the true cumulative distribution function must be within 0.0003255 (i.e., 8139 evaluations) of our measurement with probability $1 - \alpha = 99\%$. For the values with low frequency however, the Dvoretzky-Kiefer-Wolfowitz inequality yields weak error bounds.

Instead, we use that Chernoff bounds can be applied to the random variable X_k given by sum of variables $X_{i,k}$ indicating whether the i-th evaluation of the distribution attains value k. We then vary p_k, the unknown probability that the underlying distribution attains value k, and determine the threshold p_{max} at which Chernoff's bound shows that $p_k \geqslant p_{max}$ implied that our observed sample had an a-priori probability of at most α'; the same procedure is used to determine the threshold p_{min} for which $p_k \leqslant p_{min}$ would imply that the observed sample has a-priori probability at most α'. Note that we can also group together multiple values of k into a single bucket and apply this approach to the frequency of the overall bucket. This can be used to address all values with frequency 0 together. Finally, we chose α' suitably such that a union bound over all buckets[1] yields the desired probability bound of $1 - \alpha = 99\%$ that *all* frequencies of the underlying distribution are within the computed error bounds.

Due to our large number of samples, the resulting error bars for the probability mass function are so small that they cannot be meaningfully represented in Fig. 3.1; in fact, on the interval $[990, 1010]$ the error bars are at most 8.05% (multiplicative, not additive), and on the interval $[994, 1006]$ at most 0.93%. On the other hand, data points outside $[990, 1010]$ in Fig. 3.2 should be taken as rough indication only.

In other words, we have run a sufficient number of simulations to conclude that the ground truth is likely to be very close to a binomial (i.e., essentially normal) distribution with mean $H/2 = 1000$ and standard deviation close to 2.741; the former is easily shown, which we do next.

3.1.2. Stochastic Analysis

Lemma 1. $\mathbb{E}[d(H)] = H/2$.

Proof. Consider the bijection $f\colon \Omega \to \Omega$ on the sample space given by $f(s) = \bar{s}$, i.e., we exchange all delays of 0 for delays of 1 and vice versa. We will show that for a sample s with $d(2000)[s] = \delta$ it holds that $d(2000)[f(s)] = H - \delta$. As the wire delays are u.i.d., all points in Ω have the same weight under the probability measure, implying that this is sufficient to show that $\mathbb{E}[d(H)] = H/2$.

We prove by induction on y that $d(x, y)[f(s)] = y - d(x, y)[s]$ for all $0 \leqslant y \leqslant H$ and $x \in \mathbb{Z}$; by the above discussion, evaluating this claim at $(x, y) = (0, H)$ completes the proof. For the base case of $y = 0$, recall that $d(x, 0)[f(s)] = d(x, 0)[s] = 0$ by definition.

For the step from y to $y + 1$, consider the node at $(x, y + 1)$ for $x \in \mathbb{Z}$. For $c \in \{-1, 0, +1\}$, the wire delay w_c from $(x + c, y)$ to $(x, y + 1)$ satisfies $w_c[f(s)] = 1 - w_c[s]$ by construction. By the induction hypothesis, $d(x + c, y)[f(s)] = y - d(x + c, y)[s]$. Together, this yields that $(x, y + 1)$ receives the pulse from $(x + c, y)$ at time

$$d(x + c, y)[f(s)] + w_c[f(s)] = y + 1 - (d(x + c, y)[s] + w_c[s]).$$

[1] That is, the number of non-zero values to which we do not apply Dvoretzky-Kiefer-Wolfowitz plus one (for values of frequency 0).

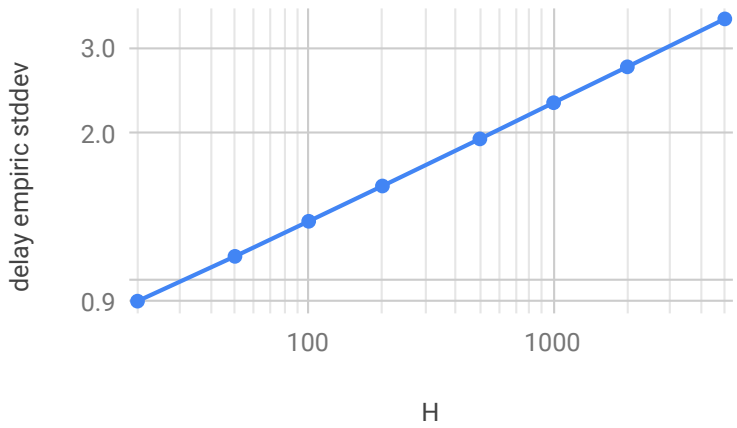

Figure 3.3.: Log-log plot of the empiric standard deviation of $d(H)$ as a function of H.

We conclude that

$$\begin{aligned} d(x, y+1)[f(s)] &= \operatorname*{median}_{c\in\{-1,0,1\}} \{y+1-(d(x+c,y)[s]+w_c[s])\} \\ &= y+1-\operatorname*{median}_{c\in\{-1,0,1\}} \{d(x+c,y)[s]+w_c[s]\} \\ &= y+1-d(x,y+1)[s]. \end{aligned}$$

□

3.1.3. Asymptotics in Network Depth

As discussed earlier, forwarding the pulse signal using a line topology would result in $d(H)$ being a binomial distribution with mean $H/2$ and standard deviation $\sqrt{H}/2$. For $d(2000)$, we observe a standard deviation that is smaller by about an order of magnitude.

Running simulations and computing the empiric standard deviation for various values of H resulted in the data plotted in Fig. 3.3 as a log-log plot. Using the technique discussed above, we can compute error bars.[2] Again, the obtained error bounds cannot be meaningfully depicted; errors are about 1% (multiplicative, not additive) with probability $1-\alpha = 99\%$. The largest error margin is at 200 layers, with 1.35%.

Fig. 3.3 suggests a polynomial relationship between standard deviation σ and grid height H. The slope of the line is close to $1/4$, which suggests $\sigma \sim H^\beta$ with $\beta \approx 1/4$.

This is a quadratic improvement over the reference case of a line topology.

[2]This relies on the exponential tails demonstrated in Sect. 3.1.1. Without this additional observation, the error bars for all possible skews (including large skews like $H/2$) would cause large uncertainty in the standard deviation.

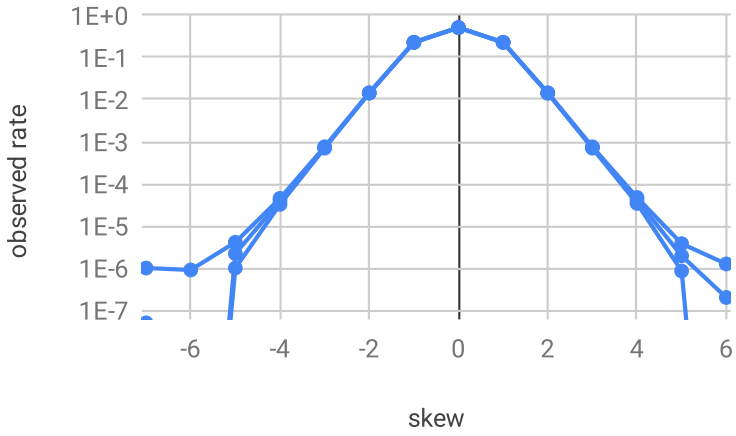

Figure 3.4.: The estimated probability mass function for $s(2000)$, with a logarithmic y-axis. The error bounds are only visible at the fringes.

3.2. Skew is Tightly Concentrated

3.2.1. Empiric Analysis

In this subsection, we study $s(2000)$, which we formally defined in Sect. 2.4. Recall that $s(2000)$ is the skew at layer 2000 between neighboring nodes. As we will show in Sect. 3.2.3, the behavior for other layers is very similar. In particular, the skews increase stunningly slowly with H.

We gathered data from 20 million simulations[3], and see a high concentration around 0 in Fig. 3.4, with roughly half of the probability mass at 0. Note that again the error bars cannot be represented meaningfully in Fig. 3.4; in fact, on the interval $[-3, +3]$ the error bars are at most 6.1% (multiplicative, not additive), and on the interval $[-2, +2]$ at most 1.4%.

Observe that the skew does not follow a normal distribution at all: The probability mass seems to drop off exponentially like $e^{-\lambda|x|}$ for $\lambda \approx 2.9$ (where x is the skew), and not quadratic-exponentially like $e^{-x^2/(2\sigma^2)}$, as it would happen in the normal distribution. The probability mass for 0 is a notable exception, not matching this behavior.

The Dvoretzky-Kiefer-Wolfowitz inequality [DKW56] implies that the true cumulative distribution function must be within 0.0005147 (i.e., 5147 evaluations) of our measurement with probability $1 - \alpha = 99\%$. In particular, observing skew 6 twice without observing skew -6 is well within error tolerance.

[3]Curiously, we saw skew -7 exactly once, skew -6 never, skew +6 four times. Further investigation showed this to be a fluke, but we want to avoid introducing bias by picking the "nicest" sample.

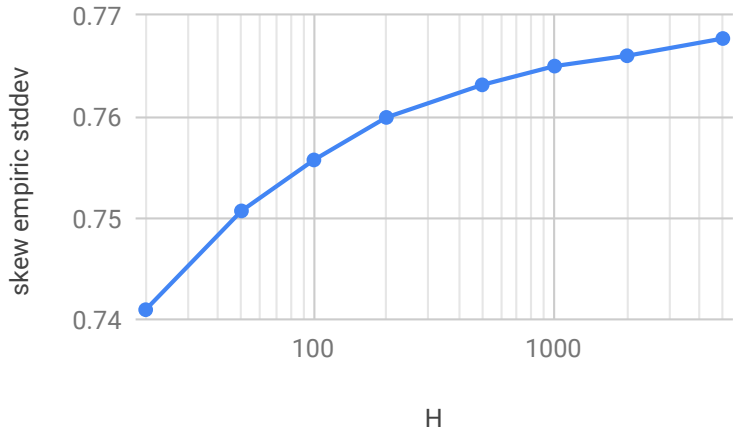

Figure 3.5.: Empiric standard deviation of $s(H)$ as a function of H, as a log-lin plot.

3.2.2. Stochastic Analysis

First, we observe that the skew is symmetric with mean 0. This is to be expected, as the model is symmetric. It also readily follows using the same argument as used in the proof of Lem. 1.

Corollary 2. $s(H)$ *is symmetric with* $\mathbb{E}[s(H)] = 0$.

Proof. Consider the bijection used in the proof of Lem. 1. As shown in the proof, for any $(x, y) \in \mathbb{Z} \times \mathbb{N}_0$ and $s \in \Omega$, we have that

$$\begin{aligned} d(x,y)[f(s)] - d(x+1,y)[f(s)] &= y - d(x,y)[s] - (y - d(x+1,y)[s]) \\ &= -(d(x,y)[s] - d(x+1,y)[s]). \end{aligned}$$

□

Next we prove that the worst-case skew on layer H is indeed H, c.f. Fig. 2.4.

Lemma 3. *There is a sample* s *such that* $s(H)[s] = H$.

Proof. In s, we simply let all wire delays w_i of wires leading to a node with positive x be 1, and let all wire delays w_j of wires leading into a node with non-positive x be 0. A simple proof by induction shows that for positive x we get $d(x, y)[s_1] = y$ and for non-positive x we get $d(x, y)[s_1] = 0$. We conclude that $s(H)[s] = d(1, y)[s] - d(0, y)[s] = H$. □

The constructed sample is not the only one which exhibits large skew. For example, simultaneously changing the delay of all wires between $x = 0$ and $x = 1$ does not affect times when nodes pulse. Moreover, for all $x \notin \{0, 1\}$, we can concurrently change the delay of any one of their incoming wires without effect. It is not hard to see that the total probability mass of the described samples is small. We could not find a way to show that this is true in general; however, the experiments from Sect. 3.2.1 strongly suggests that this is the case. (Hence our question: How to approach statistical problems like this?)

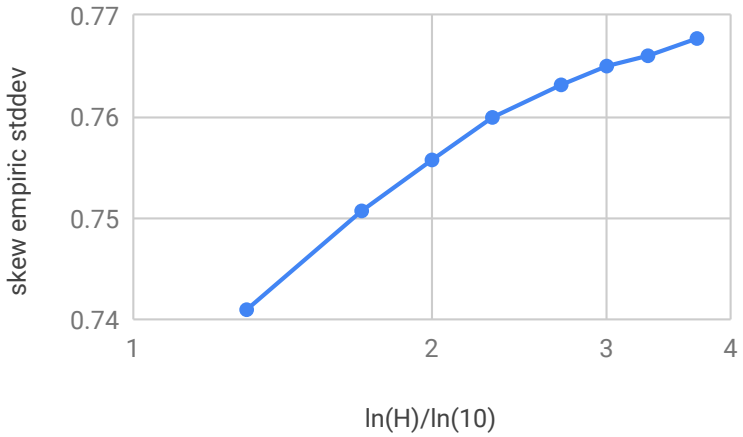

Figure 3.6.: The data of Fig. 3.5 in a loglog-lin plot.

3.2.3. Asymptotics in Network Depth

Again we lack proper models to describe the asymptotic behavior, and instead calculate the empiric standard deviation from sufficiently many simulations.

Fig. 3.5 shows that the skew remains small even for large values of H.[4] Note that the X-axis is logarithmic. Yet again, showing the error bars calculated using Chernoff bounds and DKW (Dvoretzky-Kiefer-Wolfowitz) is not helpful, as we get relative errors of about 0.6% with 99% confidence. The largest error margin is at 100 layers and below, with an error about 0.95%.

This suggests that the standard-deviation of $s(H)$ grows strongly sub-logarithmically, possibly even converges to a finite value. In fact, the data indicates a growth that is significantly slower than logarithmic: in Fig. 3.6, the X-axis is doubly logarithmic. Thus, the plot suggests that $s(H) \in O(\log \log H)$.

Note that if we pretended that adjacent nodes exhibit independent delays, the skew would have the same concentration as the delay. In contrast, we see that adjacent nodes are tightly synchronized, implying strong correlation.[5]

3.2.4. Asymptotics in Horizontal Distance

So far, we have limited our attention to the skew between neighboring nodes. In contrast, at horizontal distances $\delta \geqslant 2H$, node delays are independent, as they do not share any wires on any path to any clock generator. Therefore, the skew would be given by independently sampling twice from the delay distribution and taking the difference. It remains to consider how skews develop for smaller horizontal distances.

[4] A standard deviation below 1 link delay uncertainty is not unreasonable: Two circuits clocked over independent links, hop-distance 1 from a perfect clock source, have $\sigma(skew) = \sqrt{\mathbb{E}[0.5^2]} = 0.5$ uncertainties.

[5] For the sake of brevity, we do not show respective plots.

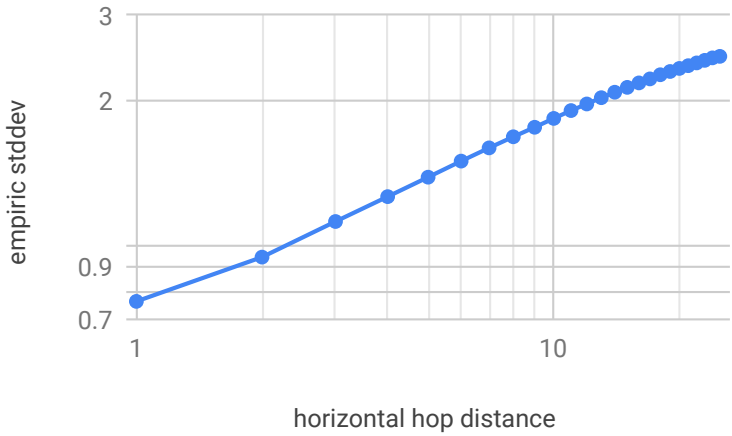

Figure 3.7.: Empiric standard deviation of $s^{\delta}(0, 500)$ as a function of horizontal distance δ in a log-log plot.

In Fig. 3.7, we see that the skew grows steadily with increasing δ. These simulations were run with $H = 500$ for higher precision and execution speed. Note that the standard deviation of the skew converges to roughly $\sqrt{2} \cdot 1.95 \approx 2.75$, i.e., $\sqrt{2}$ times the standard deviation of the delay at $H = 500$, as δ approaches $2 \cdot 500 = 1000$. This holds because the sum of independent random variables has variance equal to the sum of variances of the variables. As before, depicting the error bars determined by our approach is of no use, as relative errors are about 1.6%. The largest error margin is at horizontal distance 25, with 1.79%.

The small relative errors indicate that the log-log representation of the data is meaningful, and suggests that the standard deviation increases roughly proportional to δ^{γ} for $\gamma \approx 1/3$. It is not surprising that the slope falls off towards larger values, as we know that the curve must eventually flatten and become constant for $\delta \geqslant 2H = 1000$. Overall, we observe that the correlation of skew at a distance is stronger than expected, specifically $\gamma \approx 1/3$ instead of the expected $\gamma \approx 1/2$ for small δ.

Chapter 4

Conclusion

In this part, we studied the behavior of the TRIX grid under u.i.d. link delays, using statistical tools. Our results clearly demonstrate that the TRIX grid performs much better than one would expect from naive solutions, and thus should be considered when selecting a fault-tolerant clock propagation mechanism.

Concretely, our simulation experiments show that the delay as function of the distance H from the clock source layer is close to normally distributed with a standard deviation that grows only as roughly $H^{1/4}$, a qualitative change from e.g. a line topology that would not be achieved by averaging or similar techniques. Moreover, the skew between neighbors in the same layer is astonishingly small. While not normally distributed, there is strong evidence for an exponential tail, and the standard deviation of the distribution as function of H appears to grow as $O(\log \log H)$. Checking the skew over larger horizontal counts d (within the same layer) shows a less surprising pattern. However, still the increase as function of d appears to be slightly slower than $\sqrt{d}$.

These properties render the TRIX grid an attractive candidate for fault-tolerant clock propagation, especially when compared to clock trees. We argue that our results motivate further investigation, considering correlated delays based on measurements from physical systems as well as simulation of frequency and/or non-white phase noise. In addition, the impact of various other factors need to be studied, such as many isolated faults, few correlated faults, the physical realization as a less regular grid with a central cluster of clock sources, and the imperfection of the input provided by the clock source(s).

Last but not least, we would like to draw attention to the open problem of analyzing the stochastic process we use as an abstraction for TRIX. This is also the reason why we lack a purely mathematical analysis. While our simulation experiments are sufficient to demonstrate that the exhibited behavior is highly promising, gaining an understanding of the underlying cause would allow making qualitative and quantitative predictions beyond the considered setting. As both the nodes' decision rule and the topology are extremely simple, one may hope for a general principle to emerge that can also be applied in different domains.

Part II.

Distance-r Graph Cover

Chapter 5

Introduction

In this chapter, we revisit the popularity of sparse graphs and graph cover problems, how they generalize, and give a summary of preliminaries and our contribution.

5.1. Motivation

Studying sparse graphs is crucial for computer science in several ways. First, many real networks are sparse. For instance, the recent analysis of Schmid et al. [AFS20] on the dataset of Rost [Ros19] shows that many real networks have tree-width at most 5, meaning they are quite sparse and well-structured graphs. Sparse graphs are important on the theory side, too: In the analysis of dense graphs, we often employ sparsification techniques and solve the problem on the sparse instance in order to understand how the dense instances work.

Sparsity is usually measured by means of an edge density of some kind: This can be the edge density of the entire graph, of certain subgraphs, shallow minors of the graph, or every minor of the graph, or similar metrics. Sometimes we also consider topological or combinatorial attributes, famous examples being the classes of planar and bounded genus graphs. Sometimes, in addition to the edge density requirement, we restrict the degree of each vertex to get bounded degree graphs. The class of bounded degree graphs has been studied extensively from the distributed perspective; most prominently in the form of locally-checkable labeling (LCL) problems [Bal+21; Bra+17].

In the sequential setting, many APX-hard[1] covering problems can be well approximated if they are limited to the class of sparse graphs. Hence, it is interesting to understand how sparsity enables better distributed algorithms in distributed computing models, which could mean improving the approximation factor or reducing the number of communication rounds.

[1] Loosely speaking, these problems have no PTAS. A PTAS is a family of algorithms that run in polynomial time and compute an approximation of the optimum solution, where the approximation ("apx.") factor is arbitrarily close to 1.

In the distributed setting every node is considered as a processor that can communicate with its neighbors per synchronized rounds. The aim is to reduce the total number of such rounds while providing a *good* solution.

In this work, we continue the line of study on sparse graphs and explore the algorithmic properties of a wide range of sparse graphs, namely the class of graphs of bounded expansion, with an extra combinatorial property: sparse graphs of high girth. The girth of the graph is the length of its shortest cycle, or infinity if no cycle exists.

Girth plays a role in understanding structural properties of graphs. Sparse graphs of high girth appear in important constructions such as spanner graphs [Alt+93]. Similarly random graphs have only a few short cycles and at the same time, depending on their parameters, they could be quite sparse. In such a graph class (we will formally specify them later) we study several central covering problems in their most generic form: distance-r covering problems.

As a result, we answer some more cases of the famous question of Naor and Stockmeyer: "What can be computed locally?" [NS93] More precisely, we show that the problems 1. DISTANCE-r DOMINATING SET 2. DISTANCE-r CONNECTED DOMINATING SET 3. DISTANCE-r VERTEX COVER and 4. DISTANCE-r CONNECTED VERTEX COVER. on the considered graph class have constant factor approximation in a constant number of rounds in the CONGEST model of computation, if r (the distance) is constant. Whenever feasible, we also give the precise relation to r.

The aforementioned problems are hard in general graphs when it comes to distributed settings. For instance there is no constant-factor approximation CONGEST algorithm for the distance-2 dominating set even if we let the algorithm to run for $o(n^2)$ rounds [Bar+20], where n is the number of nodes. Observe that in order to exchange information, two nodes require at most $O(n)$ rounds of communication, however, for such a restricted case of the problem (only distance-2) the amount of data to be transferred is too big to fit in one message that respects the bandwidth of the network. Hence it needs $\Omega(n^2)$ rounds of computation to merely approximate the optimum solution. Thus, a natural approach to progress on such problems is to consider them on restricted graph families, as we will do in the following chapters.

The distance-1 versions of the above problems play important roles in theoretical computer science. They are among Karp's 21 NP-complete problems. They also have many other implications: dominating set is one of the base cases to show inapproximability within $\omega(\log n)$ factor, and one of the central problems to derive $W[2]$-hardness results. Vertex cover is one of the well known APX-hard problems, and a textbook example of fixed parameter tractability. This raises the question how the more general distance-r versions behave; we answer this question for the case of a specific, sparse graph class.

5.2. Related Work

In distributed settings, for the dominating set problem on general graphs, Kuhn et al. [DKM19] provided a $(1+\epsilon)(1+\log(\Delta+1))$-approximation in $f(n)$ rounds. In this bound, Δ is the maximum degree and $f\colon \mathbb{N} \to \mathbb{N}$ is the number of rounds that is needed

to compute the network decomposition [Awe+89; Awe+92; Gha19]. Given the recent breakthrough result of Rozhon and Ghaffari [RG20], the aforementioned algorithm runs in a polylogarithmic number of rounds.

For the vertex cover problem, in graphs of bounded degree, the result of Åstrand and Suomela [AS10] provides a constant factor approximation in a constant number of rounds, and later, Bar-Yehuda et al. [BCS16] provided a constant factor approximation in a sublogarithmic number of rounds. This is complemented by the lower bound of Kuhn, Moscibroda and Wattenhofer (KMW) [KMW16] shows that a logarithmic approximation in almost sublogarithmic time for the vertex cover problem and the minimum dominating set (and some other covering problems) is impossible in general graphs, even in the LOCAL model of computation. Their lower-bound graph for vertex cover has high girth, but it is also of unbounded arboricity (more generally unbounded average degree). However, even for the highly restricted class of bounded degree two colored bipartite graphs, Göös and Suomela [GS14] showed that it is not possible to provide a $(1+\epsilon)$-approximation of the vertex cover problem in a sublogarithmic number of rounds.

We are interested in what happens in between graph classes. If we consider a graph class of very high girth and very low edge density, e.g. trees, then these problems are easy to approximate in zero rounds: take all non-leaf vertices. The above observations raise the questions: In which graph classes does the problem admit a constant approximation factor in a constant number of rounds? What about distance-r problems?

For the dominating set problem, Lenzen et al. [CHW08; LPW13] provided the first constant-factor approximation in a constant number of rounds in planar graphs, which was improved by Czygrinow et al. [CHW08]. Later, Amiri et al. [AS16; ASS16] provided a new analysis method to extend the result of Lenzen et al. to bounded genus graphs. This has been gradually generalized to excluded minor graphs [Czy+18] and bounded expansion [KSV21].

A natural generalization of excluded minor graphs is the class of bounded expansion graphs. Simply put, bounded expansion graphs only exclude minors on a localized level; there may still be large clique minors in the graph.

On graphs of bounded expansion, only a logarithmic time constant factor approximation is known for the dominating set problem; however, it seems that one can extend the algorithm of [Czy+18] to bounded expansion graphs, as they only consider "local" minors. If we go slightly beyond these graphs, to graphs of bounded arboricity (where every subgraph has a constant edge density), the situation is worse: only an $O(\log \Delta)$-approximation in $O(\log n)$ rounds was known. There was a $O(\log n)$ round $O(1)$-approximation in such graphs; however, this algorithm is randomized [LW10], only recently it has been proven that it can be performed deterministically [Ami21].

All these results are about the distance-1 dominating set problem. Significantly fewer work exists on the topic of the distance-r dominating set problem. We are only aware of the algorithm of Amiri et al. [Ami+18] for bounded expansion graphs that provides a constant factor approximation in a logarithmic number of rounds.

5.2.1. Existing Approaches for Distance-r Dominating Set

There are several existing approaches one might employ to tackle the problem: 1. Take the r-th power of the graph and go back to the distance-1 dominating set, 2. Decentralize existing decomposition methods in the sequential setting and employ them, 3. Use existing fast distributed graph decomposition methods for sparse graphs. In the following, we explain how all of the above approaches, without introducing new ideas, are impractical in providing sublogarithmic round algorithms for distance-r covering problems.

For the first approach, we lose the sparsity of the graph already on stars. Hence, we cannot rely on existing algorithms for solving the domination problem in sparse graphs.

Decentralizing the existing sequential decomposition methods does not seem promising if one hopes to achieve anything better than logarithmic rounds: To the best of our knowledge, every such sequential decomposition already consumes polylogarithmically many rounds. Even assuming the decomposition is given in advance, such methods handle the clusters sequentially; however the number of clusters is usually at least logarithmic, requiring at least logarithmically many rounds.

For the third approach, these existing fast distributed graph decomposition methods are mostly inspired by existing methods in classical settings, like Baker's method [Bak94]; this includes, e.g., the $O(\log^* n)$ round algorithm of [CHS06]. The idea is to partition a sparse graph into connected clusters such that each cluster has a small diameter and the number of in-between cluster edges is small. Then, find the optimal solution inside each cluster efficiently, and because the number of edges between a pair of clusters is small, we can just ignore conflicts.

However, this fails already for distance-2 domination, since the number of edges between clusters in the power graph is high. Recent research has shown that there is a lower bound of $\tilde{\Omega}(n^2)$ for distance-2 dominating set, both for solving it exactly [Bac+19], and even for constant-factor approximations [Bar+20].

Also, we cannot rely on global properties (such as tree decomposition) like in the sequential setting, since this increases the number of rounds to the diameter of the graph, which can easily be superlogarithmic.

Therefore, any distributed algorithm that solves distance-r covering problems has to exploit special properties of the underlying graph class or problem, motivating our choice of sparse high-girth graphs.

5.3. Contribution: Algorithmic Upper Bound and Unconditional Lower Bound

Throughout the paper, we assume that a graph $G = (V, E)$ is given. In the (distance-1) dominating set problem, we seek a set $D \subseteq V$ such that every other vertex of G is a neighbor of a vertex in D. In the connected version of the problem, the induced graph of G on the vertices of $\hat{D}$ should be connected.

In the vertex cover problem, we seek a set C of vertices of the graph such that every edge in the graph is incident to at least one of the vertices in C. Similarly, the connected version of the problem asks for a vertex cover $\hat{C}$ such that the induced graph of C on G is connected. In all of the above problems we would like to minimize the size of the corresponding set.

The distance-r versions of covering problems are defined similarly to the classic versions: for the dominating set problem, we consider the distance-r neighborhood. In the distance-r vertex cover problem, we say that vertex v covers edge e if and only if vertex v is within distance r of both endpoints. Observe that for $r = 1$, these distance-r versions are equivalent to the classic versions.

Our main algorithmic contribution is that for constant r, the distance-r dominating set problem admits a constant factor approximation in a constant number of rounds in sparse high-girth graphs. Furthermore, we obtain non-trivial approximation guarantees for super-constant r (i.e. $r \in o(\log^* n)$).

More specifically, the algorithm we developed for the distance-r dominating set problem can be boiled down to this: Each vertex chooses its dominator to be the neighbor that has a maximum degree in the r-th power graph. This is the algorithm that proves the upper bound Thm. 4. We generalize the algorithm to handle the Distance-r Connected Dominating Set (proven by Lem. 30), Distance-r Vertex Cover (Lem. 32), and Distance-r Connected Vertex Cover (Cor. 34) problems.

Theorem 4. *Let $\mathcal{C}$ be a graph class of bounded expansion $f(r)$ and girth at least $4r + 3$. There is a CONGEST algorithm that runs in $O(r)$ rounds and provides an $O(r \cdot f(r))$-approximation of minimum distance-r dominating set on $\mathcal{C}$, even if $r \geqslant 2$ is non-constant in n.*

Given that the distance-r dominating set problem is equivalent to the dominating set problem of the r-th power of the input graph, the algorithm can also be used to provide a constant factor approximation in a non-trivial class of dense graphs for covering problems. There are very few known algorithms with a constant factor guarantee in a constant number of rounds on non-trivial dense graphs, e.g., the algorithm of Schneider et al. [SW08] on graphs of bounded independence number (for the independent set and the connected dominating set problem) partially falls into this category.

To show that our upper bound is reasonably tight, we provide a lower bound as well. This we obtain by a reduction from a lower bound for independent set on the ring [CHW08; LPW13] to the distance-r dominating set on the ring (naturally, a ring with high girth). More formally we prove Thm. 5.

Theorem 5. *Assume an arbitrary but fixed $\delta > 0$ and $r > 1$, with $r \in o(\log^* n)$. Then, there is no deterministic LOCAL algorithm that finds in $O(r)$ rounds a $(2r + 1 - \delta)$-approximation of distance-r dominating set for all $G \in \mathcal{C}$, where $\mathcal{C}$ is the class of cycles of length $\gg 4r + 3$.*

5.4. Preliminaries and Notation

We assume basic familiarity with graph theory. In the following, we introduce basic graph notation to avoid ambiguities. We refer to the book by Diestel [Die12] for further reading.

Graph, Path, Distance: We only consider simple, connected, undirected graphs $G = (V, E)$, with $n = |V|$. A path $P = (v_0, v_1, \ldots, v_l)$ in G is a non-empty sequence of vertices such that no vertex repeats, and for every $0 \leqslant i < l$ it holds that there is an edge in $\{v_i, v_{i+1}\} \in E$. We say that path P has length l, i.e. the number of edges.[2] For $u, v \in V$, define the distance $d(u, v)$ as the length of the shortest path between the vertices u and v. For a set $S \subseteq V$, we define $d(u, S)$ as the maximum distance between vertex u and any vertex in S.

Neighborhood, Distance-r: Two vertices $u, v \in V$ are neighbors in G if there is an edge $e = \{u, v\} \in E$. We extend this definition to the distance-r neighborhood $N^r[v]$ and open distance-r neighborhood $N^r(v)$ of a vertex v in the following way:

$$N^r[v] := \{u \in V \mid d(u, v) \leqslant r\} \qquad N^r(v) := N^r[v] \setminus \{v\}$$

Similarly for a set $S \subseteq V$ we define:

$$N^r[S] := \bigcup_{v \in S} N^r[v] \qquad N^r(S) := N^r[S] \setminus S$$

Girth, Radius: The girth g of a graph G is the length of its shortest cycle, or ∞ if acyclic. The radius R of G is the minimum integer R for which $\exists v \in V : N^R[v] = V$.

Distance-r Dominating Set: A set $M \subseteq V$ is a distance-r dominating set if $V = N^r[M]$. If there is no smaller such set, then M is a minimum distance-r dominating set of G.

Distance-r Connected Dominating Set: A set of vertices $D \subseteq V$ is a distance-r connected dominating set of G if D is a distance-r dominating set of G and the subgraph of G induced on vertices in D is connected.

Distance-r Vertex Cover: A set $C \subseteq V$ is a distance-r vertex cover of G if for every edge $e = \{u, v\}$, there exists a vertex $w \in C$ such that the distance of both u and v from w is at most r. The special case of $r = 1$ is the standard vertex cover problem. Note that in contrast to dominating set, there is no equivalence for vertex cover between the distance-r and power graph version.

Distance-r Connected Vertex Cover: Similarly, a set $\hat{C}$ is a distance-r connected vertex cover of G if it is a distance-r vertex cover of G and the induced subgraph of G on vertices of $\hat{C}$ is connected.

Edge Density, r-Shallow Minor, Expansion: Let $G = (V, E)$ be a graph; its edge density is $|E|/|V|$. A graph H is an r-shallow minor of G if H can be obtained from G by the following operations. First, we take a subgraph S of G and then partition the vertices of S into vertex disjoint connected subgraphs $S_1, \ldots, S_t$ of S, each of radius at most r and, at the end, contract each S_i ($i \in [t]$) to a single vertex to obtain H. We denote by $\nabla_r(G)$ the maximum edge density among all r-shallow minors of the graph G.

[2] For this work, it is not relevant whether we allow paths of length 0.

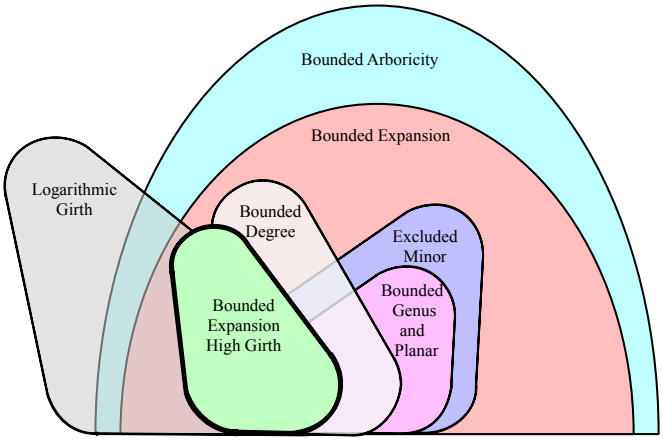

Figure 5.1.: Diagram of the relation of sparse graph classes. The graph class in the lower bound construction of Kuhn et al. [KMW16] is a subclass of logarithmic girth class of unbounded arboricity. The bounded expansion class is a subclass of bounded arboricity class. Bounded expansion is also a superclass of many common sparse graph classes: planar, bounded genus, excluded minors, and bounded degree. The class of bounded expansion with high girth intersects each of the other four classes, but neither contains nor is fully contained in any of them.

A graph class $\mathcal{C}$ is bounded expansion if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for every graph $G \in \mathcal{C}$ and integer $r \in \mathbb{N}$ we have $\nabla_r(G) \leqslant f(r)$; here f is the *expansion function*. A class of graphs $\mathcal{C}$ has constant expansion if we also have $f \in O(1)$.

Every planar, bounded genus, and excluded minor graph is a constant expansion graph. Every class of bounded degree graphs is also bounded expansion, but not of constant expansion. For more information on bounded expansion graphs, we refer the reader to the book by Nešetřil and Ossona de Mendez [ND12]. Also see Fig. 5.1 for an overview of how various graph classes relate to each other.

LOCAL and CONGEST Model of Computation: The LOCAL model of computation assumes that the problem is being solved in a distributed manner: Each vertex in the graph is also a computational node, the input graph is also the communication graph, and initially, each vertex only knows its own unique identifier and its neighbors. An algorithm proceeds in synchronous rounds on each vertex in parallel. In each round, the algorithm can run an arbitrary amount of local computation, send a message of arbitrary size to its neighboring vertices, and then receive all messages from its neighbors. Each vertex can decide locally whether it halts with an output or continues. The most common metric is the number of communication rounds.

This model was first introduced by Linial [Lin92]; later Peleg [Pel00] named it LOCAL model.

The CONGEST model is very similar to the LOCAL model, except that identifiers can be represented in $O(\log n)$ bits, and each message can only hold $O(\log n)$ bits, where n is the number of vertices in the network.

Finally, the Port numbering model removes the identifiers of vertices, and instead assigns identifiers to the "ports" of each vertex. For example, each vertex is only aware of the ports $1, \ldots, d_v$, where d_v is the degree of the vertex. This model makes many kinds of symmetry breaking impossible. We show that in sparse, high-girth graphs, this model still permits a non-trivial approximation.

Chapter 6

Bounds in the CONGEST Model

This chapter is a combination of our publications [AW21] at CIAC 2021 and TCS Special Issue on Algorithms and Complexity 2021 [AW22]. Note that the former paper has already been cited by Czygrinow et al. [CHW22].

This thesis contains in Sect. 6.2.2 a simpler analysis of the approximation, which also improves the result by factor 2.

6.1. CONGEST Algorithm

We introduce a new CONGEST algorithm, and prove that for any high-girth graph class the algorithm deterministically computes a $\mathcal{O}(r \cdot f(r))$-factor approximation of an optimum distance-r dominating set, and runs in $\mathcal{O}(r)$ rounds. More precisely, this holds for all graph classes in which all graphs have girth at least $4r + 3$. This improves the state of the art at least for all $r \in o(\log^* n)$ and $f \in \mathcal{O}(1)$.

This algorithm can be easily, but not trivially, extended to similar graph cover problems (connected distance-r dominating set, distance-r vertex cover, connected distance-r vertex cover).

6.2. Distributed Approximation Algorithm for Dominating Set

In this section we prove the following theorem.

Thm. 4 Let $\mathcal{C}$ be a graph class of bounded expansion $f(r)$ and girth at least $4r+3$. There is a CONGEST algorithm that runs in $O(r)$ rounds and provides an $O(r \cdot f(r))$-approximation of a minimum distance-r dominating set on $\mathcal{C}$, even if $r \geqslant 2$ is non-constant in n.

We prove this by providing Alg. 1, satisfying all bounds. At its core, the algorithm is simple: Each vertex computes the size of its distance-r neighborhood, i.e., the distance-r degree. This degree is propagated so that each vertex selects in its distance-r

neighborhood the vertex with the highest such degree. The output is the set of all selected vertices. We expect this to yield a good approximation because only few candidates can be selected. Alg. 1 defines this formally. The main technical contribution is Lem. 12, which concludes that Alg. 1 is correct and thus proves Thm. 4.

Algorithm 1: CONGEST computation of r-MDS, on each vertex v in parallel

1: Compute $|N^r(v)|$, e.g. using Alg. 2
2: // Phase 1: Select the vertex with the highest degree:
3: $(prio^v, sel^v) := (|N^r(v)|, v)$
4: **for** r rounds **do**
5: Send $(prio^v, sel^v)$ to all neighbors
6: Receive $(prio^u, sel^u)$ from each neighbor u
7: $(prio^v, sel^v) := \max_{u \in N^1[v]}\{(prio^u, sel^u)\}$
8: Remember all received messages that contained $(prio^v, sel^v)$
9: **end for**
10: // Phase 2: Propagate back to the selected vertex:
11: $D^v := \{sel^v\}$
12: **for** r rounds **do**
13: **for** each neighbor $u \in N^1(v)$ **do**
14: Determine which vertices sent by u in Phase 1 are in D^v
15: Send these to u, encoded as a bitset of size r
16: **end for**
17: Receive bitsets, extend D^v accordingly
18: **end for**
19: Join the dominating set if and only if $v \in D^v$, in other words:
20: **return** $v \in D^v$

We say that Alg. 1 computes a set D by returning $\top$ for all vertices in the set, and $\bot$ for all others. Naturally, messages and comparisons only consider the ID of vertices, and not the vertices themselves. This abuse of notation simplifies the algorithm and analysis. In line 7, we order tuples lexicographically: Tuples are ordered by the first element (the size of the distance-r neighborhood); ties are broken by the second element (the ID of the vertex).

In lines 15 and 17, we suggest to use "bitsets" to communicate a potentially large set of nodes efficiently. Here we exploit that vertices v and its neighbor u know which candidate-vertices were announced by vertex v in Phase 1, and in which order. This means that vertex v can encode a set of vertices simply by sending e.g. 0 to indicate absence and 1 to indicate presence in the set. Note that this obeys the message size restriction of CONGEST if and only if $r \in O(\log n)$. Observe that this is satisfied for constant r and $r \in O(\log^* n)$. For larger $r \in \omega(\log n)$, this scheme of communication is not viable. A different scheme could be used, e.g. the one in Alg. 7, where only a single bit is sent each round. We propose the bitset-approach because it seems easier to explain and understand, and because $r \in O(\log n)$ holds for all interesting cases.

Algorithm 2: CONGEST computation of $|N^r(v)|$, on each vertex v in parallel

1: $n_u := 1$ for all $u \in N^1(v)$
2: **for** r rounds **do**
3: To each vertex $u \in N^1(v)$, send $1 + \sum_{w \in N^1(v) \setminus \{u\}} n_w$
4: $n_u :=$ the number received from u, for each $u \in N^1(v)$
5: **end for**
6: **return** $\sum_{w \in N^1(v)} n_w$

Alg. 2 implements the computation of the distance-r neighborhood. The intuition is to compute the size of a *rooted* tree, for all possible roots at once. The high girth of G and line 3 mean that each vertex is counted only once (if at all).

We will now prove Alg. 1's correctness and approximation factor.

6.2.1. Correctness

First, we will show basic correctness properties. One can trivially check that all messages contain only $O(\log n)$ many bits. Specifically, the bitsets have only size $r \in o(\log^* n)$.

Lemma 6. *Alg. 2 computes the size of* $N^r(v)$ *for each vertex* v *in parallel.*

Proof. First, observe that in only $r-1$ rounds of communication, no cycle can be detected, as the girth is at least $4r+3$. This means that $N^i(v)$ is a tree for every $i \leqslant r-1$ and $v \in V$. We define the tree $T^{\neg v}_{u,i}$ as the (set of vertices in the) tree of edge-depth i, rooted at vertex u, excluding the branch to vertex v, where v is a neighbor of vertex u.

We prove by induction: At vertex v, after the i-th round[1] (where $0 \leqslant i \leqslant r-1$), n_u stores the size of the tree $T^{\neg v}_{u,i}$.

For the induction basis $i = 0$, we know $\forall u, v : n_u = 1 = \left|T^{\neg v}_{u,0}\right| = |\{u\}|$.

This leaves the induction step: At the beginning of the i-th round (for $1 \leqslant i \leqslant r-1$), we know that $n_u = \left|T^{\neg v}_{u,i-1}\right|$ by the induction hypothesis, for every u, v. Consider vertex v. By construction, its distance-i open neighborhood is the union of all edge-depth $i-1$ trees of v's neighbors, so: $N^i(v) = \bigcup_{u \in N^1(v)} T^{\neg v}_{u,i-1}$. Due to the high girth requirement, we know that all sets in this union are disjoint. Vertex v can therefore compute $\left|N^i(v)\right|$ by summing up all its n_us, and can even compute $\left|T^{\neg u}_{v,i}\right|$ for an arbitrary vertex u by subtracting the corresponding n_u again. This is exactly what happens in line 3. Then v sends $\left|T^{\neg u}_{v,i}\right|$ to each neighbor u, which stores it in the corresponding variable n_v. By symmetry, this also means that vertex v now has stored $\left|T^{\neg v}_{u,i}\right|$ in n_u, thus proving the induction step.

With the meaning of n_u established, line 6 of Alg. 2 must compute $|N^r(v)|$. □

[1] We interpret "after the zeroth round" as "before the first round"

Next, we show that Alg. 1 selects the maximum degree neighbor:

Lemma 7. *In Alg. 1, at the start of Phase 2 (line 10 et seq.), each vertex v has selected a vertex sel^v. This is the unique vertex* $\arg\max_{u \in N^r[v]}\{(|N^r(u)|, u)\}$.

Proof. By construction, only tuples of the form $(|N^r(w)|, w)$ with $w \in V$ are ever stored. The max operator is commutative and associative, so it is sufficient to prove that each vertex v considers precisely the tuples for $w \in N^r[v]$. This can be shown by straightforward induction: After round i, vertex v considers precisely the tuples for $w \in N^i[v]$. The base case is $i = 0$, and the induction step is straight-forward. □

Hence, each vertex selects the maximum neighbor. Next, we show that this is back-propagated:

Lemma 8. *If there is a vertex u that selects v ($sel^u = v$), then $v \in D^v$.*

Proof. Consider the path along which v was forwarded during the selection phase. By straight-forward induction, one can see that after i rounds of propagation, for all vertices w on the path with $d(w, u) \leqslant i$, have $v \in D^w$. The path has length at most r edges, so $v \in D^v$ after r rounds. □

And because no further vertices are added into any D^v, we get:

Corollary 9. *The selected vertices are precisely the computed set:* $D = \{sel^v \mid v \in V\}$

Together with Lem. 7, this shows that Alg. 1 computes a dominating set.

Lemma 10. *The computed set D is a distance-r dominating set.*

Proof. Assume towards contradiction that a vertex v is not dominated. Lem. 7 shows that v selected a vertex sel^v in its distance-r neighborhood. Cor. 9 shows that $sel^v \in D$, which distance-r-dominates v, which is a contradiction. □

The time complexity analysis is trivial.

Lemma 11. *Alg. 1 runs in $O(r)$ rounds.*

6.2.2. Approximation Analysis

So far we have seen the correctness of the algorithm and the running time bound. In this subsection, we prove the approximation bound of Lem. 12. Specifically, we prove that the size of D, the set of selected vertices, is within factor $1 + 2r \cdot f(r)$ of $|M|$, a minimum distance-r dominating set.

Lemma 12. *If the graph class $\mathcal{C}$ has expansion $f(r)$ and girth at least $4r + 3$, then the set of vertices D selected by Alg. 1 is small:* $|D|/|M| \in O(r \cdot f(r))$.

In the remainder of this subsection, we prove Lem. 12. Note that this means that if r is constant, then the approximation factor is constant, too.

We now analyze the behavior of Alg. 1 on a particular graph $G \in \mathcal{C}$. First, we want to get rid of an edge-case:

Lemma 13. *If* $|M| = 1$, *then* $|D| = 1$.

Proof. Let $M = \{v\}$. Then $|N^r[v]| = |V|$. Let u be the vertex with $|N^r[u]| = |V|$ with maximum ID; this may or may not be different from vertex v. By construction and Lem. 7, all vertices must select u, therefore $D = \{u\}$. □

We begin by showing that the optimal solution implies a partition into Voronoi cells [PS85], which we will use throughout the analysis. First, we define what a *covering* vertex is. Note that this can be (and often is) different from the vertex selected by the algorithm.

Definition 1

Let $c : V \to M$ be the mapping from vertices in V to corresponding dominating vertices in M, breaking ties first by distance and then by ID:

$$c(v) \coloneqq \operatorname*{arg\,min}_{u \in N^r[v] \cap M} \{(d(u, v), u)\} \tag{6.1}$$

We order tuples lexicographically, again. The equivalent term $\operatorname{arg\,min}_{u \in M}\{(d(u, v), u)\}$ provides shorter notation: By construction, each vertex v has a vertex in M in its r-neighborhood, so arg min will select from $N^r[v]$ anyway. Next, we partition V into Voronoi cells $H_m \coloneqq \{v \in V | c(v) = m\}$ for each $m \in M$.

Corollary 14. *Each* H_m *is connected and has radius at most* r.

Proof. As vertex m dominates all vertices in H_m, we know that there is a path of length at most r from vertex m to each vertex in H_m. □

We use the high-girth property to show that the Voronoi cells behave nicely:

Lemma 15. *The subgraph induced by* H_m *in* G *is a tree.*

Proof. Assume towards contradiction that there is a cycle C' in H_m. We construct a cycle that has length at most $2r + 1$, a contradiction.

Specifically, construct a BFS-tree of H_m rooted in m. Then, the cycle C' must contain an edge e between $u, v \in H_m$. Consider the cycle that consists of the path from u to v along the BFS-tree and the edge $\{u, v\}$. This cycle must have length at most $r + r + 1$, because the BFS-tree has depth at most r. This contradicts G having a girth of at least $4r + 3$. □

Lemma 16. *For any two Voronoi cells $H_m \neq H_n$, there is at most one edge between them.*

Proof. Let $\{u, v\} \in E$ and $\{s, t\} \in E$ be two different edges between H_m and H_n. W.l.o.g. assume $c(u) = c(s) = m$ and $c(v) = c(t) = n$, and assume $v \neq t$ (but $u = s$ is possible).

By Cor. 14, we know that the subgraphs induced by H_m and H_n are each connected, so there must be a path p_m entirely in H_m between vertices u and s, possibly of length 0. Likewise, a path p_n must exist entirely in H_n between vertices v and t. The union of the paths and the assumed edges forms a cycle $C_{u,v,s,t}$, as no vertex can be repeated. We will now prove that $C_{u,v,s,t}$ is too small.

The paths p_m and p_n have length at most $2r$ each. Therefore, we have found the cycle $C_{u,v,s,t}$ to have length at most $4r + 2$, in contradiction to the minimum girth $4r + 3$. □

Let $G' = (V', E')$ be the result of contracting H_m to a single vertex, for each $m \in M$.

Lemma 17. *The edge set E' is small:* $|E'| \leqslant f(r) \cdot |M|$

Proof. Using Cor. 14, we can apply the definition of the function $f(r)$:

$$|E'| = \frac{|E'|}{|V'|} \cdot |V'| \leqslant f(r) \cdot |M|$$ □

Next, we construct a set of candidates, based on each Voronoi cell.

Definition 2

For each Voronoi cell H_m, we define the set of vertices T_m as the union of all shortest paths $P_{m,u}$ between vertex m and each vertex u on the Voronoi boundary, and we define T as the union of all T_m:

$$T_m := \bigcup_{\substack{\{u,v\} \in E \\ c(u)=m, c(v)\neq m}} P_{m,u} \qquad\qquad T := \bigcup_{m \in M} T_m$$

This is well-defined due to Lem. 15. Observe that T_m is not necessarily equal to H_m: All leaves in T_m have edges in G that lead outside the Voronoi cell. A leaf-vertex in H_m that has no such edges will not be in T_m. This reduces the number of candidates sufficiently:

Lemma 18. *The set T is small:* $|T| \leqslant (1 + 2r \cdot f(r))\,|M|$

Proof. Consider an arbitrary but fixed $\{v, u\} \in E$ with $c(v) \neq c(u)$. Each path $P_{c(v),v}$ has at most r vertices not in M, because it is a shortest path, and by construction all vertices are dominated by $c(v)$. Each edge in E' corresponds to at most two such paths, due to Lem. 16. With Lem. 17, this bounds the number of paths to at most $2f(r)\,|M|$.

Therefore, T contains at most $2r \cdot f(r)\,|M| + |M|$ vertices. □

Now we can prove in two steps that the algorithm only selects vertices from the candidate set T:

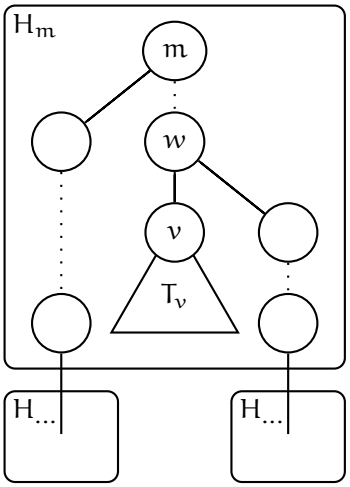

Figure 6.1.: Typical vertex layout in proof of Lem. 20. The identity of vertex u does not matter; hence, it is not shown.

Lemma 19. *If $|M| > 1$, then there is always a vertex just out of reach (i.e. in distance $r + 1$):*

$$\forall v \in V \; \exists u \in V : d(u, v) = r + 1 \tag{6.2}$$

Proof. Assume towards contradiction that there is a vertex v for which no such vertex u exists. Then, there is also no vertex u' with $d(u', v) > r + 1$, because one could pick a shortest path and construct such a u. Therefore, $D' = \{v\}$ would be a dominating set, a contradiction to $|M| > 1$. □

Lemma 20. *Let v be a vertex selected by u. Then $v \in \mathcal{T}_{c(v)}$.*

Proof. Assume towards contradiction that $v \notin \mathcal{T}_{c(v)}$. For brevity, let $m := c(v)$. Observe that $m \neq v$, because $m \in \mathcal{T}_m$. Let w be the next vertex on a shortest path from v towards m; possibly m itself. We now analyze the properties of vertex w and conclude that vertex u should not have selected v. Refer to Fig. 6.1 for an overview.

By Lem. 15, the subgraph induced by H_m is a tree. If we root this H_m-tree at vertex m, we can denote the subtree rooted at v as T_v. This subtree has depth at most $r - 1$ (because $v \neq m$), so w covers the entire subtree: $T_v \subseteq N^r(w)$. All vertices $x \in V \setminus T_v$ are closer to w than to v, as all paths from x to v must go through w. So, the neighborhood of v is included in the neighborhood of w: $N^r[v] \subseteq N^r[w]$.

Now we can use Lem. 19: There must be a vertex t that has distance $r + 1$ to vertex v, so $t \notin N^r[v]$. This means that $t \notin N^r[v]$ and $t \in N^r[w]$. Therefore, the degree of w is strictly larger: $|N^r[w]| > |N^r[v]|$. Thus, vertex u would prefer selecting w over v, and also was able to do so (because $u \in N^r[v] \subseteq N^r[w]$).

This leads to a contradiction: Vertex u selected v, although vertex w should be preferred by the algorithm. □

We can combine the previous lemmas to achieve an exact bound:

Corollary 21. *The set* D *is small:* $|D| \leqslant (1 + 2r \cdot f(r))\,|M|$

Proof. Follows from Lems. 18 and 20. □

This proves Lem. 12, and thus Thm. 4. More specifically, we have proved the upper bound $(1 + 2r \cdot f(r)) \cdot |M|$ on $|D|$.

6.2.3. Tightness of the Analysis

We have seen that the algorithm is an $O(r \cdot f(r))$ approximation. Is it possible that the algorithm actually performs significantly better than what the analysis guarantees? This subsection proves that there are graphs for which the algorithm yields an $\Omega(r \cdot f(r))$ approximation, meaning that the above analysis of the algorithm is asymptotically tight.

We will focus on the tightness of Cor. 21, by constructing a worst-case input graph. See also Lem. 12.

Lemma 22. *For arbitrary but fixed values* $r \geqslant 2, \nu \geqslant 2$*, there is a graph* $G_{r,\nu}$ *such that:*

- *the singleton graph class* $\mathcal{C}_{r,\nu} := \{G_{r,\nu}\}$ *has expansion* $f(r) = \nu$ *and girth at least* $4r + 3$*, and*
- *executing Alg. 1 on* $G_{r,\nu}$ *computes a* $(r \cdot f(r))$*-approximation (or worse) of the minimum distance-*r *dominating set of* $G_{r,\nu}$*.*

Proof. The remainder of this subsection constructs $G_{r,\nu}$ and proves its properties. □

The construction is a modified version of the subdivided biclique. Let X and Y be two disjoint sets of vertices, each of size 2ν. For each pair $(x, y) \in X \times Y$, create a path $P_{x,y}$ starting at vertex x and ending in vertex y, with $2r$ new vertices, such that $d(x, y) = 2r + 1$. This means that no vertex can simultaneously cover x and y, i.e., no vertex is within distance r of both x and y. In path $P_{x,y}$, let $b_{x,y}$ be the second vertex (the one after x). Create a set $B_{x,y}$ of $k = 2r\nu$ new vertices, and connect each vertex in $B_{x,y}$ by a single edge to the vertex b. Let V be the union of all the sets X, Y, $P_{x,y}$, $B_{x,y}$; and let E be the set of edges as described. Then $G_{r,\nu} = (V, E)$ is the constructed graph.

First, we prove that the graph class satisfies all requirements.

Lemma 23. *The graph class* $\mathcal{C}_{r,\nu} = \{G_{r,\nu}\}$ *has expansion* $f(r) = \nu$ *and girth at least* $4r + 3$*.*

Proof. The girth of $G_{r,\nu}$ is at least $4 \cdot (2r + 1) > 4r + 3$, as a cycle needs to pass through at least two vertices from X and two vertices from Y.

To prove the low expansion of $\mathcal{C}_{r,\nu}$, it suffices to show $\nabla_r(G) = \nu$. This can be shown by contracting as much as possible around all vertices in $X \cup Y$, which results in the biclique $K_{2\nu,2\nu}$, with 4ν vertices and $4\nu^2$ edges. Therefore, the constructed graph has $\nabla_r(G) \geqslant \nu$. This is the contraction choice that maximizes the average degree, as it eliminates all degree-2 vertices. Therefore $\nabla_r(G) = \nu$, and thus $f(r) = \nu$. □

Next, we show that the algorithm computes a comparatively large dominating set:

Lemma 24. *Alg. 1 computes a* $(r \cdot f(r))$*-approximation (or worse) of minimum distance-*r *dominating set on* $G_{r,\nu}$*.*

Proof. By construction, $X \cup Y$ is a dominating set, meaning $|M| \leqslant 4f(r)$. Therefore, it suffices to show that $|D| \geqslant 4r \cdot f(r)^2$.

We do so by simulating the algorithm on $G_{r,\nu}$. We only need to consider the vertices selected by vertices on the paths do. Specifically, pick a specific path $P_{x,y}$ between $x \in X$ and $y \in Y$. Vertices v_x closer to x than to y cover the attached vertices $B_{x,y}$, so $|N^r(v_x)| \geqslant 2r + k = 2r + 2r \cdot f(r)$. The vertices closer to vertex x cover more of the other paths ending in x, each step increases $|N^r(v_x)|$ by at least $2f(r) - 1$, and loses at most 1 vertex out of sight in the y direction. Note that we ignore the vertices in $B_{x,y'}$ with $y' \neq y$, which would only make this argument stronger. The important property is that $|N^r(v_x)|$ strictly increases towards x, among vertices v_x with $d(v_x, x) < d(v_x, y)$.

Each vertex v_y closer to y than to x does not cover the attached vertices $B_{x,y}$ close to vertex x, as distance r from them would imply distance r to x. We can compute $|N^r(v_y)| \leqslant r + N^r(v_r) + 1 - 1 = r + r \cdot 2f(r) < 2r + 2r \cdot f(r) \leqslant |N^r(v_x)|$, so vertex v_y will choose some vertex v_x. As we already established, $|N^r(v_x)|$ increases with decreasing distance to x. Therefore, each v_y will select the vertex closest to x, meaning at least half of each path will be selected, specifically the one on the v_l side.

This means the algorithm selects at least r vertices per path, and there is one such path for each $X \times Y$ combination. Hence $|D| \geqslant r \cdot 4 \cdot f(r)^2$. Recall that $|M| \leqslant 4f(r)$, so the algorithm achieves an approximation factor of at least $r \cdot f(r)$ for the constructed graph. Compared with the upper bound of $1 + (2r \cdot f(r))$ this is asymptotically tight. □

This concludes the proof of Lem. 22 (tightness of approximation). Observe that this result does not imply anything about the problem's intrinsic hardness, but rather that in the worst case, the presented algorithm may use up the approximation slack. In other words, the upper bound in Lem. 12 is asymptotically tight.

6.3. Unconditional Lower Bound

In this section, we prove that computing a significantly better approximation of the problem is hard. Intuitively speaking, this is because symmetry cannot be broken in $o(\log^* n)$ rounds, and without that, it is hard to construct any non-trivial distance-r dominating set.

We show the hardness by a reduction from the "large" independent set problem to the distance-r dominating set problem, on the graph class of cycles. Intuitively speaking we find a distance-r dominating set D on cycle C; two consecutive vertices of D on C are of distance at most $2r + 1$ from each other, and hence, these vertices help us to break the symmetry, and as $r \in o(\log^* n)$ it yields an independent set of size $O(n)$ in $o(\log^* n)$ rounds.

Thm. 5 Assume an arbitrary but fixed $\delta > 0$ and $r > 1$, with $r \in o(\log^* n)$. Then, there is no deterministic LOCAL algorithm that finds in $O(r)$ rounds a $(2r+1-\delta)$-approximation of distance-r dominating set for all $G \in \mathcal{C}$, where $\mathcal{C}$ is the class of cycles of length $\gg 4r+3$.

As we will see later, the trivial distance-r dominating set V (i.e., the set of all vertices), is a $(2r+1)$-approximation in the case of cycles.

This has been proved implicitly in the work of [LPW13]. However, we find it simpler to provide a new proof tailored for our setting, but only for n being a multiple of $2r+1$. In essence, we show a reduction from the "large" independent set problem to the distance-r dominating set problem, on the graph class of cycles. Intuitively speaking, any algorithm that does significantly better than the trivial dominating set *anywhere* on the cycle leads to a linear sized independent set; and the bound is constructed such that the algorithm needs to do better than trivial somewhere indeed.

The idea is simple: Find a distance-r dominating set D on cycle C; we know two consecutive vertices of D on C are of distance at most $2r+1$ from each other, and hence, these vertices help us to break the symmetry, and as $r \in o(\log^* n)$ it yields an independent set of size $O(n)$ in $o(\log^* n)$ rounds. In the following, we will formalize this idea into an proper argument.

Assume towards contradiction that $\mathcal{ALG}$ is such a deterministic distributed algorithm, which finds a distance-r dominating set in $G \in \mathcal{C}$ of size at most $(2r+1-\delta)\,|M|$, where M is a minimum distance-r dominating set and r and δ as in Thm. 5.

We show that $\mathcal{ALG}$ can be used to construct an algorithm violating known lower bounds on "large" independent set [CHW08; LPW13]:

Lemma 25 (Lemma 4 of [CHW08])**.** *There is no deterministic distributed algorithm that finds an independent set of size $\Omega(n/\log^* n)$ in a cycle on n vertices in $o(\log^* n)$ rounds.*

We present the reduction algorithm in Alg. 3.

Algorithm 3: CONGEST computation of an IS on a cycle $G \in \mathcal{C}$, for each v in parallel

1: Compute a distance-r dominating set D by simulating $\mathcal{ALG}$.
2: Determine the connected components $V \setminus D$.
3: **for** each component C_i **do**
4: Determine the two adjacent vertices to C_i, i.e. $u, v \in N(C_i)$.
5: Let u be the vertex with the lower ID, name it representor of C_i.
6: All vertices of odd distance to u in C_i join I.
7: **end for**
8: **return** I

We begin by showing basic correctness:

Lemma 26. *Alg. 3 runs in* $o(\log^* n)$ *rounds.*

Proof. By assumption, $\mathcal{ALG}$ executes in $O(r)$ rounds. On the other hand, observe that each vertex in D only covers up to a distance of r. Because D is a dominating set, all components must have length at most $2r$. Hence, discovering the adjacent vertex of lowest ID can be done in $O(r)$ as well as propagating the distance information. By construction $r \in o(\log^* n)$, so Alg. 3 takes $o(\log^* n)$ rounds. □

Lemma 27. *Alg. 3 computes set* I*, which is an independent set.*

Proof. For two distinct vertices $u, v \in I$, if they belong to different components, then there is no edge between them; otherwise, if they are in the same component, their distance is at least 2, as they are distinct vertices of odd distance from their representor. □

Now we can show that this yields a large independent set:

Lemma 28. *The dominating set is not too large:* $|D| \leqslant (1-\delta')n$ *for some* $\delta' > 0$*.*

Proof. By assumption, we know $|D| \leqslant (2r+1-\delta)\,|M|$, where M is the minimum distance-r dominating set. Construct M' by picking every $2r+1$-th vertex so that $|M'| = n/(2r+1)$. Note that M' is a distance-r dominating set, so we have $|M| \leqslant |M'|$. Together we get $|D| \leqslant (2r+1-\delta)n/(2r+1) = (1-\delta')n$, for $\delta' \coloneqq 1/(2r+1) > 0$. □

Lemma 29. *The set* I *is large:* $|I| \in \Omega(n/\log^* n)$

Proof. Many vertices must be part of some component: $|V \setminus D| \geqslant \delta' n$ for some $\delta' > 0$ by Lem. 28. At least half of those vertices are taken into I, thus $|I| \geqslant \delta' n/2 \in \Omega(n/\log^* n)$. □

Proof of Thm. 5. Lems. 25 and 29 imply that algorithm $\mathcal{ALG}$ cannot exist. □

Note that this does not preclude randomized algorithms. This is because randomized algorithms can indeed achieve a better approximation quality, at least on cycles, by randomly joining the dominating set with sufficiently small probability if necessary, for several rounds, and finally all uncovered vertices join.

6.4. Vertex Cover, Connected Dominating Set, and Connected Vertex Cover

In this section, we extend Alg. 1 to solve three other covering problems, namely Distance-r Vertex Cover, Distance-r Connected Vertex Cover, and Distance-r Connected Dominating Set; the precise definitions can be found in Sect. 5.4. Again, we assume the input graph is $G = (V, E)$ so that we can directly refer to its edge and vertex set.

The rough idea for each of these extensions is simple: For the connected sets, we take the original set and connect two vertices that are within a small distance via short paths. It is possible to show that due to the structure of bounded expansion graphs, this

approach does not add much extra overhead and it is a constant factor approximation to the problem. For vertex cover, observe that every vertex cover is already a dominating set, to go the other way around we define boundary vertices of each Voronoi cell of the dominating set, then include them into the solution to vertex cover. It is possible to show that this approach provides a constant factor approximation for the problem (distance-r variation).

Lemma 30. *There is a CONGEST algorithm that computes an $O((r \cdot f(r))^2)$ approximation of Distance-r Connected Dominating Set in graphs of bounded expansion with high girth in $O(r)$ rounds.*

Proof. We prove this by constructing Alg. 4 as a simple extension of Alg. 1, or any other appropriate CONGEST distance-r dominating set algorithm.

Algorithm 4: CONGEST computation of connected r-MDS, on each vertex v in parallel

1: Compute a Distance-r Dominating Set D of the graph
2: Determine the closest dominating vertex sel^v
3: **if** any neighbor u has a $sel^u \neq sel^v$ **then**
4: Call vertex v a *border vertex*
5: Determine the path P^v from v to sel^v
6: **end if**
7: **return** $\hat{D}$ as the union of D, all border vertices, and their paths P^v

Alg. 4 is a CONGEST algorithm, as the distance-r dominating set algorithm is CONGEST, and all other messages only contain a constant amount of identifiers.

Alg. 4 takes $O(r)$ rounds, because the distance-r dominating set algorithm does so, too, and all other steps also only take $O(r)$ rounds.

$\hat{D}$ is a dominating set because $D \subseteq \hat{D}$ is a dominating set.

Define Voronoi cells H_d for each $d \in D$. Note that Cor. 14 and Lem. 15 apply analogously.

We show that $\hat{D}$ is connected by construction, if G is connected: Vertices v within a Voronoi cell $H_d \cap \hat{D}$ are connected by construction, as they are all connected to $d \in \hat{D}$. Furthermore, for every path P_G in the input graph G, one can construct a corresponding walk W_H in the Voronoi graph by mapping each vertex to its Voronoi cell (i.e. sel^v). Thus, the Voronoi cells are connected.

Finally, we show the approximation quality: Consider the minimal Distance-r Connected Dominating Set $\hat{M}$. One can easily see that the minimum distance-r dominating set M is not larger: $|M| \leqslant |\hat{M}|$. An argument similar to Lem. 17 shows that the number of border vertices is bounded in $|D|$; and by construction of D the Voronoi cells have radius at most r (and therefore so do the paths). By Thm. 4, we can now deduce:

$$|\hat{D}| \in O(r \cdot f(r) \cdot |D|) \subseteq O((r \cdot f(r))^2 \cdot |M|) \subseteq O((r \cdot f(r))^2 \cdot |\hat{M}|) \qquad \square$$

For constant r, the terms simplify to the following:

Corollary 31. *For constant r, there is a CONGEST algorithm that computes a constant factor approximation of Distance-r Connected Dominating Set for constant r in graphs of bounded expansion with high girth in constant number of rounds.*

Likewise, we can solve the related vertex cover problem. Intuitively speaking, we can define Voronoi cells according to the computed dominating set, determine borders between cells, and include all borders into the vertex cover. More formally:

Lemma 32. *There is a CONGEST algorithm that computes an $O(r \cdot f(r) \cdot f(r))$ approximation of Distance-r Vertex Cover in graphs of bounded expansion with high girth in $O(r)$ rounds.*

Proof. We prove this by constructing Alg. 5 as a simple extension of Alg. 1, or any other appropriate CONGEST distance-r dominating set algorithm.

Algorithm 5: CONGEST computation of distance-r vertex cover, on each vertex v in parallel

1: Compute a Distance-r Dominating Set D of the graph
2: Determine the closest dominating vertex sel^v
3: If any neighbor u of vertex v has a $sel^u \neq sel^v$, call v a border
4: **return** C as the union of D and all border vertices

Alg. 5 is a CONGEST algorithm as the Distance-r Dominating Set algorithm is CONGEST, and all other messages only contain a constant amount of identifiers. Alg. 5 takes $O(r)$ rounds, because the Distance-r Dominating Set algorithm does so, too, and all other steps also only take $O(r)$ rounds.

Define Voronoi cells according to sel^v for $v \in V$. The set C is a distance-r vertex cover by simple case distinction: All edges $e = u, v$ that are fully inside a Voronoi cell, i.e., there is a vertex $w \in C$ with $w = sel^u = sel^v$, is covered by vertex w. All edges $e = u, v$ with $sel^u \neq sel^v$ are covered by vertices u and v, as both vertices were detected as borders.

Finally, we show the approximation quality: Consider the minimal Distance-r Vertex Cover M^{VC}. One can easily see that the minimum distance-r dominating set M is smaller: $|M| \leqslant \left|M^{VC}\right|$. An argument similar to Lem. 17 shows that the number of border vertices is bounded in $|D|$. By Thm. 4, we can now deduce:

$$|C| \in O(f(r) \cdot |D|) \subseteq O(r \cdot f(r) \cdot f(r) \cdot |M|) \subseteq O(r \cdot f(r) \cdot f(r) \cdot \left|M^{VC}\right|) \qquad \square$$

Again, for constant r, the terms simplify to the following:

Corollary 33. *For constant r, there is a CONGEST algorithm that computes a constant factor approximation of the minimum Distance-r Vertex Cover in a constant number of rounds.*

The arguments in Lem. 30 apply analogously to Cor. 33:

Corollary 34. *For constant r, there is a CONGEST algorithm that computes a constant factor approximation of the minimum Distance-r Connected Vertex Cover in a constant number of rounds.*

This proves that Alg. 1 can be extended to several related graph cover problems.

Chapter 7

Extension to the Port Numbering Model

This unpublished contribution extends the algorithm's versatility by removing the dependency on unique vertex identifiers. Although this is in general a desirable property, it also highlights a weakness regarding symmetry in graphs.

7.1. Algorithm and Proof

We show that the algorithm can be easily, but not trivially, modified to run in the port numbering model (which provides less abilities to the algorithm, and is thus considered a weaker model than Congest).

This approach also mostly works in the port numbering model, if we just replace vertex IDs by port numbers, and replacing $N^1(v)$ by $P(v) = \{1, 2, \ldots, d_v\}$, be the set of ports of a vertex. However, the edge case $|M| = 1$ causes several complications, which is why we introduced the Congest algorithm first.

Algorithm 6: Port numbering computation of $|N^r(v)|$, on each vertex v in parallel

1: $n_u := 1$ for all $u \in P(v)$
2: **for** r rounds **do**
3: To each vertex $u \in P(v)$, send $1 + \sum_{w \in P(v) \setminus \{u\}} n_w$
4: $n_u :=$ the number received from u, for each $u \in P(v)$
5: **end for**
6: **return** $\sum_{w \in P(v)} n_w$

Algorithm 7: Port numbering computation of r-MDS, on each vertex v in parallel

1: // Phase 0: Detect and handle special cases:
2: **if** degree is 0 **then**
3: **return** $\top$
4: **end if**
5: Test for $|M| = 1$ case and terminate accordingly, e.g. using Alg. 8
6: Compute $|N^r(v)|$, e.g. using Alg. 6
7: // Phase 1: Select the vertex with the highest degree:
8: $(prio^v, port^v) := (|N^r(v)|, \perp)$ // round 0
9: **for** r rounds, in round $i = 1, 2, 3, \ldots, r$ **do**
10: Send $(prio^v)$ to all neighbors
11: Receive $(prio^u)$ from each neighbor $u \in P(v)$
12: Set $(prio^v, port^v)$ to $(prio^u, u)$, if this increases $prio^v$
13: Remember all received messages that contained $prio^v$
14: **end for**
15: // Phase 2: Propagate back to the selected vertex:
16: Set $I := \{j\}$ to the set of marked rounds, initially the singleton j, where j is the last round in which we last updated $prio^v$
17: **for** r rounds, with $i = r, r-1, \ldots, 3, 2, 1$ **do**
18: If round i is marked ($i \in I$), recall which port triggered our update of $prio^v$ earlier, and notify it.
19: // Note that this can be done using empty messages.
20: If vertex v is notified in round i by any neighbor, recall which $prio^v$ was advertised during Phase 1,
and mark round $j < i$ in which this $prio^v$ was discovered.
21: **end for**
22: Join the dominating set if and only if $0 \in I$, in other words:
23: **return** Round 0 is marked

Compared to Algs. 1 and 2, the algorithms barely changed at all. In fact, only Alg. 8 and the explicit handling of the case $n = 1$ is really new. In Alg. 7, sel^v which used to track vertex IDs has been replaced by port numbers, and a more elaborate yet functionally identical scheme is used to notify the vertex that was selected. In particular, observe that the tie-breaking by vertex ID in Alg. 1 no longer happens, and that the analysis does not use it in any way. Instead, the analysis only compares the values of $|N^r(v)|$ in Lem. 20. For the Voronoi cells in Definition 1, we can supply any arbitrary tie-breaking, e.g. by inventing IDs during the analysis.

We will only consider connected graphs. Due to line 3, we only need to consider graphs of size $n \geqslant 2$.

We begin with simple observations about the running time of the algorithm and the arboricity of the graph:

Algorithm 8: Port numbering detection and (if $|M| = 1$) selection of an r-MDS

For each neighbor $u \in N(v)$, let $d^v_{u,0} := \infty$
for r rounds, in round i (starting at $i = 1$) **do**
  if v has degree 1 **then**
    To the neighbor u, send 1
  else
    To each neighbor u, send $1 + \max_{w \in N(v), w \neq u}\{d^v_{w,i-1}\}$ // May be ∞
  end if
  Receive value from each neighbor $u \in N(v)$, save as $d^v_{u,i}$
end for
Compute $l_v := \max_{u \in N(v)} d^v_{u,r}$
Let $l_{min} := l$
for r rounds **do**
  Propagate l_v and l_{min}, and update l_{min} to the minimum of all seen values
end for
if $l_{min} > r$ **then**
  return $|M| \neq 1$ (i.e., no early-termination)
else if $l_{min} = l_v$ **then**
  return $|M| = 1, v \in D$ (i.e., early-termination with v in the dom. set)
else
  return $|M| = 1, v \notin D$ (i.e., early-termination with v *not* in the dom. set)
end if

Corollary 35. *Alg. 8, and therefore also Alg. 7, runs in time* $O(r)$.

Lemma 36. *If* $|M| = 1$, *then* G *is a tree.*

Proof. Follows immediately from the fact that the radius must be at least r, and the bounded girth $g \geqslant 4r + 3 > 2r + 1$. □

Next, observe that each $d^v_{u,i}$ provides some sort of upper bound on distance, which we need to define first:

Definition 3

Consider a vertex $v \in V$. For each vertex $u \in N(v)$, define the directed distance $\vec{d}(v, u)$ as the length of the longest path starting with $(v, u, \ldots)$.

Observe that $\vec{d}(u, v)$ is well-defined, as a path cannot visit a vertex more than once. Therefore, there are only finitely many paths in G. Now we can express the upper bound provided by $d^v_{u,i}$:

Lemma 37. *Consider a vertex $v \in V$. For each $u \in N(v)$ and round $i \in 0, \ldots, r$, it holds that:*

1. $d^v_{u,i}$ *is an upper bound:* $\vec{d}(v,u) \leqslant d^v_{u,i}$.
2. $d^v_{u,i}$ *can be tight: If* $d^v_{u,i} < \infty$, *then* $\vec{d}(v,u) = d^v_{u,i} \leqslant i$.
3. $d^v_{u,i}$ *can be an implicit lower bound: If* $d^v_{u,i} = \infty$, *then* $\vec{d}(v,u) > i$.

Proof. By induction on i. First, we show the base case with $i = 0$. Observe that $\vec{d}(v,u) < \infty = d^v_{u,0}$ and $\vec{d}(v,u) > 0$ hold trivially.

For the induction step with $i \geqslant 1$, consider a path $P_{v,u}$ of maximum length starting with $(v,u,\ldots)$. By construction, it has length $\vec{d}(v,u)$. We distinguish two cases, like in the algorithm:

- If vertex u has degree 1, then vertex u sends the value 1, and v receives $d^v_{u,i} = 1$. Observe that $P_{v,u} = (v,u)$, $\vec{d}(v,u) = 1$, and therefore all points are satisfied.
- If vertex u has degree at least 2, then $P_{v,u} = (v,u,w,\ldots)$ for some vertex $w \neq u, v$. The value sent by u must be equal to $d^u_{w,i-1} + 1$, therefore $d^v_{u,i} = d^u_{w,i-1} + 1$. By construction of path $P_{v,u}$ and vertex w, we get $\vec{d}(v,u) = \vec{d}(u,w) + 1$. This gives us the first point:

  $$\vec{d}(v,u) = \vec{d}(u,w) + 1 \leqslant d^u_{w,i-1} + 1 = d^v_{u,i} \tag{7.1}$$

  For the second and third point, we need to make another case distinction:

  - If $d^u_{w,i-1} < \infty$, then $d^v_{u,i} = d^u_{w,i-1} + 1 < \infty$, so point three is trivial, and point two holds: $\vec{d}(v,u) = \vec{d}(u,w) + 1 = d^u_{w,i-1} + 1 = d^v_{u,i}$, as well as $d^v_{u,i} = d^u_{w,i-1} + 1 \leqslant i - 1 + 1 = i$.
  - If $d^u_{w,i-1} = \infty$, then $d^v_{u,i} = d^u_{w,i-1} + 1 = \infty$, so point two is trivial, and point three holds: $\vec{d}(v,u) = \vec{d}(u,w) + 1 > i - 1 + 1 = i$. □

With this tool, we can show that the algorithm correctly decides whether $|M| = 1$:

Lemma 38. *If* $|M| \neq 1$, *then Alg. 8 returns "*$|M| \neq 1$*".*

Proof. It suffices to show that all vertices compute $l = \infty$, because then all vertices compute $l_{min} = \infty > r$.

Let v be any vertex. Because $|M| > 1$, the set $\{v\}$ cannot distance-r dominate the entire graph G. Let w be some vertex that has distance larger than r from vertex v. Take a shortest path P from v to w, and define vertex u as the vertex adjacent to vertex v. Then, $\vec{d}(v,u) > r$, because the longest path cannot be shorter than P. By Lem. 37 (in particular points 1 and 2), we can deduce that $d^v_{u,r} = \infty$, and thus $l_v = \infty$. □

Lemma 39. *If $|M| = 1$, then Alg. 8 returns "$|M| = 1$".*

Proof. It suffices to show that some vertex computes $l \leqslant r$, because then all vertices compute $l_{min} \leqslant r$.

Let $M = \{v\}$ be any optimal solution. Observe that for any neighbor $u \in N(v)$, it holds that $\vec{d}(v,u) \leqslant r$. Observe the converse of point 3 of Lem. 37: If $\vec{d}(v,u) \leqslant r$ (which we have here), then $d^v_{u,r} \neq \infty$. From this we can deduce that $d^v_{u,i} = \vec{d}(v,u) \leqslant r$. As this holds for any u, we can deduce that $l \leqslant r$ for vertex v. □

Finally, we want to show that the resulting dominating set D is actually small. We first define the radius of a vertex:

Definition 4

Let v be any vertex. Then define $r(v)$ as the radius of the graph around v:

$$r(v) \coloneqq \max_{u \in V} d(v,u) \tag{7.2}$$

Observe that on trees, the radius is closely related to the directed distance:

Corollary 40. *The radius of a vertex is the maximum of its directed distances:*

$$r(v) = \max_{u \in N(v)} \vec{d}(v,u) \tag{7.3}$$

This relation helps us in the interpretation of l:

Lemma 41. *Each vertex v holds in variable l effectively its radius $r(v)$:*

$$l_v = \begin{cases} r(v) & \text{if } r(v) \leqslant r \\ \infty & \text{otherwise} \end{cases} \tag{7.4}$$

Proof. If there is any neighbor $u \in N^1(v)$ with $\vec{d}(v,u) > r$, then by Lem. 37 we get $d^v_{u,r} = \infty$ and thus $l = \infty$. Otherwise, for all $u \in N^1(v)$ we get $d^v_{u,r} = \vec{d}(v,u)$, and $l_v = r(v)$ by Cor. 40. □

Using the radius, we can define the center of a graph:

Definition 5

For any non-empty graph G, define the center $C(G)$ as the set of vertices of minimum radius:

$$C(G) \coloneqq \left\{ v \in V \;\middle|\; r(v) = \min_{u \in V} r(u) \right\} \tag{7.5}$$

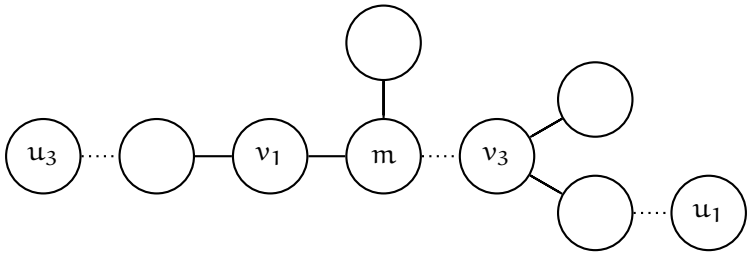

Figure 7.1.: Typical vertex layout in proof of Lem. 42. The identity of vertex v_2 does not matter; hence, it is not shown. The dotted edges indicate that the vertices are connected by some path of unspecified length (at least 1).

This center of a graph is small:

Lemma 42. *The center of a non-empty tree always exists and is small:* $1 \leqslant |C(G)| \leqslant 2$

Proof. For any non-empty graph, $\min_{u \in V} r(u)$ is well-defined, and attained by at least one vertex v. Therefore, it suffices to show that $|C(G)| \leqslant 2$.

Proof by contradiction of the opposite: Assume there are three distinct vertices $v_1, v_2, v_3 \in C(G)$. Then, because G is a tree, not all are neighbors of each other. Without loss of generality, assume that v_1 and v_3 are not neighbors. As such, we can pick any vertex on the path from v_1 to v_3 and denote it m. See Fig. 7.1 for a reference figure.

Let u be any vertex. If the path from u to m does not contain any of v_1, v_3, then $d(u, m)$ is smaller than $r(v_1)$ by construction of m. If the path from u to m contains v_1, observe that $d(u, m) < d(u, m) + d(m, v_3) = d(u, v_3) \leqslant r(v_3)$. Analogously if the path from u to m contains v_3. Therefore, $r(m) < r(v_1) = r(v_3)$, which contradicts $v_1, v_3 \in C(G)$. □

Now we can observe that Alg. 8 computes exactly the center of a graph, and thus the computed set D is small:

Lemma 43. *If* $|M| = 1$*, then Alg. 8 returns "*$v \in D$*" in at most two vertices.*

Proof. We have already seen that $|M| = 1$ implies that G is a tree, so by Lem. 42 and Definition 5 it suffices to show that exactly the vertices in $C(G)$ return "$v \in D$".

Because $|M| = 1$, we can deduce that all vertices compute the same value $l_{min} = \min_{u \in V} l_u$.

Let c be a vertex in $C(G)$. By Lem. 41 we can deduce $l_{min} \leqslant l_c = r(c) \leqslant r$ must be finite. (We have already shown something slightly weaker in Lem. 39.) Furthermore, vertices $v \notin C(G)$ must compute $l_v \geqslant r(v) > r(c)$. Finally, because vertices c in $C(G)$ all have identical $r(c)$, we get $l_{min} = r(c)$, and thus exactly the vertices in $C(G)$ return "$v \in D$". □

Corollary 44. *If* $|M| = 1$*, then Alg. 7 computes a 2-approximation.*

Corollary 45. *Alg. 7 provides an* $\Omega(r \cdot f(r))$*-approximation of minimum distance-*r *dominating set on any graph class* $\mathcal{C}$ *of bounded expansion* $f(r)$ *and girth at least* $4r + 3$*.*

In conclusion, we see that although the approach can be rather easily made to work in the port numbering model, the $|M| = 1$ case causes a lot of headaches. In practice, an algorithm might execute this "branch" of the algorithm in parallel with the alternative, thus folding the runtime in half. However, this does not affect asymptotic runtime behavior, so we chose to keep the test separate, for simplicity.

In the CONGEST model, we do not need to explicitly detect this case, as the vertex with maximum neighborhood and maximum ID is chosen. However, the port numbering model does not offer any IDs, and this is the reason why an alternative tie-breaking mechanism has to be used: graph centrality, in this case.

Chapter 8

Conclusion

We have generalized a well-known problem of distributed systems, and analyzed the resulting distance-r dominating set problem. Our findings include a simple algorithm, which yields a constant approximation in constant time (if r is constant; $O(r)$ time and $O(r \cdot f(r))$ in general, where f is the expansion function), and only needs the Congest model. We have also shown that our analysis of the algorithm's approximation quality is tight, and show a lower bound of $2r + 1$ for any algorithm that runs in $O(r)$ time. For super-constant r, this leaves a gap, and it remains an open question whether the algorithm or the analysis can be improved.

Furthermore, we have shown how the algorithm can be made to work for similar graph cover problems (vertex cover, connected dominating set, connected vertex cover), and also in the more restrictive port numbering model.

Of course, this result only applies to graph classes satisfying specific conditions; therefore, many other interesting graph classes remain open. For example, in how far can the girth requirement be removed? Is there a better parameterization than $f(r)$? Clearly, the landscape of distance-r dominating set complexity needs more exploration.

Part III.

Packet Forwarding

Chapter 9

Introduction

In this chapter, we show the intricacies of "making packets go from here to there". In particular, we revisit the history of packet forwarding, and define the formal framework we need in order to improve the state of the art to cover two packet destinations.

9.1. Motivation

Many processes involve packets being sent through a network. Examples can be as large as the internet, or as small as a System-on-Chip design. In all real-life scenarios, buffers, bandwidth, and time budget are never infinite, and packet loss should be reduced or even avoided entirely.

The problem of packet delivery can be partitioned into routing (path-selection) and forwarding (or more generally: scheduling) [LMR88]. We will focus on the forwarding aspect of the problem. In other words, even assuming that we already know what route a packet is supposed to take, the packet forwarding problem asks in which order and at which time each packet should be sent. This makes a significant difference, and hence we would like to determine a good forwarding algorithm.

In order to measure the performance of an algorithm, we need to carefully consider which injection patterns we want to permit. For example, if we permit arbitrarily many packets to be injected into the system in each round, the problem might simply be unsolvable with finite buffers. To handle this issue, we assume that injections happen according to the "leaky bucket" model [Cru91]: We assume that there is a rate ρ and a burstiness parameter σ, such that if packets were to drain at rate ρ, the number of packets in the system would never exceed σ.

With these assumptions in hand, the question becomes: How large do buffer sizes have to be, in terms of ρ and σ, to guarantee that no packet loss will occur? Observe that even in a simple scenario, the naive algorithm behaves badly Figs. 9.1 and 9.2. This is the question we will explore in this and the following chapter.

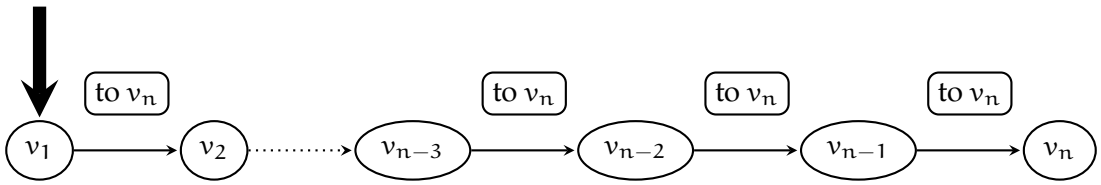

Figure 9.1.: Visualization of the trivial algorithm that always forwards packets seemingly working fine. The graph is just a path, each circle represents a node, each arrow represents a directed edge of the graph. The dotted line indicates that part of the path has been omitted for illustration purposes. The adversary has been injecting a single packet at the first vertex v_1, with the last node as destination, in each round, for $\Omega(n)$ rounds. This is represented by a thick arrow. At this point in time, every buffer contains a single packet, represented by a box.

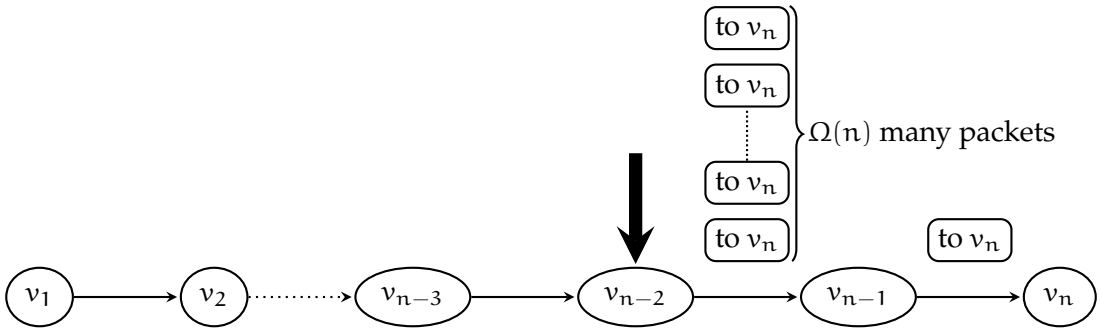

Figure 9.2.: Visualization of the trivial algorithm that always forwards packets resulting in a single buffer storing $\Omega(n)$ many packets. After Fig. 9.1, the adversary proceeded to insert packets at v_{n-2}, one packet per round. However, the algorithm did not stop forwarding the packets previously in the buffers to $v_2, v_3, \ldots, v_{n-2}$. This effectively means that the buffer to v_{n-1} saw two incoming and one outgoing packet, for $\Omega(n)$ many rounds.
Therefore, the naive algorithm either requires large buffers, or has to suffer packet loss. It is known [Dob+17; PR17] that this problem can be avoided.

9.2. Related Work

Early work on the Adversarial Queuing Theory (AQT) model [Bor+01] focused on the qualitative measure of stability. Here, a protocol is *stable* if and only if its maximum buffer space usage is bounded over all (ρ, σ)-bounded injection patterns (for some pre-specified ρ which is at most the edge capacity). More recently, Miller and Patt-Shamir [MP16] initiated the quantitative study of the maximum buffer space usage. Their work considered the special case where G is a tree and all packets share a common destination. They showed that buffer space of $2\rho + \sigma$ is necessary and sufficient for all ρ, σ, so long as $\rho \leqslant C$ (the capacity of all edges in the network). Miller and Patt-Shamir's protocol, however, is centralized: the decision of whether or not an individual buffer forwards a packet may depend on the entire network configuration. They left as an open question the space complexity of *local* forwarding in trees with a single destination. That is, what is the space complexity if the behavior of each node is based only on the current state of its local ($O(1)$ distance) neighborhood?

This question was answered independently by Dobrev et al. [Dob+17] and Patt-Shamir and Rosenbaum [PR17]. Both papers show that $\Theta(\rho \cdot \log n + \sigma)$ buffer space is necessary and sufficient for local forwarding in a tree when all packets have the same destination. The two papers describe essentially the same algorithm that achieves this complexity, called *odd-even downhill* (*OED*) forwarding. The results of [Dob+17; PR17] were later generalized by Patt-Shamir and Rosenbaum [PR19], who showed an optimal locality/buffer-space trade-off. Interestingly, this latter work implies that $O(\log n)$ locality is sufficient to achieve the same (asymptotic) space complexity as the centralized protocol of Miller and Patt-Shamir [MP16].

While the results above settle the space complexity of (local) forwarding in the AQT in single-destination trees, they do not address the case where packets can have multiple destinations. In [PR17], Patt-Shamir and Rosenbaum show that if there are d possible destinations, then $\Omega(d+\sigma)$ buffer space is required, even for centralized offline algorithms (i.e., the entire injection pattern is known to the algorithm in advance). This lower bound was known to be tight for $d = \Omega(n)$, as Adler and Rosén proved that a simple greedy (local) protocol achieves $O(n + \sigma)$ buffer space in arbitrary direct acyclic graphs (DAGs) with arbitrary destinations [AR02]. In [MPR19], Miller et al. prove a tight buffer space upper bound of $O(d + \sigma)$ for trees with d possible destinations, although once again their solution is centralized. They leave open the question of whether a similar result can be achieved by a local protocol.

Given the results of [Dob+17; PR17] and [AR02], it would be natural to conjecture that $O(d \log n)$ buffer space is achievable by a local protocol. Indeed, [Dob+17; PR17] and [AR02] show that this is the case for $d = 1$ and $d = \Omega(n)$, respectively. However, we are unaware of any existing local algorithm that can achieve $o(n)$ buffer space usage even for two distinct destinations on a path. This is the direction that we want to explore.

9.3. Contribution: Generalization to Two Destinations

We address the open problem of Packet Forwarding with multiple destinations. In particular, we show that under a mild relaxation of the model in which packets can move backward and forward in the network,[1] $O(\log n)$ buffer space is achievable by a local protocol when packets may have 2 distinct destinations along a path. Specifically, we show the following:

Theorem 46. *Let $G = (V, E)$ be a path of length n and suppose all edges have bi-directional capacity 1, i.e. for each edge $e = (u, v)$,* both *from u to v and from v to u a single packet can cross in each round. There exists a local forwarding protocol such that for all (ρ, σ)-bounded injection patterns, the maximum buffer space usage is $O(\log n + \sigma)$.*

While this result is only a modest generalization of the results of [Dob+17; PR17], it requires a nontrivial modification of known techniques (and many "natural" generalizations of the OED algorithm of [Dob+17; PR17] do not achieve this bound).

The basic idea of our algorithm for two destinations is to simulate two executions of the OED forwarding algorithm of [Dob+17; PR17]: one for the farther destination, and one for all packets (with either destination). Suppose the two destinations are s_1 and s_2, with s_2 the farther (i.e., rightmost) destination. Each round, the simulated OED instances determine for each simulated buffer whether it should forward a packet. Our algorithm determines the actual packet forwarding according to the following rules:

- if neither simulated instance forwards, no packet is forwarded;
- if both simulated instances forward, a packet with destination s_2 is forwarded;
- if only the "destination oblivious" OED instance forwards a packet, then a packet with destination s_1 is forwarded;
- if only the s_2 destination OED instance forwards a packet, then the buffer swaps a packet with its right neighbor—an s_2-destined packet is sent forward, while an s_1-destined packet is sent backward across the edge.

Thus, packets are only pushed backwards as the result of "packet swaps" in the last rule. In our analysis, we show that forwarding according to these rules is always feasible and that the maximum buffer load is at most the sum of the loads of two single-destination instances on the path. Theorem 46 then follows from the guarantees for OED forwarding proven in [Dob+17; PR17].

In Section 9.4, we introduce our computational model more formally. Section 10.1 formalizes the 2 destination algorithm and proves Theorem 46. Finally, we conclude with open questions in Section 10.2.

[1] We note that the $\Omega(\log n)$ space lower bound of [Dob+17; PR17] still applies in this model with bi-directional packet movement.

9.4. Preliminaries and Notation

In this paper, we consider a simple queuing network consisting of a directed path. We model the network as a (directed) graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_{n+1}\}$ and $E = \{(v_i, v_{i+1}) | i \in [n]\}$. Each outgoing edge represents a buffer that stores packets as they wait to traverse the edge. Each edge has a *capacity* that designates the number of packets that may simultaneously cross the edge. For simplicity, we assume that all edges have capacity 1.

Our execution model proceeds in synchronous *rounds*. Each round consists of two *steps*:

arrival step during which new packets spontaneously arrive via *injections* into buffers in the network

forwarding step during which each node/buffer decides which (if any) packets to forward during the round, and forwards packets.

Packet transmission across an edge is assumed to be faithful. That is, every packet forwarded during a forwarding step successfully crosses its edge before the beginning of the next round. After forwarding, a packet is placed in the next buffer along its path, or removed from the network if the next node is the packet's destination. This edge-buffered approach may seem inconvenient on a directed path, but simplifies generalization to other graphs and comparability to other results.

Packet injections into the network are adversarial, but subject to some constraints. Each packet arrives with a pre-determined path or *route* from its source (i.e., injection site) to its destination, and a packet is only removed from the network upon reaching its destination. An *injection pattern* consists of a potentially infinite sequence of packet injections. We parameterize injection patterns by two parameters, ρ and σ, where ρ is the average *rate* of the pattern, and σ is its *burstiness*. Specifically, an injection pattern is (ρ, σ)-*bounded* if for every edge e and any T consecutive rounds $r_0, r_0 + 1, \ldots, r_0 + T - 1$, the number of packets injected in rounds $r_0, \ldots, r_0 + T - 1$ whose routes contain e is at most $\rho \cdot T + \sigma$.

A *forwarding protocol* determines for each configuration (i.e., state of all buffers in the network) which and how many packets should be forwarded across each edge in the network. In the AQT model, the quality of a forwarding protocol is determined by the maximum buffer usage—that is, the maximum load of any buffer taken over all buffers and rounds—over all (ρ, σ)-bounded injection patterns.

As we want to consider local algorithms, nodes are not informed about these injected packets. Instead, they must observe their incident buffers, and communicate with their neighbors during the forwarding step.

In contrast to the previous single destination trees [PR17], we will consider *two* destinations, or sink-nodes, and name them s_1 and s_2 respectively. For simplicity, we will assume that s_2 is reachable from s_1 (i.e. s_1 appears first in the directed path).

Define the depth as the maximum distance between $v \in V$ and $w \in \{s_1, s_2\}$, i.e. the depth.

We define $L_X^t : E \to \mathbb{N}$ as a destination-specific configuration at time t, mapping each edge e to the amount of packets (or "load") at edge e that needs to be sent across it towards destination X. Because we only consider two destinations, X is in $\{s_1, s_2\}$, so we define the short-hand notation $L_i^t := L_{s_i}^t$. We define $L^t := (L_1^t, L_2^t)$ as the overall configuration at time t. Note that L^t describes the entire state at time t.

At time $t = 0$, no packages are in transit: $\forall X : L_X^0 = 0$.

Define $L_{1+2}^t := L_1^t + L_2^t$ as the total amount of packets waiting on an edge. With this, we can define the cost of an algorithm F on a specific graph G with a specific adversary A as $\text{cost}(F; G, A) := \sup\{L_{1+2}^t(e) \mid t \in \mathbb{N}, e \in E\}$. For a class of adversaries $\mathcal{A}$, we denote $\text{cost}(F; \mathcal{G}, \mathcal{A}) := \sup\{\text{cost}(F; G, A) \mid G \in \mathcal{G}, A \in \mathcal{A}\}$. Our goal is to bound the cost for the class of adversaries $\mathcal{A}(\rho, \sigma)$ respecting the AQT parameters (ρ, σ).

Finally, we define a notational short-hand:

Definition 6

Let $\text{OED}(L, e, f)$ be the boolean-valued function that evaluates the OED-criterion on some configuration L, for a node that forwards from edge e to f:

$$\text{OED}(L, e, f) := \Big(L(e) > L(f)\Big) \textbf{ or } \Big(L(e) = L(f) \textbf{ and } L(f) \text{ is odd}\Big)$$

Chapter 10

Two-Destination Odd-Even Downward

This chapter presents findings that have not been published. These findings are a non-trivial extension of an already-existing algorithm. This contribution is an important step towards a fully-generalized result.

10.1. The Swapping Algorithm

We extend the model by permitting the algorithm to swap packets, i.e. send packets in both directions of a communication link. Using this extension, we can construct an algorithm OED^2_C that uses the single-destination algorithm OED in order to run in the two destination model.

To this end, we permit a *swap action* as an alternative that the algorithm may choose during the *forwarding step*: If a node v wants to execute a swap action, it does so by forwarding a packet p_1 to the next buffer e towards packet p_1's destination. Simultaneously, node v takes a packet p_2 from buffer e and moves it into its own buffer. Note that this *increases* the distance of packet p_2 to its destination, yet it proves to be useful. It is the algorithm's responsibility to ensure that no conflict arises from this, i.e., the algorithm must not attempt to remove a packet that does not exist.

In short, our algorithm attempts to run $\text{OED}(L_2)$ and $\text{OED}(L_{1+2})$ simultaneously, and resolves conflicts using a case-distinction, sometimes involving the swap action. We formalize this in Alg. 9.

While this appears simple enough, we need to show that these operations are always feasible and has the desired effect. In particular, we need to verify that a swap action will actually have packets to swap available.

Algorithm 9: Swapping Algorithm OED^2_C at node $v \in V$ between edges e and f

```
1: if v = s_i and e contains a packet with destination s_i then
2:     Remove the packet with destination v from e
3: else if not OED(L_2, e, f) and OED(L_{1+2}, e, f) then
4:     e forwards a packet with destination s_1 to f
5: else if OED(L_2, e, f) and not OED(L_{1+2}, e, f) then
6:     // execute a swap action
7:     e forwards a packet with destination s_2 to f
8:     f backwards a packet with destination s_1 to e
9: else if OED(L_2, e, f) and OED(L_{1+2}, e, f) then
10:    // both simulations want to forward a packet
11:    e forwards a packet with destination s_2 to f
12: end if
```

Lemma 47. *If* OED^2_C *attempts to forward a packet of a specific type out of a buffer* $e := (u, v)$ *to the next buffer* $f := (v, w)$*, then buffer* e *contains at least one such packet.*

Proof. For lines 7 and 11, this is obvious from the fact that $\mathrm{OED}(L_2, e, f)$ is true.

For line 4, we can use the fact that $\mathrm{OED}(L_2, e, f)$ is false, but $\mathrm{OED}(L_{1+2}, e, f)$ is true.

- If $L_2(f)$ is odd, we can deduce that $L_2(e) < L_2(f)$ (because equality would make the OED criterion true), and that $L_{1+2}(e) \geqslant L_{1+2}(f)$. Expanding the terms:

$$L_1(e) + L_2(e) = L_{1+2}(e) \geqslant L_{1+2}(f) \geqslant L_2(f) > L_2(e)$$

- If $L_2(f)$ is even, we can deduce that $L_2(e) \leqslant L_2(f)$. Observe that $L_{1+2}(e) = L_2(e)$ is impossible, as it would imply $L_{1+2}(e) \leqslant L_2(f) \leqslant L_{1+2}(f)$, which cannot satisfy $\mathrm{OED}(L_{1+2}, e, f)$ while being even.

This shows that $L_1(e) > 0$ in both cases. □

Lemma 48. *If* OED^2_C *tries to remove more than one packet of a specific type from a buffer, then there are sufficiently many such packets.*

Proof. Observe that this can only happen if a node v decides to forward a specific packet type while the preceding node u executes a swap action, grabbing the same packet type.

As we want to argue about buffers instead of nodes, we will define the shorthand names $d := (\mathrm{pred}_u, u)$, $e := (u, v)$, and $f := (v, \mathrm{succ}_v)$, where pred_u is the preceding node of u, and succ_v is the succeeding node of v. For a visualization, see Fig. 10.1.

This scenario happens only if $\mathrm{OED}(L_2, d, e) = \mathit{true}$ and $\mathrm{OED}(L_{1+2}, d, e) = \mathit{false}$ (hence node u's behavior), and $\mathrm{OED}(L_2, e, f) = \mathit{false}$ as well as $\mathrm{OED}(L_{1+2}, e, f) = \mathit{true}$ (hence node v's behavior).

We already know that $L_1(e) \geqslant 1$ from Lem. 47, because node v decided to forward a packet with destination s_1. What we need to show is that $L_1(e) \neq 1$.

Assume towards contradiction that $L_1(e) = 1$, and distinguish whether $L_2(e)$ is even or odd.

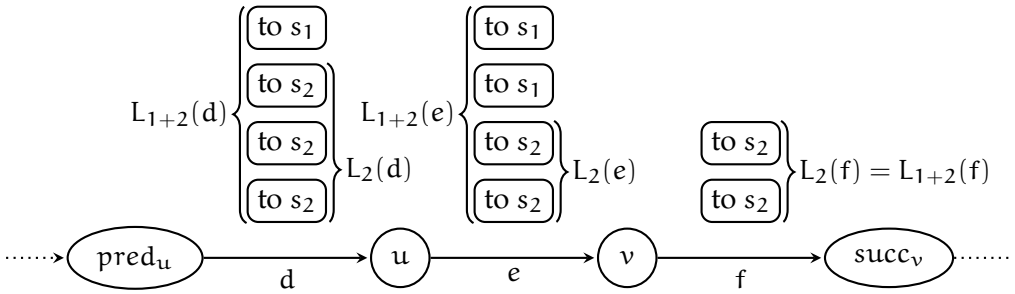

Figure 10.1.: Visualization of a situation in which two packets of the same type are removed from the same buffer.

- If $L_2(e)$ is odd: From $\mathrm{OED}(L_2, e, f) = \textit{false}$ we know that $L_2(e) + 1 \leqslant L_2(f)$. From $\mathrm{OED}(L_{1+2}, e, f) = \textit{true}$ (and the fact that $L_{1+2}(e) = L_2(e) + 1$ must be even) we know that $L_{1+2}(e) > L_{1+2}(f)$, implying the contradiction:

$$L_2(e) + 1 = L_{1+2}(e) > L_{1+2}(f) \geqslant L_2(f) \geqslant L_2(e) + 1.$$

- If $L_2(e)$ is even: From $\mathrm{OED}(L_2, d, e) = \textit{true}$ we know that $L_2(d) \geqslant L_2(e)+1 = L_{1+2}(e)$, which implies $L_{1+2}(d) \geqslant L_2(d) \geqslant L_{1+2}(e)$. From $\mathrm{OED}(L_{1+2}, d, e) = \textit{false}$ (and the fact that $L_{1+2}(e) = L_2(e) + 1$ must be odd) we know that $L_{1+2}(d) < L_{1+2}(e)$. □

In order to prepare for showing an upper bound on the maximum buffer load, let us recall the respective bound for the plain OED algorithm.

Theorem 49 (Theorem 3.1 of [PR17])**.** *Let $\mathcal{G}$ be the family of single destination trees, and $\mathcal{A} = \mathcal{A}(\rho, \sigma)$ with $\rho \leqslant 1$. There exists a local protocol OED such that* $\mathrm{cost}(\mathrm{OED}, G, \mathcal{A}) \leqslant 2\log_3(\mathrm{depth}(G)) + 2\sigma + 2$.

From this, we immediately get a bound on L_2:

Corollary 50. *Let $\mathcal{G}$ be the family of connected, directed lines with two destinations s_1 not after s_2. Furthermore, consider the adversary $\mathcal{A} = \mathcal{A}(\rho, \sigma)$ with $\rho \leqslant 1$. Then OED^2_C maintains $\forall t \in \mathbb{N}, e \in E : L_2^t(e) \leqslant 2\log_3 \mathrm{depth}(G) + 2\sigma + 2$.*

Proof. Follows directly from the fact that we effectively execute the original OED algorithm on all packets with destination s_2. □

For L_{1+2}, matters are not as simple. The adversary is free to insert up to 2 packets concurrently, without a burst, so long as both of them are routed through disjoint sets of edges; for example, one packet with destination s_1, and one packet injected behind s_1 with destination s_2. To approach this issue, we first need to find a way to separate the counting of s_1 and s_2 packets.

Definition 7

Fix an arbitrary number $H \geqslant 0$ and an adversary A, specified as the sequence of injected packets or absence thereof. In the context of an execution L, we define $\mathrm{top}_H(A)$ as the adversary that inserts packets as specified by A, but only if the target buffer e has size at least H (i.e. $L^t(e) \geqslant H$); otherwise, the packet is never inserted.

With this, we can show that ignoring the "bottom part" of the buffers is viable, in the sense that we can ignore packets that get injected there.

Lemma 51. *Fix an arbitrary even number $H \geqslant 0$, adversary A, and sequence of configurations L such that all nodes follow the* OED *algorithm. We can construct an adversary $\check{A}$ and sequence of configurations $\check{L}$ by removing the bottom of each buffer, such that all nodes appear to execute the* OED *algorithm:*

$$\forall t, e : \check{L}^t(e) \coloneqq \max\{0, L^t(e) - H\} \tag{10.1}$$

$$\check{A} \coloneqq \mathrm{top}_H(A) \tag{10.2}$$

$$\implies \check{L} \textit{ follows OED} \tag{10.3}$$

Proof. If $\mathcal{A}$ injects a packet into a buffer e with $L(e) < H$, it cannot cause the packet to "appear" in $\check{L}$. More formally, this never results in $\check{L}(e) > 0$. Therefore, the adversary $\mathcal{A}$ is effectively reduced to $\mathrm{top}_H(\mathcal{A})$, or fewer packets.

What remains to be shown is that all nodes follow the OED algorithm. Observe that OED only evaluates basic conditions: comparing buffer sizes and their parity. These properties are preserved by construction, if at least one buffers satisfies $\check{L} > 0$, therefore OED must take the same action on $\check{L}$ as on L.

This leaves the case in which both buffers satisfy $\check{L} = 0$. Observe that OED will never forward a packet from a buffer e with $L^t(e) \leqslant H$ into a buffer f with $L^t(f) \geqslant H$, because H is even. Thus, in $\check{L}$ it appears that the nodes did nothing at all, which is precisely what the OED algorithm prescribes for the case $\check{L}(e) = \check{L}(f) = 0$. □

Theorem 52. *Let $\mathcal{G}$ be the family of connected, directed lines with two destinations s_1 not after s_2. Furthermore, consider the adversary $\mathcal{A} = \mathcal{A}(\rho, \sigma)$ with $\rho \leqslant 1$. Then* OED^2_C *achieves, for all $G \in \mathcal{G}, A \in \mathcal{A}$:* $\mathrm{cost}(\mathrm{OED}^2_C, G, A) \leqslant 4\log_3(\mathrm{depth}(G)) + 4\sigma + 6$

Proof. By Cor. 50, $L_2 \leqslant 2\log_3(\mathrm{depth}(G)) + 2\sigma + 2$.

However, Thm. 49 does not as readily apply to L_{1+2}. We can use the bound on L_2 to choose the size of the "bottom" part:

$$H \coloneqq 2\left\lceil \log_3(\mathrm{depth}(G))\right\rceil + 2\sigma + 2 \tag{10.4}$$

By construction, H is even. This allows us to apply Lem. 51, by which we only need to consider the adversary $\mathrm{top}_H(\mathcal{A})$, for a reduced packet load $\check{L}_{1+2}$. For our choice of H, this adversary only injects packets with destination s_1. Since algorithm OED^2_C also behaves as if $\mathrm{OED}(L_{1+2})$ was executed, by Lem. 51 it also behaves as if $\mathrm{OED}(\check{L}_{1+2})$ was executed.

Now we can apply Thm. 49 again to conclude that

$$\begin{aligned} L_{1+2} &\leqslant H + \check{L}_{1+2} \\ &\leqslant H + 2\log_3(\text{depth}(G)) + 2\sigma + 2 \\ &< 4\log_3(\text{depth}(G)) + 4\sigma + 6. \end{aligned}$$

□

Thm. 46 is an immediate corollary of Thm. 52. In summary: If we permit backwarding of packets, then we have an algorithm that only needs $O(\log \text{depth}(G))$ buffers at each node, even in face of adversarial injections for two destinations.

10.2. Conclusion

We have presented an algorithm for two destinations that uses a net packet flow of at most one packet across each edge. In particular, observe that a naive attempt to "run two OED instances at the same time" fails, as it might attempt to forward two packets at the same time.

Future work might show that this can be generalized to d many destinations, while maintaining a net flow of 1; ideally in arbitrary graphs, with arbitrary placement of destinations. This is not trivial, as additional destinations provide additional opportunities for conflict, i.e. multiple nodes might attempt to remove the same packets.

An open question remains whether some or even all of the backwarding can be replaced by some kind of virtualization, in order to achieve the desired bandwidth restriction of one packet per edge and round. We have investigated several schemes but could find neither an impossibility result nor a promising candidate, so this question seems to be non-trivial.

Appendix A

TRIX Simulations

A.1. Potential Systematic Errors

In this section we discuss possible sources of systematic errors in our simulations and how we guarded against them.

A.1.1. Bugs

Since all experiments are software simulations, measurements have to be taken to insure against bugs. In this regard, there are several arguments to be made, some involving the Random Number Generator (RNG):

- We cross-validated four different implementations: (1) A very simple Python implementation that uses system randomness; (2) a slightly more involved Python implementation that exhaustively enumerates all possible wire delay combinations; (3) a straight-forward C implementation using system randomness; and (4) an optimized C implementation with the slightly weaker RNG "`xoshiro512starstar`" [BV18].

- Even though only implementation (4) was fast enough to be used to generate the bulk of the results, all implementations agree on the probability distributions of delay and skew for examples they can handle. Implementation (2) can only run up to layer 3 ($H \leqslant 3$), implementations (1) and (3) were used up to around layers 100 and 1000, respectively.

- Implementation (4) is short (200 lines, plus about 100 lines for the RNG [BV18]), and is simple enough to be inspected manually.

- Multiple machines were used, so hardware failure can be ruled out with sufficient confidence.

A.1.2. Randomness and Model

- Tests with a number of weak RNGs showed that TRIX seems to be robust against this kind of deviation.
- Explorative simulations and validation simulations were kept strictly separate.
- For most discussions we assumed $H = 2000$, because this definitely covers all practical applications. In fact, we expect that many applications only need $H = 200$ or even $H = 20$. In this paper we use a large value for H to show that the observed behavior is not a fluke that occurs due to low H, but that the growth of delay and skew as function of H is indeed asymptotically slow.
- Using a larger domain only scales the result linearly, as expected.
- Using different wire delay models (e.g. choosing uniformly from $\{0, 0.5, 1\}$ instead of $\{0, 1\}$) does not significantly change the result, and in fact improves it slightly, as expected.
- We only focused on fault-free executions. Observe that single isolated faults only introduce an additional uncertainty of at most 1 (recall the normalization $u = 1$). As faults are (supposed to be) rare and not maliciously placed, this means that the predictions for fault-free systems have substantial and meaningful implications also for systems with faults.

A.2. Figure Data

In the following, we provide the data for all graphs.

Fig. 3.1		Fig. 3.2	
x (delay)	y (rate)	x (delay)	y (normal)
985	0.00000012	985.5	985.843
986	0.00000040	986.5	986.615
987	0.00000140	987.5	987.338
988	0.00001076	988.5	988.457
989	0.00004740	989.5	989.460
990	0.00019060	990.5	990.462
991	0.00066280	991.5	991.457
992	0.00206788	992.5	992.464

Fig. 3.1		Fig. 3.2	
x (delay)	y (rate)	x (delay)	y (normal)
993	0.00561240	993.5	993.470
994	0.01324480	994.5	994.472
995	0.02753772	995.5	995.475
996	0.05006436	996.5	996.479
997	0.08002964	997.5	997.486
998	0.11158960	998.5	998.492
999	0.13625144	999.5	999.498
1000	0.14553656	1000.5	1000.503
1001	0.13611860	1001.5	1001.508
1002	0.11158608	1002.5	1002.515
1003	0.07996644	1003.5	1003.520
1004	0.05016040	1004.5	1004.526
1005	0.02753824	1005.5	1005.531
1006	0.01324832	1006.5	1006.537
1007	0.00557100	1007.5	1007.542
1008	0.00205200	1008.5	1008.545
1009	0.00066300	1009.5	1009.545
1010	0.00018904	1010.5	1010.552
1011	0.00004660	1011.5	1011.556
1012	0.00001044	1012.5	1012.650
1013	0.00000144	1013.5	1013.385
1014	0.00000044	1014.5	1014.363
1015	0.00000008		

	Fig. 3.3	Figs. 3.5 and 3.6
x (H)	y (stddev delay)	y (stddev skew)
20	0.9012352	0.74096549
50	1.1147817	0.75071102
100	1.3166986	0.75573837
200	1.5567596	0.75993333
500	1.9468302	0.76314438
1000	2.3115292	0.76499614
2000	2.7406959	0.76601516
5000	3.4405614	0.76771794

Fig. 3.4

x (skew)	lower bound	observed rate	upper bound
-7	0.0000000	0.0000001	0.0000010
-6	0.0000000	0.0000000	0.0000009
-5	0.0000010	0.0000022	0.0000041
-4	0.0000324	0.0000389	0.0000452
-3	0.0007020	0.0007322	0.0007578
-2	0.0142418	0.0143777	0.0144897
-1	0.2283952	0.2287592	0.2291231
0	0.5120736	0.5124375	0.5128015
1	0.2281477	0.2285116	0.2288756
2	0.0142291	0.0143650	0.0144769
3	0.0007024	0.0007326	0.0007583
4	0.0000343	0.0000410	0.0000475
5	0.0000009	0.0000020	0.0000038
6	0.0000000	0.0000002	0.0000013

Fig. 3.7

hop dist.	empiric stddev
1	0.76334
2	0.94346
3	1.11873
4	1.26012
5	1.38309
6	1.49367
7	1.59118
8	1.67803
9	1.75621
10	1.83219
11	1.90233
12	1.96331
13	2.02323
14	2.07683
15	2.12580
16	2.17155
17	2.21355
18	2.25797
19	2.29356
20	2.32732
21	2.35812
22	2.38959
23	2.41912
24	2.44794
25	2.46569

Bibliography

[Abo+15] F. Abouzeid et al. "28nm FD-SOI technology and design platform for sub-10pJ/cycle and SER-immune 32bits processors". In: *ESSCIRC*. 2015, pp. 108–111.

[AFS20] Saeed Akhoondian Amiri, Klaus Foerster, and Stefan Schmid. "Walking Through Waypoints". In: *Algorithmica* (Jan. 2020). DOI: 10.1109/TIT.2014.2339840.

[Alt+93] Ingo Althöfer et al. "On Sparse Spanners of Weighted Graphs". In: *Discrete & Computational Geometry* 9 (1993), pp. 81–100. DOI: 10.1007/BF02189308. URL: https://doi.org/10.1007/BF02189308.

[Ami+18] Saeed Akhoondian Amiri et al. "Distributed domination on graph classes of bounded expansion". In: *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*. ACM. 2018, pp. 143–151.

[Ami21] Saeed Akhoondian Amiri. "Deterministic CONGEST Algorithm for MDS on Bounded Arboricity Graphs". In: *CoRR* abs/2102.08076 (2021). arXiv: 2102.08076. URL: https://arxiv.org/abs/2102.08076.

[AR02] Micah Adler and Adi Rosén. "Tight Bounds for the Performance of Longest-in-System on DAGs". In: *STACS 2002: 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14–16, 2002 Proceedings*. Ed. by Helmut Alt and Afonso Ferreira. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 88–99. ISBN: 978-3-540-45841-8. DOI: 10.1007/3-540-45841-7_6. URL: http://dx.doi.org/10.1007/3-540-45841-7_6.

[AS10] Matti Astrand and Jukka Suomela. "Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks". In: *SPAA 2010: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, Thira, Santorini, Greece*. 2010, pp. 294–302. DOI: 10.1145/1810479.1810533. URL: https://doi.org/10.1145/1810479.1810533.

[AS16] Saeed Akhoondian Amiri and Stefan Schmid. "Brief Announcement: A Log-Time Local MDS Approximation Scheme for Bounded Genus Graphs". In: *Proc. 30th International Symposium on Distributed Computing (DISC)*. Springer. 2016.

[ASS16] Saeed Akhoondian Amiri, Stefan Schmid, and Sebastian Siebertz. "A Local Constant Factor MDS Approximation for Bounded Genus Graphs". In: *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*. 2016.

[AW21] Saeed Akhoondian Amiri and Ben Wiederhake. "Distributed Distance-r Covering Problems on Sparse High-Girth Graphs". In: *Algorithms and Complexity - 12th International Conference, CIAC 2021, Virtual Event, May 10-12, 2021, Proceedings*. Ed. by Tiziana Calamoneri and Federico Corò. Vol. 12701. Lecture Notes in Computer Science. Springer, 2021, pp. 37–60. DOI: 10.1007/978-3-030-75242-2_3. URL: https://doi.org/10.1007/978-3-030-75242-2_3.

[AW22] Saeed Akhoondian Amiri and Ben Wiederhake. "Distributed distance-r covering problems on sparse high-girth graphs". In: *Theoretical Computer Science* (2022). ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2022.01.001. URL: https://www.sciencedirect.com/science/article/pii/S0304397522000081.

[Awe+89] Baruch Awerbuch et al. "Network Decomposition and Locality in Distributed Computation". In: *30th Annual Symposium on Foundations of Computer Science, 1989*. 1989, pp. 364–369.

[Awe+92] Baruch Awerbuch et al. "Fast Network Decomposition". In: *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing*. PODC '92. ACM, 1992, pp. 169–177.

[Bac+19] Nir Bachrach et al. "Hardness of Distributed Optimization". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019*. ACM, 2019, pp. 238–247. DOI: 10.1145/3293611.3331597.

[Bak94] Brenda S Baker. "Approximation algorithms for NP-complete problems on planar graphs". In: *Journal of the ACM (JACM)* 41.1 (1994), pp. 153–180.

[Bal+21] Alkida Balliu et al. "Locally Checkable Labelings with Small Messages". In: *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*. Ed. by Seth Gilbert. Vol. 209. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 8:1–8:18. DOI: 10.4230/LIPIcs.DISC.2021.8. URL: https://doi.org/10.4230/LIPIcs.DISC.2021.8.

[Bar+20] Reuven Bar-Yehuda et al. "Distributed Approximation on Power Graphs". In: *Proceedings of the 2020 ACM Symposium on Principles of Distributed Computing, PODC 2020*. ACM, 2020, pp. 501–510. DOI: 10.1145/3382734.3405750.

[BCS16] Reuven Bar-Yehuda, Keren Censor-Hillel, and Gregory Schwartzman. "A Distributed (2+ϵ)-Approximation for Vertex Cover in O(logδ/ϵ log log δ) Rounds". In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA*. 2016, pp. 3–8. DOI: 10.1145/2933057.2933086. URL: https://doi.org/10.1145/2933057.2933086.

[Bor+01] Allan Borodin et al. "Adversarial Queuing Theory". In: *J. ACM* 48.1 (Jan. 2001), pp. 13–38. ISSN: 0004-5411. DOI: 10.1145/363647.363659. URL: http://doi.acm.org/10.1145/363647.363659.

[Bra+17] Sebastian Brandt et al. "LCL Problems on Grids". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*. Ed. by Elad Michael Schiller and Alexander A. Schwarzmann. ACM, 2017, pp. 101–110. DOI: 10.1145/3087801.3087833. URL: https://doi.org/10.1145/3087801.3087833.

[BV18] David Blackman and Sebastiano Vigna. "Scrambled Linear Pseudorandom Number Generators". In: *CoRR* abs/1805.01407 (2018). arXiv: 1805.01407. URL: http://arxiv.org/abs/1805.01407.

[Chi+11] R. Chipana et al. "SET susceptibility analysis in buffered tree clock distribution networks". In: *RADECS*. 2011, pp. 256–261.

[Chi+12] R. Chipana et al. "SET susceptibility estimation of clock tree networks from layout extraction". In: *LATW*. 2012, pp. 1–6.

[CHS06] Andrzej Czygrinow, Michał Hańćkowiak, and Edyta Szymańska. "Distributed approximation algorithms for planar graphs". In: *Italian Conference on Algorithms and Complexity*. Springer. 2006, pp. 296–307.

[CHW08] Andrzej Czygrinow, Michal Hańćkowiak, and Wojciech Wawrzyniak. "Fast Distributed Approximations in Planar Graphs". In: *Distributed Computing*. Springer Berlin Heidelberg, 2008, pp. 78–92.

[CHW22] Andrzej Czygrinow, Michał Hanćkowiak, and Marcin Witkowski. "Distributed distance domination in graphs with no $K_{2,t}$-minor". In: *arXiv* (2022).

[CK14] R. Chipana and F. L. Kastensmidt. "SET Susceptibility Analysis of Clock Tree and Clock Mesh Topologies". In: *ISVLSI*. 2014, pp. 559–564.

[Cru91] Rene L. Cruz. "A calculus for network delay, Part I: Network elements in isolation". In: *IEEE Trans. Inf. Theory* 37.1 (1991), pp. 114–131. DOI: 10.1109/18.61109. URL: https://doi.org/10.1109/18.61109.

[CSG98] David Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers Inc., 1998.

[Czy+18] Andrzej Czygrinow et al. "Distributed Approximation Algorithms for the Minimum Dominating Set in K_h-Minor-Free Graphs". In: *29th International Symposium on Algorithms and Computation, ISAAC*. 2018, 22:1–22:12.

[DDP03] Ariel Daliot, Danny Dolev, and Hanna Parnas. "Self-stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks". In: *Proc. 6th International Symposium on Self-Stabilizing Systems (SSS)*. 2003, pp. 32–48.

[DH07] Danny Dolev and Ezra N. Hoch. "Byzantine Self-stabilizing Pulse in a Bounded-Delay Model". In: *Proc. 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems (2007)*. 2007, pp. 234–252.

[DHS86] Danny Dolev, Joseph Y. Halpern, and H. Raymond Strong. "On the Possibility and Impossibility of Achieving Clock Synchronization". In: *J. Comput. Syst. Sci.* 32.2 (1986), pp. 230–250.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*. Vol. 173. Graduate texts in mathematics. Springer, 2012.

[DKM19] Janosch Deurer, Fabian Kuhn, and Yannic Maus. "Deterministic Distributed Dominating Set Approximation in the CONGEST Model". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019*. 2019, pp. 94–103.

[DKW56] A. Dvoretzky, J. Kiefer, and J. Wolfowitz. "Asymptotic Minimax Character of the Sample Distribution Function and of the Classical Multinomial Estimator". In: *Ann. Math. Statist.* 27.3 (Sept. 1956), pp. 642–669. DOI: 10.1214/aoms/1177728174.

[Dob+17] Stefan Dobrev et al. "Optimal Local Buffer Management for Information Gathering with Adversarial Traffic". In: *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*. 2017, pp. 265–274. DOI: 10.1145/3087556.3087577. URL: https://doi.org/10.1145/3087556.3087577.

[Dol+14] Dolev Dolev et al. "Fault-tolerant Algorithms for Tick-generation in Asynchronous Logic". In: *J. ACM* 61.5 (2014), 30:1–30:74.

[Dol+16] Danny Dolev et al. "HEX: Scaling honeycombs is easier than scaling clock trees". In: *J. Comput. Syst. Sci.* 82.5 (2016), pp. 929–956. DOI: 10.1016/j.jcss.2016.03.001. URL: https://doi.org/10.1016/j.jcss.2016.03.001.

[DW04] S. Dolev and J. L. Welch. "Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults". In: *Journal of the ACM* 51.5 (2004), pp. 780–799.

[Gha19] Mohsen Ghaffari. "Distributed Maximal Independent Set using Small Messages". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*. 2019, pp. 805–820.

[GS14] Mika Göös and Jukka Suomela. "No sublogarithmic-time approximation scheme for bipartite vertex cover". In: *Distributed Computing* 27.6 (Dec. 2014), pp. 435–443. ISSN: 1432-0452. DOI: 10.1007/s00446-013-0194-z. URL: https://doi.org/10.1007/s00446-013-0194-z.

[Guj+15] A. Gujja et al. "Redundant Skewed Clocking of Pulse-Clocked Latches for Low Power Soft Error Mitigation". In: *RADECS*. 2015, pp. 1–7.

[KMW16] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. "Local Computation: Lower and Upper Bounds". In: *J. ACM* 63.2 (Mar. 2016), 17:1–17:44.

[KSV21] Simeon Kublenz, Sebastian Siebertz, and Alexandre Vigny. "Constant Round Distributed Domination on Graph Classes with Bounded Expansion". In: *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Proceedings*. Vol. 12810. Lecture Notes in Computer Science. Springer, 2021, pp. 334–351. DOI: 10.1007/978-3-030-79527-6_19. URL: https://doi.org/10.1007/978-3-030-79527-6_19.

[Lin92] Nathan Linial. "Locality in Distributed Graph Algorithms". In: *SIAM J. Comput.* 21.1 (1992), pp. 193–201.

[LMR88] Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. "Universal Packet Routing Algorithms (Extended Abstract)". In: *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*. IEEE Computer Society, 1988, pp. 256–269. DOI: 10.1109/SFCS.1988.21942. URL: https://doi.org/10.1109/SFCS.1988.21942.

[LPW13] Christoph Lenzen, Yvonne Anne Pignolet, and Roger Wattenhofer. "Distributed minimum dominating set approximations in restricted families of graphs". In: *Distributed Computing* 26.2 (2013), pp. 119–137.

[LR19] Christoph Lenzen and Joel Rybicki. "Self-Stabilising Byzantine Clock Synchronisation Is Almost as Easy as Consensus". In: *J. ACM* 66.5 (2019), 32:1–32:56. DOI: 10.1145/3339471.

[LW10] Christoph Lenzen and Roger Wattenhofer. "Minimum Dominating Set Approximation in Graphs of Bounded Arboricity". In: *Proc. 24th International Conference on Distributed Computing (DISC)*. 2010, pp. 510–524.

[LW20a] Christoph Lenzen and Ben Wiederhake. "Brief Announcement: TRIX: Low-Skew Pulse Propagation for Fault-Tolerant Hardware". In: *Stabilization, Safety, and Security of Distributed Systems - 22nd International Symposium, SSS 2020, Austin, TX, USA, November 18-21, 2020, Proceedings*. Ed. by Stéphane Devismes and Neeraj Mittal. Vol. 12514. Lecture Notes in Computer Science. Springer, 2020, pp. 295–300. DOI: 10.1007/978-3-030-64348-5_23. URL: https://doi.org/10.1007/978-3-030-64348-5_23.

[LW20b] Christoph Lenzen and Ben Wiederhake. "TRIX: Low-Skew Pulse Propagation for Fault-Tolerant Hardware". In: *CoRR* abs/2010.01415 (2020). arXiv: 2010.01415. URL: https://arxiv.org/abs/2010.01415.

[Mal+16] V. Malherbe et al. "Investigating the single-event-transient sensitivity of 65 nm clock trees with heavy ion irradiation and Monte-Carlo simulation". In: *IRPS*. 2016, SE-3-1-SE-3–5.

[MP16] Avery Miller and Boaz Patt-Shamir. "Buffer Size for Routing Limited-Rate Adversarial Traffic". In: *DISC 2016: Proceedings of the 30th International Symposium on Distributed Computing, Paris, France, September 27-29, 2016*. Springer, 2016, pp. 328–341. ISBN: 978-3-662-53426-7. DOI: 10.1007/978-3-662-53426-7_24. URL: http://dx.doi.org/10.1007/978-3-662-53426-7_24.

[MPR19] Avery Miller, Boaz Patt-Shamir, and Will Rosenbaum. "With Great Speed Come Small Buffers: Space-Bandwidth Tradeoffs for Routing". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. Ed. by Peter Robinson and Faith Ellen. ACM, 2019, pp. 117–126. DOI: 10.1145/3293611.3331614. URL: https://doi.org/10.1145/3293611.3331614.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[ND12] Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*. Vol. 28. Springer Science & Business Media, 2012.

[NS93] Moni Naor and Larry Stockmeyer. "What Can Be Computed Locally?" In: *Proc. ACM 25th Annual ACM Symposium on Theory of Computing (STOC)*. 1993, pp. 184–193.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. ISBN: 0-89871-464-8.

[PH90] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1990.

[PR17] Boaz Patt-Shamir and Will Rosenbaum. "The Space Requirement of Local Forwarding on Acyclic Networks". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*. Ed. by Elad Michael Schiller and Alexander A. Schwarzmann. ACM, 2017, pp. 13–22. DOI: 10.1145/3087801.3087803. URL: https://doi.org/10.1145/3087801.3087803.

[PR19] Boaz Patt-Shamir and Will Rosenbaum. "Space-Optimal Nearly-Local Forwarding on Trees". In: *INFOCOM 2019: Proc. 2019 IEEE Conference on Computer Communications, Paris, France, April 29 – May 2, 2019*. To appear. 2019.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985.

[RG20] Václav Rozhon and Mohsen Ghaffari. "Polylogarithmic-time deterministic network decomposition and distributed derandomization". In: *Proc. ACM 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. ACM, 2020, pp. 350–363. DOI: 10.1145/3357713.3384298.

[Ros19] Matthias Rost. "TopologyZoo Treewidth Analysis". In: *https://github.com/MatthiasRost/topologyzoo-treewidth-analysis* (2019).

[SSS07] Ulrich Schmid, Andreas Steininger, and M. Sust. "FIT-IT-Projekt DARTS: dezentrale fehlertolerante Taktgenerierung". In: *Elektrotech. Informationstechnik* 124.1-2 (2007), pp. 3–8. DOI: 10.1007/s00502-006-0409-0. URL: https://doi.org/10.1007/s00502-006-0409-0.

[ST87] T. K. Srikanth and Sam Toueg. "Optimal Clock Synchronization". In: *J. ACM* 34.3 (1987), pp. 626–645.

[SW08] Johannes Schneider and Roger Wattenhofer. "A log-star distributed maximal independent set algorithm for growth-bounded graphs". In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008*. 2008, pp. 35–44.

[Wan+16] H. B. Wang et al. "Single-Event Transient Sensitivity Evaluation of Clock Networks at 28-nm CMOS Technology". In: *IEEE Trans. Nucl. Sci.* 63.1 (2016), pp. 385–391.

[Wis+09] L. Wissel et al. "Flip-Flop Upsets From Single-Event-Transients in 65 nm Clock Circuits". In: *IEEE Trans. Nucl. Sci.* 56.6 (2009), pp. 3145–3151.

[WL88] Jennifer Lundelius Welch and Nancy A. Lynch. "A New Fault-Tolerant Algorithm for Clock Synchronization". In: *Information and Computation* 77.1 (1988), pp. 1–36.

[Xan09] Thucydides Xanthopoulos, ed. *Clocking in Modern VLSI Systems*. Springer, 2009.