The Computational Cost of Asynchronous Neural Communication

Yael Hitron
Weizmann Institute of Science, Israel
yael.hitron@weizmann.ac.il

Merav Parter
Weizmann Institute of Science, Israel
merav.parter@weizmann.ac.il

Gur Perri
Weizmann Institute of Science, Israel
gur.perri@weizmann.ac.il

Abstract

Biological neural computation is inherently asynchronous due to large variations in neuronal spike timing and transmission delays. So-far, most theoretical work on neural networks assumes the *synchronous* setting where neurons fire simultaneously in discrete rounds. In this work we aim at understanding the barriers of asynchronous neural computation from an algorithmic perspective. We consider an extension of the widely studied model of synchronized spiking neurons [Maass, Neural Networks 97] to the asynchronous setting by taking into account edge and node delays.

- **Edge Delays:** We define an asynchronous model for spiking neurons in which the latency values (i.e., transmission delays) of non self-loop edges *vary* adversarially over time. This extends the recent work of [Hitron and Parter, ESA'19] in which the latency values are restricted to be fixed over time. Our first contribution is an impossibility result that implies that the assumption that self-loop edges have no delays (as assumed in Hitron and Parter) is indeed necessary. Interestingly, in real biological networks self-loop edges (a.k.a. autapse) are indeed free of delays, and the latter has been noted by neuroscientists to be crucial for network synchronization.

  To capture the computational challenges in this setting, we first consider the implementation of a single NOT gate. This simple function already captures the fundamental difficulties in the asynchronous setting. Our key technical results are space and time upper and lower bounds for the NOT function, our time bounds are *tight*. In the spirit of the distributed synchronizers [Awerbuch and Peleg, FOCS'90] and following [Hitron and Parter, ESA'19], we then provide a general synchronizer machinery. Our construction is very modular and it is based on efficient circuit implementation of threshold gates. The complexity of our scheme is measured by the overhead in the number of neurons and the computation time, both are shown to be polynomial in the largest latency value, and the largest incoming degree Δ of the original network.
- **Node Delays:** We introduce the study of asynchronous communication due to variations in the response rates of the neurons in the network. In real brain networks, the *round duration* varies between different neurons in the network. Our key result is a simulation methodology that allows one to transform the above mentioned synchronized solution under edge delays into a synchronized under node delays while incurring a small overhead w.r.t space and time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases asynchronous communication, asynchronous computation, spiking neurons, synchronizers

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements We are thankful to Roei Tell and Gil Cohen for helpful discussions on Boolean Circuits. We also thank Yoram Moses for referring us to related work on asynchronous digital circuits and discussing the connections between the two settings.

© Y.Hitron, M.paerter, G.Perri;
licensed under Creative Commons License CC-BY
42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:47
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Understanding how the brain works, as a computational device, is a central challenge of modern neuroscience, artificial intelligence, and lately also in theoretical computer science and distributed computing [18, 19, 20, 17, 16, 32, 6, 39, 37]. This line of work usually assumes a simple synchronized model [23, 24] in which neurons fire simultaneously in discrete rounds in response to their neighboring neurons that fired in the previous round. This model, while being very convenient for algorithm design, does not take into account the inherent *asynchronous* nature of neural communication. In the neuroscience literature it has been noted that the asynchronous nature of these networks mostly stems from two independent sources [29]: *edge delays* (known as response latency[1].) [35, 5] and *node delays* (known as refractory period) [34, 3]. In this paper, we aim at understanding the computational cost incurred by such asynchronous communication. The overhead is measured by the overhead in the number of neurons and computation time required to compute a certain function. We believe that understanding the computational power, limitations and the connections between these models go beyond the setting of spiking neurons, and might also be relevant for the theory of digital logic design and circuit computation in general.

The Standard Synchronous Model [23, 24]: Before describing our asynchronous models, we first revise the standard synchronous model formally defined by Maass. In this model, the network evolves in discrete, synchronous *rounds* as a Markov chain where each neuron u in the network is a probabilistic threshold gate with a threshold (or bias) value $b(u)$. In every round t, the firing probability of neuron u only depends on the firing status of its incoming neighbors in the preceding round $t-1$. Formally, the potential $\mathrm{pot}_t(u)$ of neuron u in round t is defined by the weighted sum over its incoming neighbors that fired in round $t-1$. The neuron u fires in round t with probability that depends on the quantity $\mathrm{pot}_t(u) - b(u)$.

1.1 Asynchronous Computation with Bounded Edge Delays

We define an asynchronous model with edge delays bounded[2] by some given integer L. The dynamic of the network $\mathcal{N}$ is specified by a latency function $\ell : V \times V \times \mathbb{N} \to \mathbb{N}_{\leq L}$ interpreted as follows: For every neuron u firing in round τ, its spike reaches its outgoing neighbor v within $\ell(u, v, \tau)$ rounds where $\ell(u, v, \tau) \in [1, L]$ might be chosen adversarially for every $u \neq v$, and every round τ. The network solution $\mathcal{N}$ should then output the desired solution for *any* adversarial choice of the latency function ℓ. Setting $L = 1$ yields the standard synchronous model.

Asynchronous computation with edge delays was recently introduced by Hitron and Parter [12]. Their model is similar to ours only that in their model, the edge latencies are required to be *fixed* over time, whereas in our model the adversary is allowed to change it from round to round. The model of Hitron and Parter includes an additional restriction on the adversary (that sets the latency values) by requiring that self-spikes, i.e., of the form $\langle u, u, \tau \rangle$ have no latency and arrive within a single round to their destination. This assumption is justified in [12] by the experimental evidence that self-spikes in brain networks have almost no delays [14]. It is commonly believed in the neuroscience community that this no-delay property of self-edges is in fact essential for network synchronization [33, 21, 40, 8].

[1] Throughout, we use the terms edge-delay and latency interchangeably.
[2] This bound is crucial as will be later implied by our lower bound results that depend on L. E.g., both the computation time and the size of the network in this model *must* depend on L.

In this work, we provide a theoretical support for this hypothesis by showing that without such an assumption, one cannot even implement a single AND gate in this model. This impossibility result holds already in a setting where $L = 2$, the edge latencies are fixed over time (as assumed in [12]), and the computation time and network size are allowed to be unbounded. For this reason, we will mostly consider in this paper *nice* latency functions in which all self-spikes have a latency value of one round.

1.2 Asynchronous Computation with Bounded Node Delays

We next turn to consider an alternative source for asynchronous communication due to variations in the response timing of the individual neurons in the network. In real brain networks, for every neuron u there is a predefined time interval between two consecutive spikes of u. The length of this interval, which we call *round*, varies considerably among different neurons in the network [34, 3]. This poses the challenge of creating a synchronized response at the network level. To account this behavior, we consider a model in which network's evolution proceeds in *seconds*. A second in this context is simply the smallest measurable time unit. For a given integer $T \geq 1$, the dynamics is specified by a node-delay function $t : V \to \mathbb{N}_{\leq T}$ interpreted as follows: the round duration of each neuron v consists of $t(v)$ seconds. Specifically, the i^{th} round of v is defined by the time interval $R_i(v) = [(i-1)t(v)+1, i \cdot t(v)]$ for every $i \geq 1$. The neuron u fires in the second $i \cdot t(v)$ (i.e., at the end of its i^{th} round) only if the total potential due to spikes arriving in the interval $R_i(v)$ is sufficiently large. The network solution $\mathcal{N}$ should then output the desired solution for *any* adversarial choice of the node-delay function t. The input parameter T sets a bound on the differences between the round duration over all neurons in the network. Setting $T = 1$ yields the standard synchronous model.

Observe that the edge and node delay models do not imply one another. In the edge latency model, even though the spikes arrive in adversarially chosen rounds, all neurons in round τ still depend only on the spikes arriving in round τ. Thus, the duration of a round for all the neurons in the network is the same: a single tick (or a second) of the global clock. In contrast, in the node delay model, the adversary selects the round duration for each neuron which has two physical interpretations. First, it determines the time duration over which the potential due to arriving spikes is accumulated. In addition, it also determines the time interval between two consecutive spikes by the given neuron. In one of our most technically involved results, we show a non-trivial reduction between the edge delay and the node delay models, provided that the input network satisfies certain properties.

Finally, we note that this model has several variations which are in fact supported by our simulation results (from edge delays to node delays). In particular, one can consider a more elaborated setting in which the duration of each round per node varies in an adversarial manner *over time*, i.e., the node-delay function in such a model is of the form $t : V \times \mathbb{N}_{\geq 1} \to \mathbb{N}_{\leq T}$ interpreted as follows: the i^{th} round duration on each neuron v consists of $t(v,i)$ seconds. For example, in such a model the i^{th} round of node u can consists of 2 seconds while its $(i+1)^{th}$ round might consist of 100 seconds. In another variation, both edge and node delays are combined and the dynamic is specified by an L-bounded edge latency function $\ell : V \times V \times \mathbb{N}_{\geq 1} \to \mathbb{N}_{\leq L}$ and a T-bounded node-delay function $t : V \times \mathbb{N}_{\geq 1} \to \mathbb{N}_{\leq T}$.

1.3 Synchronizers

In the spirit of Awerbuch and Peleg's synchronizers for distributed networks [2] and the recent work of [12], our primary goal with respect to upper bound results is to provide a general simulation methodology that takes any n-neuron network $\mathcal{N}$ that *solves* the problem in the standard synchronized setting (i.e., in which all spikes arrive within a single round) and transforms it into an “analogous" network $\mathsf{Sync}(\mathcal{N})$ in the edge delay setting while incurring a small overhead in the number of neurons and the computation time (w.r.t the base network $\mathcal{N}$).

For the setting in which the edge latencies are fixed over time and bounded by some integer L, Hitron and Parter [12] showed such a simulation using their efficient constructions for neural timers and counters. Their synchronized network solution $\mathsf{Sync}(\mathcal{N})$ has $O(n + L\log L)$ neurons and $O(rL^3)$ rounds where r is the computation time of the base network $\mathcal{N}$. While being quite efficient in terms of space and time, the synchronizers of [12] are heavily built up on the strong assumption that the latency values of the edges are fixed over time. In this paper, we aimed at understanding the edge latency setting in its most general form, and ask:

> *Can one provide a general synchronization scheme in a setting that allows the latency of the network edges to vary in an adversarial manner in each round?*

A priori it is not so clear if one can compute even basic Boolean functions without assuming that the latency values are fixed. We answer this question in the affirmative by presenting a modular syncronization scheme using a different approach than that taken in [12]. The benefit of this approach is in its modularity. We start by understanding the implementation of a single NOT gate in this model in terms of upper and lower bounds on the space and the time of the computation. We then use this synchronized NOT solution as a building block in the final synchronized network solution. Specifically, the synchronized NOT gates are used to build synchronized circuits (and hence threshold gates) which in turn combined into a whole synchronized network solution. The space and time overheads incurred by our solution are polynomial in L (the bound on the latency) and Δ, the maximal incoming degree in the base network $\mathcal{N}$.

We next turn to consider synchronizers for the node delay model. Our approach is based on showing a *simulation* result that takes any synchronized solution $\mathsf{sync}_E(\mathcal{N}, L)$ obtained by the synchronizer in the L-bounded *edge* latency model, and transforms it into a synchronized solution $\mathsf{sync}_V(\mathcal{N}, T)$ that works in the T-bounded *node* delay model for $L = \Theta(T^2)$.

Remark. It is noteworthy that in contrast to the *distributed* setting of Awerbuch and Peleg [2] where the network size does not depend on the latencies, in the neural setting it is not the case. As our lower bound constructions, both the computation time and the network size must depend (in fact, polynomially) on the largest edge latency. For this reason, for any practical purposes, the study of asynchronous communication, in general, must assume bounded delays.

1.4 Our Results

We study the cost and limitations of asynchronous neural computation in a biologically plausible yet simple model of spiking neural networks. Our main focus is in the edge latency model where the dynamic is specified by an L-bounded function $\ell : V \times V \times \mathbb{N} \to \mathbb{N}_{\leq L}$. The node delay model is concerned only towards the end of the paper (Appendix C), as it is handled via reduction to the edge delay setting. In the first part of the paper, we show several negative results for the L-bounded model. This includes an impossibility results for delay on self-loop edges, as well as size and time lower bound on an implementation of a NOT gate in this model. In the second part, we consider the construction of synchronizers in this generalized setting. We note that these constructions are self-contained and are technically different from [13].

Negative Results: We first show that without assuming a minimal latency value on the self-loop edges, one cannot compute $AND(x, y)$ given two Boolean inputs x and y, even when the edge latency values are fixed over time and the largest latency is $L = 2$.

▶ **Theorem 1** (Impossibility for Arbitrary Latency Functions). *There exists no network that computes $AND(x, y)$ in a setting that allows the adversary to pick latencies in $\{1, 2\}$ for* all *edges in the network.*

The proof goes by showing that for any given candidate network solution $\mathcal{N}$, there exists a bad latency function ℓ under which $\mathcal{N}$ fails to compute $AND(x, y)$. This holds even when the latencies are fixed over time. From that point on, we restrict attention to *nice* latency functions, that assign latency value 1 to the self-loop edges in the network.

▶ **Definition 2.** *A latency function is* nice *if $\ell(u, u, \tau) = 1$ for every $u \in V$ and round τ.*

Our key technical contributions are lower bounds on the network size and the computation time for computing the $NOT(x)$ function in the L-bounded setting. Informally speaking, the $NOT(x)$ function appears to be a "complete" function for the purpose of synchronization in this asynchronous model. Indeed, our $NOT(x)$ implementation captures most of the essence of the L-bounded model. To obtain the final synchronization scheme we mainly glue together synchronized NOT units. For this reason, we spend much attention into understanding the tightness of our constructions by providing nearly matching lower bound results.

▶ **Lemma 3** (Size and Time Lower Bounds for Async. Computation of $NOT(x)$). *Any network that computes $NOT(x)$ in the L-bounded asynchronous setting must use $\Omega(L)$ neurons and $\Omega(L^3)$ time (the time lower bound is tight).*

This should be compared with the size lower bound of $\Omega(\log L)$ shown by [13] for their simplified asynchronous setting.

Positive Results: Our end result is a synchronizer that given any network $\mathcal{N}$ in the standard synchronous setting and an integer L, computes a network $\mathsf{sync}_E(\mathcal{N}, L)$ that performs the "same" computation as $\mathcal{N}$ in the L-bounded edge delay setting.

▶ **Theorem 4** (Synchronizers for Edge Delays). *There exists a synchronizer that given a network $\mathcal{N}$ with n neurons, maximum in-degree Δ, and maximum edge latency L, constructs a network $\mathsf{sync}_E(\mathcal{N}, L)$ that has an "analogous" execution in the L-bounded edge-delay setting with a total number of $\widetilde{O}(L^4 \cdot \mathrm{poly}(\Delta) \cdot n)$ neurons and a time overhead[a] of $\widetilde{O}(L^5 \cdot \log \Delta)$.*

[a] The $\widetilde{O}$ hides a factor of $\mathrm{poly}(\log(n \cdot r))$, where r is the number of simulation rounds.

Although the construction is inspired by the work of Awerbuch and Peleg [2], the implementation is very different as our neurons, unlike processors in a distributed network, are memoryless. Thus, they cannot aggregate the incoming messages as in [2]. Our construction is also different than that of [13], as the latter crucially depends on having fixed latencies over time.

For the node delay model, in Appendix C we show that given a network $\mathcal{N}$ in the standard synchronous setting and an integer T, one can compute an analogous network $\mathsf{sync}_V(\mathcal{N}, T)$ in the node-delay model with bounded node delay T by taking the following approach. Apply the algorithm of Theorem 4 with $\mathcal{N}$ and $L = \Theta(T^2)$. This results with a network $\mathsf{sync}_E(\mathcal{N}, L)$ that performs the same computation as $\mathcal{N}$ in the L-bounded edge delay model. The desired network $\mathsf{sync}_V(\mathcal{N}, T)$ is then obtained by multiplying the edge weights of a carefully defined edge *subset* in $\mathsf{sync}_E(\mathcal{N}, L)$ by a factor of T. The quite delicate analysis then shows that the network $\mathsf{sync}_V(\mathcal{N}, T)$ indeed simulates the original network $\mathcal{N}$ upon any selection of the node-delay function $t : V \to \mathbb{N}_{\leq T}$. By setting $L = O(T^2)$ in Theorem 4, we show the following for the node delay model:

▶ **Theorem 5** (Synchronizers for Node Delays)**.** *There exists a synchronizer that given a network $\mathcal{N}$ with n neurons, maximum in-degree Δ, and maximum node delay T, constructs a network $\mathsf{sync}_V(\mathcal{N}, T)$ that has an "analogous" execution in the T-bounded node-delay setting with a total number of $\widetilde{O}(T^8 \cdot \mathrm{poly}(\Delta) \cdot n)$ neurons and a time overhead of $\widetilde{O}(T^{10} \cdot \log \Delta)$.*

We note that our preference to take a modular approach rather then an optimized one inevitably leads to suboptimal space and time bounds in both Theorems 4 and 5. For example, Theorem 5 is shown via a simulation result, which further deepens our understanding of the connections between these models. We believe that by employing a more direct approach for building synchronizers in the node-delay model, one should get a considerably improved dependency in the delay bound T.

1.5 Our Approach in a Nutshell

We next provide the high level ideas for the key contributions. Throughout, unless stated otherwise we consider the edge delay model where the dynamics is specified by an arbitrary latency function.

Size and Time Lower Bound for Computing $NOT(x)$. A network $\mathcal{N}$ with input neuron x and an output neuron z computes the function $NOT(x)$ in the asynchronous setting if the following holds: when $x = 0$, the output z fires in at *least* one round regardless of the latency function; and when $x = 1$ the output z never fires for any latency function. To show a size lower bound of $\Omega(L)$ we take the following approach. First, we reduce any network $\mathcal{N}$ that computes $NOT(x)$ into a simpler and yet not larger network $\mathcal{N}_{simple}$. In the latter network the only inhibitor is the input x which also has a self-loop of large positive weight, and outgoing edges of very large negative weight to all the excitatory neurons in $\mathcal{N}$. The second part of the proof shows a lower bound for $\mathcal{N}_{simple}$ using its specialized structure. We will assume towards a contradiction that the in-degree of each neuron in $\mathcal{N}_{simple}$ is less than L and exhibit two conflicting latency functions ℓ_0, ℓ_1 that satisfy the following. If $\mathcal{N}_{simple}$ computes $NOT(x)$ with ℓ_0 and with $x = 0$, then it must fail to compute $NOT(x)$ with the function ℓ_1 and $x = 1$. To compute these latency functions, we partition the simulation into blocks $T_0, \ldots$, each containing L rounds. In each phase i, we set the latency values for all the edges and all the rounds in block T_i. Throughout, we will keep the invariant that there exists *no* neuron that fires when $x = 0$ and with ℓ_0, but does not fire when $x = 1$ and with ℓ_1. By the correctness of the network, the output neuron must fire at least once when $x = 0$, thus leading to the contradiction. In the very high level, the fact that the in-degree of each vertex is small is used in order to spread over the at most L incoming spikes of each neuron u in a balanced manner over the L rounds of the block. This will prevent the firing of a neuron z when $x = 0$. Our lower bound is complemented by an upper bound of $O(L^2)$ as described next.

The Generalized Synchronization Scheme. The scheme is based on gradual steps.

Step I: Synchronization of NOT and OR Gates. We start by considering the asynchronous computation of simple Boolean functions $NOT(x)$ and $OR(x_1, \ldots, x_\ell)$ with a small number of neurons. The key challenge is in implementing the NOT gate. When $x = 1$, the output gate is required not to fire (i.e., output 0) throughout the *entire* execution. In contrast, when $x = 0$ the output gate should fire at least once during the execution. The construction is combinatorial and uses a similar logic to the lower bound arguments. It contains a collection of $L + 1$ neurons with outgoing edges to the output z. The above mentioned lower bound result shows that the incoming degree of z must be at least $L - 2$.

Step II: Synchronization of a Boolean Circuit. Any Boolean circuit $\mathcal{A}$ can be implemented by NOT and OR gates. To simulate the computation of $\mathcal{A}$ in the asynchronous setting, we replace each gate g_i in $\mathcal{A}$ by its synchronized implementation $\mathsf{Sync}(g_i)$ constructed in Step (I). For a gate g_i in layer j with incoming gates $g_{i,1}, \ldots, g_{i,k}$, the input to the sub-network $\mathsf{Sync}(g_i)$ are the output neurons of the sub-networks $\mathsf{Sync}(g_{i,1}), \ldots, \mathsf{Sync}(g_{i,k})$. The synchronization between the layers of the circuit is governed by a directed chain of $O(dL^3)$ neurons. The head of the chain fires in the first round of the simulation and activates the network. The sub-networks $\mathsf{Sync}(g_i)$ of gates g_i in layer j are activated by the $\Theta(j \cdot L^3)$ neuron in this chain. These parameters are set so that we can be sure that the modules of layer j are activated, only *after* the spikes from the output neurons of the previous layer have reached the input of this layer. Overall the synchronized transformed network $\mathsf{Sync}(\mathcal{A})$ has $O(d \cdot L^3 + m \cdot L^2)$ neurons, where d is the depth of $\mathcal{A}$ and m is the number of gates. The overtime in the computation is $O(d \cdot L^4)$ rounds.

Step III: Synchronization of a Single (Probabilistic) Threshold Gate. To synchronize a single deterministic threshold gate, we use the fact that a threshold gate with incoming degree Δ can be implemented by a Boolean circuit with $\mathrm{poly}(\Delta)$ neurons and depth $O(\log \Delta)$. This allows us to use the synchronized construction of the previous step. Turning to probabilistic threshold gates, here it is much less clear how to implement such a gate by a Boolean circuit. We take the following approach. First, we use the fact from [20] that a spiking neuron[3] u with bias $b(u)$ is equivalent to a *deterministic* neuron u' whose bias is sampled from the Logistic distribution with mean $b(u)$. Therefore our key challenge is in sampling a value from a given Logistic distribution. To do that, we use a collection of k (input-less) spiking neurons each fires independently with probability half. These neurons provide us the random bits for this process of sampling. In fact, these fair coins tosses allows one to sample a value *almost* uniformly at random in the range $[0, 2^k]$. We will then use the method of inverse transform sampling to convert this almost-uniform sampled value to a value that is sampled from the Logistic distribution up to a small error in the sampling. Using Taylor expansion of the natural log function, we implement this Uniform to Logistic transformation by a collection of simple arithmetic operations applied on a collection of Boolean neurons. The total error in our sampling is set to be small enough so that the output distribution of the Boolean circuit is almost indistinguishable from that of the probabilistic threshold gate.

[3] The probabilistic threshold gates of SNN.

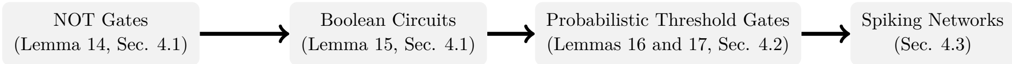

Figure 1 A road-map for synchronizing spiking neural networks.

Grand Finale: Synchronization of a Spiking Neural Network. Finally, given an SNN network $\mathcal{N}$ of (probabilistic) threshold gates the synchronized network $\mathsf{sync}(\mathcal{N})$ is obtained as follows. Each threshold gate g_i in $\mathcal{N}$ is replaced by its synchronized implementation $\mathsf{Sync}(g_i)$. The key challenge is in synchronizing these modules so that every neuron v in $\mathcal{N}$ (i.e., not only the output neuron) has an equivalent neuron v' in $\mathsf{sync}(\mathcal{N})$ that simulates v for any possible latency function throughout the entire simulation. See Fig. 1 for an illustration.

From Edge Delays to Node Delays. Given a network $\mathcal{N}$ to be simulated and an integer T, our goal is to build a synchronizer $\mathsf{sync}_V(\mathcal{N}, T)$ that *simulates* $\mathcal{N}$ in the T-bounded node-delay model. To do that, we first compute a network $\mathsf{sync}_E(\mathcal{N}, L)$ that simulates $\mathcal{N}$ in the L-bounded edge-delay model for $L = \Theta(T^2)$. Then the output network $\mathsf{sync}_V(\mathcal{N}, T)$ is obtained by dividing some of the edge weights in $\mathsf{sync}_E(\mathcal{N}, L)$ by a factor of T. The correctness argument is based on showing that for every node-delay function $t : V \rightarrow \mathbb{N}_{\leq T}$, there exists an edge-delay function $\ell : V \times V \times \mathbb{N}_{\geq 0} \rightarrow \mathbb{N}_{\leq L}$ such that the execution of the network $\mathsf{sync}_E(\mathcal{N}, L)$ with the edge-delay function ℓ (in the edge-delay model) is *similar* to the execution of the network $\mathsf{sync}_V(\mathcal{N}, T)$ with the node-delay function t (in the node-delay model). Since the network $\mathsf{sync}_E(\mathcal{N}, L)$ simulates the original network $\mathcal{N}$ for any edge-delay function, it will imply that the network $\mathsf{sync}_V(\mathcal{N}, T)$ simulates the original network $\mathcal{N}$ for any node-delay function as desired.

Additional Related Work. Asynchronized communication in spiking neural networks has been studied in several settings. Maass [22, 25] considered a quite elaborated model for deterministic neural networks with *arbitrary* response *functions* for the edges, and a vector firing times for all neuron. The approach of [22] mostly concerned the computational power of this model upon choosing the best parameters for the network. I.e., showing feasibility results for various functions. In contrast, in this work our goal is to *bound* the computation time and the network size under this asynchronous setting. Khun et al. [15] studied the asynchronous dynamics under the stochastic model of DeVille and Peskin [7].

Turning to the setting of logical circuits, there is a long line of work on the asynchronous setting under various model assumptions [1, 11, 36, 4] that do not quite fit the memory-less setting of spiking neurons. A more related work to our setting is by Martin, Manohar and Moses [28, 26, 27] who studied the computational power of asynchronous digital circuits. In particular, they characterize the necessary and sufficient conditions for a valid operation of a given circuit in the asynchronous setting. For example, they showed that if all edges and nodes suffer from an unbounded delay then the computational power of the circuit must be very limited. The focus in our work is quite different. Instead of studying the computational power of the asynchronous setting, we bound the computational overhead for solving concrete problems.

2 The Synchronous and Asynchronous SNN Models

A deterministic neuron u is modeled by a *deterministic* threshold gate. Letting $b(u)$ to be the threshold value of u, then u outputs 1 if the weighted sum of its incoming neighbors exceeds $b(u)$. A *spiking neuron* is modeled by a probabilistic threshold gate which fires with a sigmoidal probability that depends on the difference between its weighted incoming sum and $b(u)$.

Neural Network Definition. A *Neural Network* (NN) $\mathcal{N} = \langle X, Z, Y, w, b\rangle$ consists of n input neurons $X = \{x_1, \ldots, x_n\}$, m output neurons $Y = \{y_1, \ldots, y_m\}$, and k auxiliary neurons $Z = \{z_1, ..., z_k\}$. In spiking neural network (SNN), the neurons can be either deterministic threshold gates or probabilistic threshold gates. The directed weighted synaptic connections between $V = X \cup Z \cup Y$ are described by the weight function $w : V \times V \to \mathbb{R}$. A weight $w(u, v) = 0$ indicates that a connection is not present between neurons u and v. Finally, for any neuron v, the value $b(v) \in \mathbb{R}$ is the threshold value (activation bias). The in-degree of every input neuron x_i is zero, i.e., $w(u, x) = 0$ for all $u \in V$ and $x \in X$. Additionally, each neuron is either inhibitory or excitatory: if v is inhibitory, then $w(v, u) \leq 0$ and if v is excitatory, then $w(v, u) \geq 0$ for every u. This restriction arises from the biological structure of the neurons.

Network Dynamics in the Synchronous Setting. The network evolves in discrete, synchronous rounds as a Markov chain. The firing probability of every neuron in round τ depends on the firing status of its neighbors in round $\tau - 1$, via a standard sigmoid function, with details given below. For each neuron u, and each round $\tau \geq 0$, let $\sigma_\tau(u) = 1$ if u fires (i.e., generates a spike) in round τ. Let $\sigma_0(u)$ denote the initial firing state of the neuron. The firing state of each input neuron x_j in each round is the input to the network. For each non-input neuron u and every round $\tau \geq 1$, let $\mathrm{pot}(u, \tau)$ denote the membrane potential at round τ and $p(u, \tau)$ denote the firing probability ($\Pr[\sigma_\tau(u) = 1]$), calculated as $\mathrm{pot}(u, \tau) = \sum_{v \in V} w(v, u) \cdot \sigma_{\tau-1}(v) - b(u)$ and $p(u, \tau) = \frac{1}{1+e^{-\mathrm{pot}(u,\tau)/\lambda}}$ where $\lambda > 0$ is a *temperature parameter* which determines the steepness of the sigmoid. Clearly, λ does not affect the computational power of the network, thus we set $\lambda = 1$.

2.1 Network Dynamics in the Edge Delay Setting

The dynamic of the network is governed by a latency function $\ell : V \times V \times \mathbb{N} \to \mathbb{N}$ interpreted as follows. For every directed edge $e = (u, v)$ and round τ, a spike generated by u in round τ arrived at v after $\ell(u, v, \tau)$ rounds. In the synchronous setting, $\ell(u, v, \tau) = 1$ for every u, v, τ.

For every neuron v and round τ, let $A(u, \tau) = \{(v, \tau')|\ v \in V,\ \tau' + \ell(v, u, \tau') = \tau\}$ denote all the spike events that if occur, arrive to u at round τ. The state of u in round τ is given by:

$$\mathrm{pot}(u, \tau) = \sum_{(v,\tau')\in A(v,\tau)} w(v, u) \cdot \sigma_{\tau'}(v) \ - \ b(u) \ \text{ and } \ \sigma_\tau(u) = 1 \text{ iff } \mathrm{pot}(v, \tau) \geq 0\ . \qquad (1)$$

If u is a probabilistic threshold gate then it fires with probability $p(u, \tau) = \frac{1}{1+e^{-\mathrm{pot}(u,\tau)}}$. When $\ell(u, v, \tau) = \ell(u, v, \tau')$ for every u, v and $\tau' \neq \tau$, we may omit τ and write $\ell(u, v)$.

▶ **Definition 6** (The L-bounded Edge-Delay Setting). *Given is a network $\mathcal{N}$ and an integer L. It is assumed the network contains a special neuron, the* starter*, that fires in the first round of the simulation. The dynamic is determined by a latency function ℓ. This function ℓ can be chosen arbitrarily among all L-bounded nice functions.*

▶ **Definition 7** (Computation of a Boolean Function in the L-bounded Edge-Delay Setting)**.** *Let $f : \{0,1\}^n \to \{0,1\}^k$ be a Boolean function. A network $\mathcal{N}$ with n input neurons $x_1, \ldots, x_n$ and k output neurons $z_1, \ldots, z_k$* computes f *in this setting if for every nice L-bounded function ℓ and for every fixed possible assignment to the input neurons $b_1, \ldots, b_n$ the following holds: (i) If $f_i(b_1, \ldots, b_n) = 1$, then there exists a round in which z_i fires, where $f_i(\cdot)$ is the i^{th} bit in the output of f. (ii) If $f_i(b_1, \ldots, b_n) = 0$ then z_i does not fire throughout the entire execution. Furthermore, the network $\mathcal{N}$ computes the function f* in r rounds *if $\mathcal{N}$ computes f, and for every nice L-bounded function ℓ, input bits $b_1, \ldots, b_n$ and index i such that $f_i(b_1, \ldots, b_n) = 1$, z_i fires in some round $\tau \leq r$.*

Note that by this definition, for a network $\mathcal{N}$ that computes a Boolean function f within r rounds, to evaluate the output of the function it is sufficient to inspect the state of the output bits over the first r rounds of the network's simulation. Furthermore, as edge delays are allowed to be chosen in an adversarial manner, one cannot hope for having all output neurons to fire in the exact same around. One mechanism that we use to keep the output neurons fire simultaneously is by using self-loop edges whose latency values is fixed to be 1.

Synchronizers. A synchronizer ν is an algorithm that gets as input a network $\mathcal{N}$ and outputs a network $\mathcal{N}' = \mathsf{sync}(\mathcal{N})$ that contains all the neurons of $\mathcal{N}$, plus additional auxiliary neurons. One of the auxiliary neurons in $\mathcal{N}'$ is a *starter* neuron that fires in the *first* round of the simulation. The network $\mathcal{N}'$ works in the asynchronous setting and should have *similar execution* to $\mathcal{N}$ in the sense that for every neuron $v \in V(\mathcal{N})$, the firing pattern of v in the asynchronous network should be similar to the one in the synchronous network. The output network $\mathcal{N}'$ simulates each round of the network $\mathcal{N}$ by a *phase*.

▶ **Definition 8** (Phases)**.** *We partition the execution of $\mathcal{N}'$ into phases $1, 2, \ldots$, using a function $r : V(\mathcal{N}) \times \mathbb{N} \to \mathbb{N}$ that defines the beginning of phase p, i.e. the p^{th} phase is the round interval $[r(v,p), r(v,p+1))$.*

▶ **Definition 9** (Similar Executions (Deterministic Networks))**.** *The* synchronous *execution* Π *of a deterministic network $\mathcal{N}$ is specified by a list of states $\Pi = \{\sigma_1, \ldots, \}$ where each σ_i is a binary vector describing the firing status of the neurons in round i. The* asynchronous *execution of the network $\mathcal{N}' = \mathsf{sync}_E(\mathcal{N}, L)$ with a latency function ℓ denoted by $\Pi'(\ell)$ is defined analogously only when applying the asynchronous dynamic of Eq. (1). The execution $\Pi'(\ell)$ is divided into phases according the a function $r : V(\mathcal{N}) \times \mathbb{N} \to \mathbb{N}$.*

*The network $\mathcal{N}$ and the pair $\langle \mathcal{N}', \ell \rangle$ have a **similar execution** if $V(\mathcal{N}) \subseteq V(\mathcal{N}')$, and in addition, a neuron $v \in V(\mathcal{N})$ fires in round p in the execution Π iff v fires during phase p in $\Pi'(\ell)$. The networks $\mathcal{N}$ and $\mathcal{N}'$ are **similar** if $\mathcal{N}$ and $\langle \mathcal{N}', \ell \rangle$ have a similar execution for every nice latency function ℓ.*

Note that specifically, if a synchronous network $\mathcal{N}$ computes a Boolean function f by round r and $\mathcal{N}$ and $\mathcal{N}'$ are similar, then $\mathcal{N}'$ computes f by phase r. Therefore, if we know that each phase is of at most q rounds, we get that $\mathcal{N}'$ computes f in $r \cdot q$ rounds.

Finally, we note that the extension for randomized networks with probabilistic gates is quite straightforward if one simply fixed the random coins used by the neurons over the simulation. That is, to be able to faithfully compare the simulation of two random networks, one has to fix the random coins of both of the simulations to be the same. For this reason, given an input randomized network $\mathcal{N}$ in the synchronized model, we maintain all the random coins generated by the neurons of the network over the simulation. These random coins are then fed to the network $\mathcal{N}'$ (i.e., obtained by applying our synchronizers). Since we compare two randomized networks that use the same set of random coins, we can treat these networks as deterministic. In Appendix C we provide the analogous definitions for the T-bounded node-delay model. Throughout the main paper, we consider only the edge-delay model and to avoid cumbersome notation that synchronized network solutions for this model are denoted by $\mathsf{sync}(\mathcal{N})$ (rather than $\mathsf{sync}_E(\mathcal{N}, L)$).

3 Negative Results

Impossibility Result for Arbitrary Latency Functions. We start by considering Theorem 1, and show that if the latency values are allowed to be set in an adversarial manner in $\{1, 2\}$, then there exists no network that computes the AND of two Boolean inputs. In Appendix A, we show:

▶ **Lemma 10.** *Given input neurons x, y and an output neuron z, there is no network computing $AND(x, y)$ under every latency function $\ell : V \times V \to \{1, 2\}$.*

In the high level, we show that one can set the latency values such that all the spikes that *depend* on the value of x (resp., y) arrive at odd (resp., even) rounds. Therefore, at any round, there is no neuron that fires as a function of *both* x and y.

Size and Time Lower Bound

In this section we show the proof for Lemma 3. Here we focus on the size lower bound although the high level proof strategy for the time lower bound is quite similar. The time lower bound is presented in Appendix A.1. Our proof strategy is as follows. First we reduce any network $\mathcal{N}$ that computes $NOT(x)$ in the asynchronous setting, to a network $\mathcal{N}_{simple}$ with a simpler structure that makes it easier to make arguments on it. The second part of the argument shows the lower bound for simple networks. All missing proofs of this section are in Appendix A.

▶ **Definition 11** (Strong Neurons and Simple Networks). *A neuron u is* strong *in a given network if $w(u, u) \geq b(u)$, and otherwise it is* weak*. Note that specifically, an excitatory neuron u with $b(u) \leq 0$ is strong. Given a single input neuron x, we say that a network $\mathcal{N}$ is* simple *if the following hold: (i) x is a strong neuron and has an outgoing edge of infinite* negative *weights to all other neurons in the network; and (ii) all other neurons are excitatory.*

We note that the simple network is not a *legally defined* neural network: the input neuron has an incoming edge (self-loop), and it is an inhibitor with a positive self-loop. However, this network definition is only for the sake of the analysis and as such, it is not restricted to follow any rule.

Reduction to Simple Networks. Given a network $\mathcal{N}$ with an input neuron x, define $\mathcal{N}_{simple}$ as follows. Exclude all the inhibitory neurons from $\mathcal{N}_{simple}$ and take all edges between excitatory neurons to be as in $\mathcal{N}$. Then, add a self-loop of infinite weight to the input neuron x, and connect it to every neuron with infinite[4] negative weight.

[4] By infinite we mean large enough so that when the spike by x arrives at some neuron v, v would not fire.

▶ **Lemma 12.** *If $\mathcal{N}$ computes $NOT(x)$ within r rounds starting with the initial state $\bar{\sigma}$, then also $\mathcal{N}_{simple}$ computes it within r rounds, when starting with the initial states as in $\bar{\sigma}$ restricted to the vertices of $\mathcal{N}_{simple}$.*

The proof goes by claiming that for any latency function ℓ_{simple} for $\mathcal{N}_{simple}$, we can show the existence of a latency function ℓ for $\mathcal{N}$ whose performance is only *worse* than that of $\mathcal{N}_{simple}$ with ℓ_{simple}. That is, when $x = 0$ (resp., $x = 1$) then the potential of all neurons in $\mathcal{N}_{simple}, \ell_{simple}$ is not decreased (resp. increased) when compared to $\mathcal{N}, \ell$. Since the network $\mathcal{N}$ computes $NOT(x)$ with the latency function ℓ within r rounds, we conclude that also $\mathcal{N}_{simple}$ computes it with the latency function ℓ_{simple} within at most r rounds.

Fix a $NOT(x)$ network $\mathcal{N}$. For an integer r, a latency function ℓ for $\mathcal{N}$ is r-*good* with the initial configuration $\bar{\sigma}$, if the network computes $NOT(x)$ within r rounds. I.e., when $x = 0$, the output of $\mathcal{N}$ fires in some round $\tau \leq r$, and when $x = 1$ it never fires when all latencies are given based on ℓ. If ℓ is r-good for some integer r we say it is *good*, and otherwise the latency function is *bad* (the network fails to compute $NOT(x)$). Note that in order for a network to compute $NOT(x)$ within r rounds, it is required that any latency function is r-good for a fixed initial configuration.

Lower Bound for Simple Networks. Assume towards contradiction that there exists a *simple* network $\mathcal{N} = \mathcal{N}_{simple}$ with maximum in-degree $\Delta_{in} < L - 2$ that computes $NOT(x)$. I.e., there exists an initial configuration $\bar{\sigma}$ for all neurons but x such that every latency function ℓ is good for $\langle \mathcal{N}, \bar{\sigma} \rangle$. In what follows, we define two *conflicting* latency functions ℓ_0 and ℓ_1, such that if ℓ_0 is good when the initial state of x is 0, then it implies that ℓ_1 is bad when the initial state of x is 1.

Defining the Latency Functions ℓ_0 and ℓ_1. Recall that for every $b \in \{0, 1\}$, $\bar{\sigma}_b = [b, \bar{\sigma}]$ is the initial state vector where x has the initial state b and the initial states of all other neurons is specified by the vector $\bar{\sigma}$. The construction of ℓ_0, ℓ_1 is inductive. To avoid cumbersome notation, we start the simulation in round -1 rather than in round 0. For this first round -1, let $\ell_b(v, u, -1) = 1$ for $v \neq x$, and $\ell_b(x, u, -1) = L$ for every u and $b \in \{0, 1\}$. Thus, the positive spikes (by any $v \neq x$) fired in round -1 arrive in round 0, and the negative spikes of x arrive in round $L - 1$.

To define the latency of the edges in the remaining rounds $\tau \geq 0$, we partition them into blocks, each of size L rounds where the i^{th} block is $T_i = [iL, iL + (L-1)]$ for every $i \geq 0$. We continue in steps $i = 1, \ldots$ where in step i, the latency values of $\ell_0(e, \tau), \ell_1(e, \tau)$ are defined for every edge e in the network $\mathcal{N}$, and for every round $\tau \in T_i$. For every $b \in \{0, 1\}$ and a block T_i, define $A_{i,b}$ as the set of neurons that fire (hence *active*) in the first round of T_i when executing $\mathcal{N}$ with the initial configuration $\bar{\sigma}_b$, and the latency function ℓ_b. Throughout the process of defining the latency functions, we maintain these invariants at the beginning of step i:

- (I1) All the positive spikes generated at any round before the interval T_i arrive to their destination by the first round of T_i. Furthermore, all negative spikes generated at any round before the interval T_i arrive to their destination either by the first round of T_i or on the last round of T_i, namely, round $iL + (L - 1)$.
- (I2) $A_{i,0} \setminus \bigcup_{i' \leq i} A_{i',1} = \emptyset$.

We define the latency for the rounds in T_i, and then show that the invariants are maintained.

Defining the latency function ℓ_1 for T_i. For every self-loop edge e and every $\tau \in T_i$, let $\ell_1(e, \tau) = 1$. For every edge $e = (x, u)$ where $u \neq x$, and $\tau \in T_i \setminus \{iL + (L - 1)\}$, let $\ell_1(e, \tau) = (iL + (L - 1)) - \tau$, i.e., the spike of x arrives u in the last round of the interval

T_i. For $e = (x, u)$ and $\tau = iL + (L - 1)$, let $\ell_1(e, \tau) = L$ so that the spike arrives in the last round of the next block, i.e., round $(i + 1)L + (L - 1)$. For every other edge $e = (v, u)$ with $v \neq x$, let $\ell_1(e, \tau) = (i + 1)L - \tau$, i.e., the spike arrives at the first *first* round of the next block T_{i+1}.

Defining the latency function ℓ_0 for T_i. As for ℓ_1, for a self-loop edge e, we set $\ell_0(e, \tau) = 1$. For an edge $e = (x, u)$ we set $\ell_0(e, \tau)$ arbitrarily (since $x = 0$, those values are meaningless). We now fix a neuron u, and set the latency values of all its incoming edges (v, u). Since we have already defined the latency values of all edges up to block T_i, at the beginning of step i, the sets $A_{i,0}, A_{i,1}$ can be computed. Let $g_1, \ldots, g_\omega$ be the weak incoming neighbors of u in $A_{i,0}$, and $h_1, \ldots, h_s$ be the strong incoming neighbors of u in $A_{i,0}$. We Consider two cases. The neuron u is said to have a *dominant* neighbor if it has a neighbor with a *sufficiently* large incoming weight, where the precise weight threshold depend on whether the incoming neighbor is weak or strong. Specifically, it has a dominant neighbor if it has either a weak neighbor g_j with $w(g_j, u) \geq b(u)$, or a strong neighbor h_j with $w(h_j, u) \geq b(u)/(L - 1)$.

- **Case 1: u has a dominant neighbor.** Let $\ell_0(e, \tau) = (i + 1)L - \tau$ for every incoming edge $e = (v, u)$. That is, we schedule all the incoming spikes of u in this block to arrive at u in the *first* round of the next block T_{i+1}.
- **Case 2: u has no dominant neighbor.** Since $\deg(u) < L-2$, we have that $\omega+s < L-2$, and in particular $\omega \leq L - 2$. For each weak neuron g_j, set $\ell_0(g_j, u, iL) = j + 1$. That is, the spike from g_j in round iL is scheduled to arrive at u in round $iL + (j + 1)$. For each strong neighbor h_j, we split all the spikes generated by h_j during the $\omega + 1$ rounds $iL, \ldots, iL + \omega$ in a balanced manner over $L - (\omega + 1)$ rounds. Specifically, we set the latency values of the at most $\omega + 1$ spikes by h_j during the rounds $iL, \ldots, iL + \omega$ such that in each round $\tau \in [iL + (\omega + 2), (i + 1)L]$, u receives at most $(\omega + 1)/(L - (\omega + 1))$ spikes from h_j[5]. For every $\tau \in [iL + (\omega + 1), iL + (L - 1)]$, let $\ell_0(h_j, u, \tau) = 1$, i.e., the spike arrives one round later. The latency of all the remaining edges e and rounds τ in T_i is set to $\ell_0(e, \tau) = (i + 1)L - \tau$, so that it arrives in round $(i + 1)L$.

In AppendixA.1.2, we prove that the invariants hold by induction on the number of rounds. Since the output z is required to fire when $x = 0$ but must not fire when $x = 1$, we get the desired contradiction. In Appendix A.1, we show the time lower bound of $\Omega(L^3)$ rounds. This bound is tight, and the construction while having a similar high-level ideas is slightly more involved than the size lower bound.

4 Upper Bounds

4.1 Synchronization of Logic Gates and Boolean Circuits

First observe that the simple implementation of an OR-gate works also in the asynchronous setting.

▷ Observation 13 (OR gate). Given input neurons $x_1, \ldots, x_n$ and output neuron z, there exists a deterministic network $\mathsf{OR}_{\mathsf{sync}}$ with *no* auxiliary neurons, that computes the OR gate of $x_1, \ldots, x_n$ using L rounds. I.e, it holds that: (i) If $\sigma_0(x_1) \vee \ldots \vee \sigma_0(x_n) = 0$, then $\sigma_t(z) = 0$ for every t, and (ii) If $\sigma_0(x_1) \vee \ldots \vee \sigma_0(x_n) = 1$, then there exists a round $t \in [1, L]$ such that $\sigma_t(z) = 1$. Moreover, if an input neuron fires in round τ, the output neuron z fires in some round $t \in [\tau + 1, \tau + L]$.

[5] For simplicity, we assume that $(\omega + 1)$ divides $(L - (\omega + 1))$

We next consider the more technically involved setting of synchronizing a NOT gate.

▶ **Lemma 14** (NOT gate). *There is a network* $\mathsf{NOT_{sync}}$ *of size* $O(L^2)$ *with input neuron* x *and output* z*, that computes* $NOT(x)$ *within* $O(L^3)$ *rounds. I.e, it holds that: (i) If* $\sigma_0(x) = 1$*, then* $\sigma_t(z) = 0$ *for every* t*, and (ii) If* $\sigma_0(x) = 0$*, then there exists a round* $t \in [L, \Theta(L^3)]$ *such that* $\sigma_t(z) = 1$.

The following synchronous implementation assumes that the network contains a special starter neuron v^* that fires at the beginning of the simulation, regardless of the input value of x. Later on in Section 4.3, when presenting the complete synchronization scheme, this neuron v^* will receive the starting firing signal from the *global* pulse generator.

Network Description. The network consists of the following components, see Figure 2.

1. A *chain* $C = [c_0 = v^*, \ldots, c_{5L^2}]$ containing $5L^2 + 1$ neurons. The head of the chain is the starter neuron that fires in the first round. For every $i \geq 0$, the neuron c_i has bias $b(c_i) = 1$. Moreover, for every $i \geq 1$ the neuron c_i has an incoming edge from c_{i-1} with weight 1.
2. A *memory neuron* m that remembers the initial state of x. The memory neuron has a positive incoming edge from x, as well as a self-loop both with weight 1 and bias $b(m) = 1$.
3. A *reset* inhibitory neuron r with an edge from m of weight $w(m, r) = 1$, and bias $b(r) = 1$.
4. A collection of $L + 1$ intermediate neurons $v_0, \ldots, v_L$ that are connected to the output neuron z, where each v_i has an incoming edge from the neuron $c_{5 \cdot iL} \in C$ with weight $w(c_{i \cdot 5L}, v_i) = 1$, a self-loop of weight 1 and bias $b(v_i) = 1$. In addition, each v_i has a negative incoming edge from the reset neuron r with weight $w(r, v_i) = -\infty$. Finally, each v_i has an edge to z with weight $w(v_i, z) = 1$ and bias $b(z) = L + 1$.

The correctness of the construction and the proof of Lemma 14 are deferred to Appendix B.1.

Synchronization of a Boolean Circuit. Given the synchronized sub-networks of Observation 13 and Lemma 14, we now show how to synchronize a Boolean circuit that contains OR and NOT gates.

▶ **Lemma 15.** *Given a Boolean circuit* $\mathcal{A}$ *of OR and NOT gates with* n *inputs,* k *outputs,* m *gates and depth* d*, there exists a deterministic network* $\mathcal{N} = \mathsf{sync}(\mathcal{A})$ *with input neurons* $x_1, \ldots, x_n$*, output neurons* $z_1, \ldots, z_k$ *and* $O(dL^3 + mL^2)$ *auxiliary neurons, that computes* $\mathcal{A}$ *in* $O(dL^4)$ *rounds. I.e., it holds that (i) If* $[\mathcal{A}(\sigma_0(x_1), \ldots, \sigma_0(x_n))]_i = 0$ *then* $\sigma_t(z_i, \mathcal{N}) = 0$ *for every* t*; and (ii) If* $[\mathcal{A}(\sigma_0(x_1), \ldots, \sigma_0(x_n))]_i = 1$*, then*[6] *there exists* $t \in [1, O(dL^4)]$ *such that* $\sigma_t(z_i, \mathcal{N}) = 1$.

In the high-level, the network $\mathcal{N} = \mathsf{sync}(\mathcal{A})$ is obtained by replacing each gate g_i with its synchronized module $\mathsf{Sync}(g_i)$. The input neurons to the gate modules in layer j of $\mathcal{A}$ are the output neurons of the gate modules of layer $j - 1$ in $\mathcal{A}$. The network then contains a chain of length $O(d \cdot L^3)$ to control the synchronization between layers: the modules of layer j are activated only after the modules of the previous layer have completed their computation. See Appendix B.2.

4.2 Synchronization of a Single Threshold Gate

Deterministic Threshold Gate. Given a deterministic threshold gate g with Δ inputs, one can implement g using a Boolean Circuit with $\mathrm{poly}(\Delta)$ gates and depth $O(\log \Delta)$ (see

[6] For a vector of n bits $x \in \{0,1\}^n$, let $[x]_i$ denote the i^{th} bit of x. I.e., if $x = (x_1, \ldots, x_n)$, then $[x]_i = x_i$.

Appendix B.3). Combining with the construction described in Lemma 15 we show the following:

▶ **Lemma 16.** *Given a weighted threshold gate $g = f(x_1, \ldots, x_\Delta)$, there exists a network $\mathcal{N} =$ Sync(g) with Δ input neurons $x_1, \ldots, x_\Delta$, an output neuron z, and $O(\log \Delta \cdot L^3 + \text{poly}(\Delta) \cdot L^2)$ auxiliary neurons that computes f within $O(\log \Delta \cdot L^4)$ rounds. I.e. the output z fires in round $\tau \in [2, O(\log \Delta \cdot L^4)]$ if and only if $f(\sigma_0(x_1), \ldots, \sigma_0(x_\Delta)) = 1$.*

Probabilistic Threshold Gate. We next turn to consider the more challenging setting of probabilistic threshold gates. To synchronize such gates, we first describe how to implement them by using a Boolean *circuit* $\mathcal{A}$ that contains two types of gates: deterministic threshold gates, and input-less gates which outputs 1 with probability 1/2. We hereafter denote the latter gates by uniformly random gates[7]. The output distributions of the probabilistic threshold gate and the output gate of $\mathcal{A}$ will be very close up to a small additive error of $\epsilon \in (0, 1)$. The synchronized probabilistic gate will be obtained by applying the synchronization scheme of Lemma 15 on the circuit $\mathcal{A}$.

Our key result might be of independent interest in the context of Boolean circuits:

▶ **Lemma 17.** *Given a* probabilistic *threshold gate g with Δ inputs, and an error parameter $\epsilon \in (0, 1)$, there exists a Boolean circuit with depth* $\text{poly}(\log \Delta, \log(1/\epsilon))$ *and a total* $\text{poly}(\Delta, \log(1/\epsilon))$ deterministic *gates. In addition, there is a collection of $O(\log(1/\epsilon))$ uniformly random gates (each outputs 1 independently w.p. 1/2), and an output gate g' that approximates g in the following sense. Letting $p(\bar{x}), p'(\bar{x})$ be the probability that g, g' output 1 given input $\bar{x}$, it holds that $|p(\bar{x}) - p'(\bar{x})| \leq \theta(\epsilon)$ for any fixed assignment of input $\bar{x}$.*

Our starting point is the following useful fact from [20]:

▷ Observation 18. Let g_1 be a *probabilistic* gate with an incoming weighted sum W and bias b_1. Let g_2 be a *deterministic* threshold gate with incoming weighted sum W and bias b_2, where b_2 is sampled from the Logistic distribution with mean b_1 and scale 1. Then $\Pr[g_1 = 1] = \Pr[g_2 = 1] = 1/(1 + e^{-(W - b_1)})$.

The observation holds as the cumulative density function of the Logistic distribution is a Sigmoidal function. Since we already know how to implement a deterministic threshold gate using a Boolean circuit, the key challenge is in sampling a value from the Logistic distribution using a small number of uniformly random gates (i.e., fair coins). This is done in two key steps. First, using $O(\log 1/\epsilon)$ uniformly random gates, we sample a value from an ϵ^4-discretization of the uniform distribution [8]. Then, we use the method of inverse transform sampling to sample from a distribution that is $\Theta(\epsilon)$-close (in L_1 norm) to the Sigmoidal distribution. For a value r sampled u.a.r in $[0, 1]$, a sample from the Logistic distribution with mean b and scale 1 is given by $b + \ln(r/(1 - r))$. To compute the expression $b + \ln(r/(1 - r))$ using a Boolean circuit, we approximate the $\ln(x)$ function using the first $O(\log 1/\epsilon)$ terms of the Taylor expansion. The almost-Logistic sample will serve as the bias of a deterministic threshold gate and will be fed to the Boolean circuit of Lemma 16. The full description is given in Appendix B.4.

We can then synchronize the Boolean Circuit as described in Lemma 17.

---

[7] A uniformly random gate is a fair coin, in contrast to probabilistic threshold gate that outputs 1 based on a Sigmoidal distribution.

[8] Our sample is equivalent to sampling a value from the uniform distribution and then rounding it to the closest value of the form $i \cdot \epsilon^4$ for some integer i.

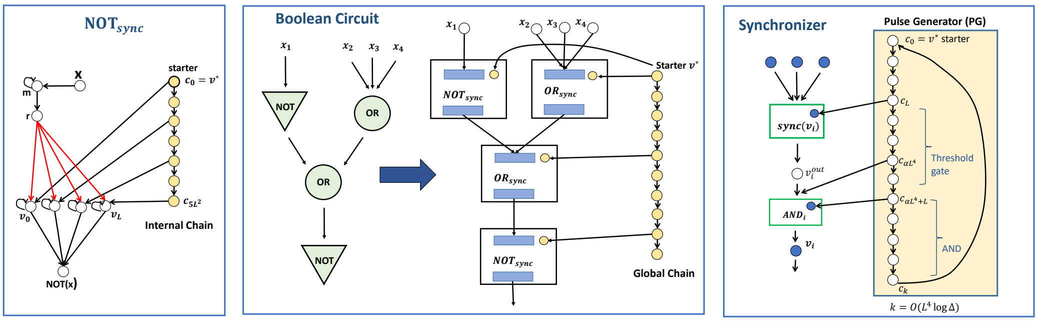

Figure 2 Left: synchronized network of a single NOT gate. Middle: A synchronized network for a Boolean circuit. Right: The transformation of a single neuron v_i in the synchronized network for the given SNN.

▶ **Corollary 19.** *Given a probabilistic threshold gate g with Δ inputs, and an error parameter $\epsilon \in (0, 1)$, there exists a network $\mathcal{N} = \mathsf{Sync}(g)$ with Δ input neurons $x_1, \ldots, x_\Delta$, an output neuron z, and $\mathrm{poly}(\Delta, \log 1/\epsilon) \cdot L^3$ auxiliary neurons such that z* approximates *the gate g within $\mathrm{poly}(\log \Delta, \log 1/\epsilon) \cdot L^4$ rounds in the following sense. For any fixed input $\bar{x}$, with probability at least $1 - \Theta(\epsilon)$, it holds that g outputs 1 iff z fires in some round in* $[1, \mathrm{poly}(\log \Delta, \log 1/\epsilon) \cdot L^4]$.

4.3 The Complete Synchronization Scheme

The complete synchronization scheme and the proof of Theorem 4 are given in Appendix B.5. In the high level, the construction has two parts: a global pulse generator, and a specific adaptation of the given network $\mathcal{N}$ into a network $\mathsf{sync}(\mathcal{N})$, see Figure 2.

The *pulse generator* is implemented by a directed cycle of length $k = \widetilde{O}(L^4 \log \Delta)$. The input layer and output layer in $\mathsf{sync}(\mathcal{N})$ are *exactly* as in $\mathcal{N}$. Let V be the neurons of $\mathcal{N}$. For each auxiliary neuron $v_i \in V$, we add its synchronized sub-network $\mathsf{Sync}(v_i)$ from Lemma 16 and Cor. 19. Recall that each neuron in $\mathcal{N}$ implements either a threshold gate or a probabilistic threshold gate. For each such $v_i \in V$, we also add an AND module AND_i, which receives input from the sub-network $\mathsf{Sync}(v_i)$ and the pulse generator. The neuron v_i is set to be the output neuron of this AND_i module.

References

1 Douglas B Armstrong, Arthur D Friedman, and Premachandran R Menon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Transactions on Computers*, 100(12):1110–1120, 1969.

2 Baruch Awerbuch and David Peleg. Network synchronization with polylogarithmic overhead. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 514–522, 1990.

3 Michael J Berry II and Markus Meister. Refractoriness and neural precision. In *Advances in Neural Information Processing Systems*, pages 110–116, 1998.

4 Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1):1, 2006.

5 Sami Boudkkazi, Edmond Carlier, Norbert Ankri, Olivier Caillard, Pierre Giraud, Laure Fronzaroli-Molinieres, and Dominique Debanne. Release-dependent variations in synaptic latency: a putative code for short-and long-term synaptic dynamics. *Neuron*, 56(6):1048–1060, 2007.

6 Chi-Ning Chou, Kai-Min Chung, and Chi-Jen Lu. On the algorithmic power of spiking neural networks. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 26:1–26:20, 2019.

7 RE Lee DeVille and Charles S Peskin. Synchrony and asynchrony in a fully stochastic neural network. *Bulletin of mathematical biology*, 70(6):1608–1633, 2008.

8 Huawei Fan, Yafeng Wang, Hengtong Wang, Ying-Cheng Lai, and Xingang Wang. Autapses promote synchronization in neuronal networks. *Scientific reports*, 8(1):580, 2018.

9 Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009.

10 Johan Håstad. On the size of weights for threshold gates. *SIAM J. Discrete Math.*, 7(3):484–492, 1994. URL: `https://doi.org/10.1137/S0895480192235878`, `doi:10.1137/S0895480192235878`.

11 Scott Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, 1995.

12 Yael Hitron and Merav Parter. Counting to ten with two fingers: Compressed counting with spiking neurons. *ESA*, 2019.

13 Yael Hitron and Merav Parter. Counting to ten with two fingers: Compressed counting with spiking neurons. *CoRR*, abs/1902.10369, 2019. URL: `http://arxiv.org/abs/1902.10369`, `arXiv:1902.10369`.

14 Kaori Ikeda and John M Bekkers. Autapses. *Current Biology*, 16(9):R308, 2006.

15 Fabian Kuhn, Joel Spencer, Konstantinos Panagiotou, and Angelika Steger. Synchrony and asynchrony in neural networks. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 949–964. SIAM, 2010.

16 Robert A. Legenstein, Wolfgang Maass, Christos H. Papadimitriou, and Santosh Srinivas Vempala. Long term memory and the densest k-subgraph problem. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 57:1–57:15, 2018.

17 Nancy Lynch and Cameron Musco. A basic compositional model for spiking neural networks. *arXiv preprint arXiv:1808.03884*, 2018.

18 Nancy Lynch, Cameron Musco, and Merav Parter. Computational tradeoffs in biological neural networks: Self-stabilizing winner-take-all networks. In *Proceedings of the 8th Conference on Innovations in Theoretical Computer Science (ITCS)*, 2017.

19 Nancy Lynch, Cameron Musco, and Merav Parter. Spiking neural networks: An algorithmic perspective. In *5th Workshop on Biological Distributed Algorithms (BDA 2017)*, July 2017.

20 Nancy A. Lynch, Cameron Musco, and Merav Parter. Neuro-ram unit with applications to similarity testing and compression in spiking neural networks. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 33:1–33:16, 2017.

21 Jun Ma, Xinlin Song, Wuyin Jin, and Chuni Wang. Autapse-induced synchronization in a coupled neuronal network. *Chaos, Solitons & Fractals*, 80:31–38, 2015.

22 Wolfgang Maass. Lower bounds for the computational power of networks of spiking neurons. *Electronic Colloquium on Computational Complexity (ECCC)*, 1(19), 1994. URL: `http://eccc.hpi-web.de/eccc-reports/1994/TR94-019/index.html`.

23 Wolfgang Maass. On the computational power of noisy spiking neurons. In *Advances in Neural Information Processing Systems 8 (NIPS)*, 1996.

24 Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

25 Wolfgang Maass. Paradigms for computing with spiking neurons. In *Models of Neural Networks IV*, pages 373–402. Springer, 2002.

26 Rajit Manohar and Yoram Moses. Analyzing isochronic forks with potential causality. In *21st IEEE International Symposium on Asynchronous Circuits and Systems, ASYNC 2015,* *Mountain View, CA, USA, May 4-6, 2015*, pages 69–76, 2015. URL: https://doi.org/10.1109/ASYNC.2015.19, doi:10.1109/ASYNC.2015.19.

27 Rajit Manohar and Yoram Moses. The eventual c-element theorem for delay-insensitive asynchronous circuits. In *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 102–109. IEEE, 2017.

28 Alain J Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Beauty is our business*, pages 302–311. Springer, 1990.

29 Robert Miller. Time and the brain. *CRC Press*, 2000.

30 Saburo Muroga. *Threshold logic and its applications.* Wiley, 1971.

31 Yu P Ofman. On the algorithmic complexity of discrete functions. In *Doklady Akademii Nauk*, volume 145, pages 48–51. Russian Academy of Sciences, 1962.

32 Christos H. Papadimitriou and Santosh S. Vempala. Random projection in the brain and computation with assemblies of neurons. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 57:1–57:19, 2019. URL: https://doi.org/10.4230/LIPIcs.ITCS.2019.57, doi:10.4230/LIPIcs.ITCS.2019.57.

33 Huixin Qin, Jun Ma, Chunni Wang, and Ying Wu. Autapse-induced spiral wave in network of neurons under noise. *PloS one*, 9(6):e100849, 2014.

34 Alexa Riehle, Sonja Grün, Markus Diesmann, and Ad Aertsen. Spike synchronization and rate modulation differentially involved in motor cortical function. *Science*, 278(5345):1950–1953, 1997.

35 BL Sabatini and WG Regehr. Timing of synaptic transmission. *Annual review of physiology*, 61(1):521–542, 1999.

36 Jens Sparsø. Asynchronous circuit design-a tutorial. In *Chapters 1-8 in" Principles of asynchronous circuit design-A systems Perspective"*. Kluwer Academic Publishers, 2001.

37 Lili Su, Chia-Jung Chang, and Nancy Lynch. Spike-based winner-take-all computation: Fundamental limits and order-optimal circuits. *Neural Computation*, 2019.

38 Christopher S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. Electronic Computers*, 13(1):14–17, 1964. URL: https://doi.org/10.1109/PGEC.1964.263830, doi:10.1109/PGEC.1964.263830.

39 Barbeeba Wang and Nancy Lynch. Integrating temporal information to spatial information in a neural circuit. *arXiv preprint arXiv:1903.01217*, 2019.

40 Ergin Yilmaz, Mahmut Ozer, Veli Baysal, and Matjaž Perc. Autapse-induced multiple coherence resonance in single neurons and neuronal networks. *Scientific Reports*, 6:30914, 2016.

A Missing Proofs for the Negative Results

Impossibility Result, Proof of Lemma 10: Assume towards contradiction there exists a network $\mathcal{N} = (V, \{x, y\}, \{z\}, w, b)$ that computes the AND gate of the initial states x_0, y_0 of the inputs x and y. It is then required that if $x_0 \wedge y_0 = 1$, then there exists a round in which z fires, and if $x_0 \wedge y_0 = 0$ then z is idle throughout the execution. Our goal is to show the existence of a bad assignment of latency values to the edges of $\mathcal{N}$. Such bad assignment exists even if we fix the latency of each edge to the same value throughout the entire execution. We begin with some quick observations.

- The state of a neuron v in round τ, namely, $\sigma_\tau(v)$, is fully determined by the network, the latency function, the input and the initial state σ_0, that is $\sigma_\tau(v) = H(\mathcal{N}, \ell, \sigma_0, x_0, y_0, v, \tau)$ for some function H.
- Given the network $\mathcal{N}$ and a latency function ℓ, the state of a neuron v in round τ is a function of the *previous* states of its incoming neighbors denoted as $u_1 \ldots u_k$:

$$\sigma_\tau(v) = F_v(\sigma_{\tau-\ell(u_1,v)}(u_1), \ldots, \sigma_{\tau-\ell(u_k,v)}(u_k)).$$

▶ **Definition 20.** *For a neuron v and round τ, we say that the state $\sigma_\tau(v)$ is x-*independent *(equivalently y-independent) if its value does not depend on the initial state of x, i.e if*

$$H(\mathcal{N}, \ell, \sigma_0, x_0 = 0, y_0, v, \tau) = H(\mathcal{N}, \ell, \sigma_0, x_0 = 1, y_0, v, \tau) \ .$$

▷ Observation 21. A concatenation of x-independent functions is also x-independent. Specifically, for round τ and neuron v with incoming neighbors $u_1, \ldots u_k$, it holds that if $\sigma_{\tau-\ell(u_1,v)}(u_1), \ldots, \sigma_{\tau-\ell(u_k,v)}(u_k)$ are x-independent then $\sigma_\tau(v)$ is also x-independent.

Given the network $\mathcal{N}$ we set the edge latencies as follows:

$$\ell(u,v) = \begin{cases} 1 & \text{if either } u = x \text{ or } v = x\text{, but not both.} \\ 2 & \text{otherwise.} \end{cases}$$

We next show that for every neuron $v \in V$, its firing state in each round if either x-independent or y-independent. Specifically, the firing state of z in each round does not depend on both x_0 and y_0. This will contradict the assumption that z computes an AND gate of x_0 and y_0.

▷ Claim 22. For every round $\tau \geq 1$ it holds that: (1) For every $v \in V \setminus \{x\}$, the firing state $\sigma_\tau(v)$ is x-independent if τ is even and y-independent if τ is odd. (2) $\sigma_\tau(x)$ is x-independent if τ is odd and y-independent if τ is even.

Proof. By induction on the round τ. For $\tau = 1$, since all outgoing edges from y have latency 2 (except for the edge (y, x), if exists), in round 1 no neuron $v \in V \setminus \{x\}$ received a spike from y and therefore $\sigma_1(v)$ is y-independent. Because the edge from x to itself has latency 2, and the edge from y to x has latency 1, in round 1 the neuron x can receive a signal from y but not from x and therefore also $\sigma_1(x)$ is x-independent. For $\tau = 2$ and $v \neq x$, since the edges from x have latencies 1, and all other edges have latency 2, it holds that

$$\sigma_2(v) = F_v(\sigma_1(x), \sigma_0(y), \sigma_0(u_1), \ldots, \sigma_0(u_k)),$$

where $u_1, \ldots, u_k$ are the neighbors of v in $V \setminus \{x\}$. Because $\sigma_1(x)$ is x-independent, and $\sigma_0(u_i)$ are the initial states, by Observation 21 we can conclude that $\sigma_2(v)$ is x-independent. Next, for the input neuron x, since all its incoming edges (except for the self-loop) have latency 1 it holds that

$$\sigma_2(x) = F_x(\sigma_0(x), \sigma_1(u_1), \ldots, \sigma_1(u_k)).$$

Because $\sigma_1(v)$ is y-independent for all $v \neq x$, we conclude that $\sigma_2(x)$ is y-independent as well.

Assume the claim holds for every round $\tau' < \tau$ and we will show the claim holds for round τ as well. For $v \neq x$ with incoming neighbors $u_1, \ldots u_k$ in $V \setminus \{x\}$, by the definition of the latencies it holds that

$$\sigma_\tau(v) = F_v(\sigma_{\tau-1}(x), \sigma_{\tau-2}(u_1), \ldots, \sigma_{\tau-2}(u_k)).$$

If τ is even, so is $\tau - 2$ and by the induction assumption $\sigma_{\tau-2}(u_i)$ are x-independent. Because $\tau - 1$ is odd, $\sigma_{\tau-1}(x)$ is x-independent. Hence, by Observation 21 we conclude that $\sigma_\tau(v)$ is also x-independent. Similarly, if τ is odd, then by the induction assumption $\sigma_{\tau-2}(u_i)$ and $\sigma_{\tau-1}(x)$ are y-independent, and therefore $\sigma_\tau(v)$ is also y-independent. Then, for the neuron x, it holds that

$$\sigma_\tau(x) = F_x(\sigma_{\tau-2}(x), \sigma_{\tau-1}(u_1), \ldots, \sigma_{\tau-1}(u_k)).$$

If τ is odd, by the induction assumption $\sigma_{\tau-2}(x)$ as well as $\sigma_{\tau-1}(u_1), \ldots, \sigma_{\tau-1}(u_k)$ are x-independent and therefore $\sigma_\tau(x)$ is x-independent. On the other hand, if x is even, then by the induction assumption $\sigma_{\tau-2}(x)$ and $\sigma_{\tau-1}(u_1), \ldots, \sigma_{\tau-1}(u_k)$ are y-independent and therefore $\sigma_\tau(x)$ is also y-independent. ◀

Since in each round the output neuron z is either x-independent or y-independent, this contradicts the assumption and Lemma 10 follows.

A.1 Size Lower Bound for Computing $NOT(x)$

A.1.1 Reduction to Simple Networks, Proof of Lemma 12:

The reduction is based on the following notion of domination between two configurations.

Domination. Given a network $\mathcal{N}$, a latency function ℓ, and a vector of starting states $\bar{\sigma}$ for all neurons but the input x, for $b \in \{0,1\}$ define $\text{pot}_b(u, \tau, \mathcal{N}, \ell, \bar{\sigma})$ as the potential of neuron u in round τ in the simulation of $\mathcal{N}$ with the initial vector state $[b, \bar{\sigma}]$, i.e., with the initial state of x is being b and all other initial states are as in $\bar{\sigma}$. When $\bar{\sigma}$ is clear from the context, we may omit it, and simply write $\text{pot}_b(u, \tau, \mathcal{N}, \ell)$. Given networks $\mathcal{N}_1, \mathcal{N}_2$ with vertices V_1, V_2 and latency functions ℓ_1, ℓ_2 respectively, we say that $\langle \mathcal{N}_1, \bar{\sigma}_1 \rangle$ and $\langle \mathcal{N}_2, \bar{\sigma}_2 \rangle$ are *compatible* if $V_1 \subseteq V_2$ and σ_1 and σ_2 agree on the mutual vertices of V_1, i.e., $\sigma_1(u) = \sigma_2(u)$ for every $u \in V_1$.

In our arguments, we consider a pair of compatible configurations $\langle \mathcal{N}_1, \bar{\sigma}_1 \rangle$ and $\langle \mathcal{N}_2, \bar{\sigma}_2 \rangle$ along with latency functions ℓ_1, ℓ_2 for these configurations. We say that $\langle \mathcal{N}_1, \bar{\sigma}_1, \ell_1 \rangle$ *dominates* $\langle \mathcal{N}_2, \bar{\sigma}_2, \ell_2 \rangle$ if $\langle \mathcal{N}_1, \bar{\sigma}_1 \rangle$ and $\langle \mathcal{N}_2, \bar{\sigma}_2 \rangle$ are compatible and in addition:

- $\text{pot}_1(u, \tau, \mathcal{N}_1, \ell_1, \bar{\sigma}_1) \leq \text{pot}_1(u, \tau, \mathcal{N}_2, \ell_2, \bar{\sigma}_2)$ for every $u \in V_1 \setminus \{x\}$ and $\tau \geq 0$.
- $\text{pot}_0(u, \tau, \mathcal{N}_1, \ell_1, \bar{\sigma}_1) \geq \text{pot}_0(u, \tau, \mathcal{N}_2, \ell_2, \bar{\sigma}_2)$ for every $u \in V_1 \setminus \{x\}$ and $\tau \geq 0$.

Let V, V_{simple} be the vertex sets of $\mathcal{N}$ and $\mathcal{N}_{simple}$ respectively. Let $\bar{\sigma}_{simple}$ be the initial state vector that agrees with $\bar{\sigma}$ on all vertices in V_{simple}. Thus, $\langle \mathcal{N}_{simple}, \bar{\sigma}_{simple} \rangle$ and $\langle \mathcal{N}, \bar{\sigma} \rangle$ are compatible. For a number of rounds r, a latency function ℓ is *r-good* for $\langle \mathcal{N}, \bar{\sigma} \rangle$ if $\mathcal{N}$ computes $NOT(x)$ within r rounds under ℓ when starting from the initial state vector $\bar{\sigma}$. Our proof strategy is as follows. We will show that every latency function ℓ_{simple} is r-good for $\langle \mathcal{N}_{simple}, \bar{\sigma}_{simple} \rangle$, by showing that there exists a function ℓ such that

$$\langle \mathcal{N}_{simple}, \bar{\sigma}_{simple}, \ell_{simple} \rangle \text{ dominates } \langle \mathcal{N}, \bar{\sigma}, \ell \rangle.$$

Given a latency function ℓ_{simple} for the network $\mathcal{N}_{simple}$, let ℓ be a latency function for $\mathcal{N}$ which is similar on excitatory neurons and gives inhibitory neurons the latency value of the neuron x. I.e., $\ell(v, u, \tau) = \ell_{simple}(v, u, \tau)$ for every pair of excitatory neurons. In addition, $\ell(v', u, \tau) = \ell_{simple}(x, u, \tau)$ for every inhibitory neuron v', and a neuron $u \in V_{simple}$. All remaining latency values (i.e., the incoming edges to the inhibitors of $\mathcal{N}$) can be chosen arbitrarily.

We show by induction on the round τ, that (i) $\text{pot}_1(u, \tau, \mathcal{N}_{simple}, \ell_{simple}) \leq \text{pot}_1(u, \tau, \mathcal{N}, \ell)$ for every $u \in V_{simple} \setminus \{x\}$, $\tau \geq 0$, and (ii) $\text{pot}_0(u, \tau, \mathcal{N}_{simple}, \ell_{simple}) \geq \text{pot}_0(u, \tau, \mathcal{N}, \ell)$ for every $u \in V_{simple} \setminus \{x\}$ and $\tau \geq 0$. For $\tau = 0$, this is true as the potential values in $\tau = 0$ are simply the initial states, and the vector of initial states of $\bar{\sigma}$ and $\bar{\sigma}_{simple}$ are compatible. For the induction step, let $\tau \geq 1$, and assume correctness for all $\tau' \leq \tau - 1$. We will prove the claims for round τ. Let u be a neuron in $V_{simple} \setminus \{x\}$, and let $v_1, \ldots, v_k$ be its incoming excitatory neighbors.

The initial state of x is 0: By the induction assumption, it holds that $\text{pot}_0(u,\tau',\mathcal{N}_{simple},\ell_{simple}) \geq \text{pot}_0(u,\tau',\mathcal{N},\ell)$ for every round $\tau' \leq \tau-1$ and every neuron $v \in \{v_1,\ldots,v_k\}$. Thus every excitatory neuron v_i that fires in round τ' in the simulation of $\mathcal{N}$, also fires in round τ' in the simulation of $\mathcal{N}_{simple}$ for every $\tau' \leq \tau-1$. Combining with the definition of the latency function ℓ, we get that each spike from v_i that arrives to u at round τ of the simulation of $\mathcal{N}$ also arrives u in round τ in the simulation of $\mathcal{N}_{simple}$. Let ω be an inhibitory incoming neighbor of u in $\mathcal{N}$, then ω does not exist in $\mathcal{N}_{simple}$. Also note that since $\sigma_0(x)=0$, a negative spike from x never arrives at u in the simulation of network $\mathcal{N}_{simple}$. Therefore, no negative spikes arrive at u in $\mathcal{N}_{simple}$. Summing over the positive and negative spike weights, we get that $\text{pot}_0(u,\tau,\mathcal{N}_{simple},\ell_{simple}) \geq \text{pot}_0(u,\tau,\mathcal{N},\ell)$ for every $u \in V_1 \setminus \{x\}$.

The initial state of x is 1: By the induction assumption it holds that $\text{pot}_1(u,\tau',\mathcal{N}_{simple},\ell_{simple}) \leq \text{pot}_1(u,\tau',\mathcal{N},\ell)$ for every round $\tau' \leq \tau-1$ and every $u \in \{v_1,\ldots,v_k\}$. Thus every v_i that fires in round τ' in the simulation of $\mathcal{N}_{simple}$, also fires in round τ' in the simulation of $\mathcal{N}$. Combining with the definition of the latency function ℓ, we get that each spike from v_i that arrives at u in round τ of the simulation of $\mathcal{N}_{simple}$ also arrives at u in round τ in the simulation of $\mathcal{N}$.

Let ω be an inhibitory incoming neighbor of u in $\mathcal{N}$. By the definition of the latency function ℓ, and the fact that x fires in every round, for each spike that ω fires and arrives at u in round τ in $\mathcal{N}$, there is a spike from x that arrives at u in round τ in $\mathcal{N}_{simple}$. Since the edges from x have weight $-\infty$, we get that the negative spikes weight arriving at u in round τ in $\mathcal{N}_{simple}$ are larger (in absolute value) than the negative spikes in $\mathcal{N}$. Thus, summing up the both positive and negative spike weights, we get that $\text{pot}_1(u,\tau,\mathcal{N}_{simple},\ell_{simple}) \leq \text{pot}_1(u,\tau,\mathcal{N},\ell)$ for every $u \in V_{simple} \setminus \{x\}$. This proves the induction step for round τ. We get that $\langle \mathcal{N}_{simple},\bar{\sigma}_{simple},\ell_{simple}\rangle$ dominates $\langle \mathcal{N},\bar{\sigma},\ell\rangle$.

Finally, we show that if $\langle \mathcal{N}_1,\bar{\sigma}_1,\ell_1\rangle$ dominates $\langle \mathcal{N}_2,\bar{\sigma}_2,\ell_2\rangle$ and ℓ_2 is r-good for $\langle \mathcal{N}_2,\bar{\sigma}_2\rangle$, then also ℓ_1 is r-good for $\langle \mathcal{N}_1,\bar{\sigma}_1\rangle$.

Consider the simulation of $\langle \mathcal{N}_2,\bar{\sigma}_2,\ell_2\rangle$, and assume that the initial state of x is 0. Since ℓ_2 is r-good for $\mathcal{N}_2$, there is a round $\tau \leq r$ in which z fires in $\mathcal{N}_2$. Since $\langle \mathcal{N}_1,\bar{\sigma}_1,\ell_1\rangle$ dominates $\langle \mathcal{N}_2,\bar{\sigma}_2,\ell_2\rangle$, we can apply the condition of domination for z,τ and get that z also fires in round τ in $\mathcal{N}_1$. Now, assume that the initial state of x is 1. Since ℓ_2 is r-good for $\mathcal{N}_2$, there is *no* round τ in which z fires in $\mathcal{N}_2$. Since $\langle \mathcal{N}_1,\bar{\sigma}_1,\ell_1\rangle$ dominates $\langle \mathcal{N}_2,\bar{\sigma}_2,\ell_2\rangle$, we can apply the condition of domination for z and every τ, and get that z also never fires in $\mathcal{N}_1$. Hence, ℓ_1 is r-good for $\mathcal{N}_1$. This completes the proof of the lemma.

A.1.2 Size Lower Bound for Simple Networks

We show that the latency values defined in the step i satisfy the invariant in the beginning of step $i+1$.

Proving that the Invariants Hold: For round $i=0$, the correctness of invariant (I1) hold since in the first round $\tau=-1$ all positive spikes are set to arrive in round 0 in both ℓ_0,ℓ_1, while a spike from x arrives at round $\tau=L-1$. As for the correctness of invariant (I2), note that both simulations are similar for all neurons $V \setminus \{x\}$, and, again, a spike from x arrives only in round $L-1$. Therefore the same neurons are active in round $\tau=0$. Hence $A_{0,0}=A_{0,1}$ and we get correctness for (I2). We now show that the invariants are preserved after each step. Assume that the correctness holds at the beginning of each step $j \leq i$, and consider now the beginning of step $i+1$.

- (I1) By the construction, in all of the cases, the values we set for ℓ_0, ℓ_1 in step i are such that all spikes except the spike from x generated at round $iL+(L-1)$ which are generated in T_i arrive at some round $\tau \leq (i+1)L$, i.e. by the first round of T_{i+1}. Furthermore, spikes from x generated at round $iL+(L-1)$ is set to arrive in round $(i+1)L+(L-1)$. The invariant holds by combining with the correctness for all steps $i' \leq i$.
- (I2) We start by proving the following auxiliary claim.

▷ Claim 23. Consider the simulation of $\mathcal{N}$ with initial state $\bar{\sigma}_b$ and latency function ℓ_b, and let τ be round in T_i. Then:

1. For a strong neuron $u \in A_{i,1}$, u fires iff $\tau \in [iL, iL+(L-2)] \subseteq T_i$. For a strong neuron $u \in A_{i,0}$, u fires for every $\tau \in T_i$.
2. For a weak neuron $u \in A_{i,b}$, u fires iff $\tau = iL$.
3. For $u \notin A_{i,b}$, u is not active in round τ.

Proof. **Case $b = 1$:** We start by showing that all three claims hold for $b = 1$. By the definition of ℓ_1, the only positive spikes received by any neuron in some round $\tau \in [iL+1, iL+(L-1)]$ are self-loop spikes. Since a strong active neuron $u \in A_{i,1}$ receives an inhibiting spike from x in round $iL+(L-1)$, it is not active in this last round. For a weak neuron u', its spike from the self-loop is not strong enough to make it active. Lastly, for $u \notin A_{i,1}$ since no negative spikes arrive at u in round iL we have that $b(u) > 0$. Due to the fact that no spikes arrive at u in round τ, we get that u stays inactive.
Case $b = 0$: claim (1) holds since a strong $u \in A_{i,0}$ never gets inhibited as x never fires. We will now consider claim (2) and (3) for a neuron u that is either a *weak* neuron, or a strong neuron that is *not* in $A_{i,0}$. By Invariant (I1), all the positive spikes from the previous blocks arrived by the first round of T_i. Thus if u fires in any round $\tau \in [iL+1, iL+(L-1)]$ (i.e., any round which is not the first one in T_i), this must be due to the incoming spikes generated in the first round of T_i. We will know prove by induction on the round τ that u does not fire in any round in $[iL+1, iL+(L-1)]$.
Induction Base, Round $\tau = iL+1$:. By the definition of the latency function ℓ_0, no spike arrives at u from an incoming neighbor in that round. Therefore, if u is weak, then a spike from itself will not make it active in round τ. In addition, if $u \notin A_{i,0}$ then u did not fire in round iL, and thus receives no self-spike in round τ. Since no negative spikes arrive at u in round iL, for every $u \notin A_{i,0}$, it must hold that $b(u) > 0$.
Induction Step $\tau \geq iL+2$. Assume that the claims (2,3) hold up to round $\tau - 1$ and consider round $\tau \geq iL+2$. Consider first the case that u has either a weak neighbor g_j with $w(g_j, u) \geq b(u)$, or alternatively a strong neighbor h_j with $w(h_j, u) \geq b(u)/(L-1)$. Then by the definition of ℓ_0 (Case I in our definition), all the spikes fired by the incoming neighbors of u are scheduled to arrive in the first round of T_{i+1}. Therefore, u does not receive any spike in round τ, and remains inactive.
Next, consider the complimentary case where all the weak neighbors g_j satisfy that $w(g_j, u) < b(u)$, and all the strong neighbors h_j satisfy $w(h_j, u) < b(u)/(L-1)$.
Case (1): $\tau \in [iL+2, iL+\omega+1]$. By the induction assumption on $\tau - 1$, u did not fire in round $\tau - 1$. By the definition of ℓ_0, u receives a spike from at most one weak neighbor g_j in round τ, and since $w(g_j, u) < b(u)$, it does not fire in this round.
Case (2): $\tau \in [iL+(\omega+2), iL+(L-1)]$. Let h_j be a strong active neighbor of u. By the definition of ℓ_0, in round τ, u receives at most $(\omega+1)/(L-(\omega+1))$ of the spikes that fired by h_j during the interval $[iL, iL+\omega]$. Furthermore, there is one additional spike that h_j fired at round $\tau - 1$, that arrives at u in round τ. Note that u does not receive spikes from weak neighbors in round τ since all spikes from weak neighbors arrive in an earlier round

$\tau'' \in [iL+2, iL+(\omega+1)]$. In addition, since u did not fire in round $\tau-1$, it also does not get any self spikes in round τ. Overall, u receives at most $((\omega+1)/(L-(\omega+1)))+1$ spikes by strong neighbors in round τ, and no other spikes (by a weak neighbor or by u). Since the spikes from the strong neighbors have weight of at most $b(u)/(L-1)$, and there are s strong active neighbors, the overall weighted sum of the received spikes at round τ is at most

$$s \cdot ((\omega+1)/(L-(\omega+1))+1) \cdot \frac{b(u)}{L-1} <$$

$$s \cdot ((\omega+1)/s+1)\frac{b(u)}{L-1} = (s+\omega+1)\frac{b(u)}{L-1} < \quad b(u) \ , \tag{2}$$

where both inequalities follow as $s+\omega < L-2$. Therefore, u does not get activated at round τ, claims (2)+(3) follow. ◀

We are now ready to prove the induction step for (I2). Assume towards contradiction that there exists a neuron $u \in A_{i+1,0} \setminus \bigcup_{i' \leq i+1} A_{i',1}$. Since u is active in the first round of T_{i+1}, using Claim 23(3), it must have an incoming neighbor that fires in the *first round* of the previous block. Let $A_{i,0}(u) = \Gamma_{in}(u) \cap A_{i,0}$ be those neighbors.

▷ Claim 24. For every strong neuron $v \in A_{i,0}(u)$, it holds that $w(v,u) < b(u)/(L-1)$.

Proof. Consider such strong $v \in A_{i,0}(u)$. By Invariant (I2) for the beginning of step i, there exists a round $j \leq i$ such that $v \in A_{j,1}$. When running $\mathcal{N}$ with initial state $\bar{\sigma}_1$ and the latency function ℓ_1, by Claim 23(1), v fires in all of the $L-1$ rounds of $[jL, jL+(L-2)]$. By the construction of ℓ_1, all these $L-1$ spikes arrive in round $(j+1)L$. Therefore, if $(L-1)\cdot w(v,u) \geq b(u)$, then u is activated in round $(j+1)L$, i.e., $u \in A_{j+1,1}$, in contradiction to the definition of u. Therefore $(L-1)\cdot w(v,u) < b(u)$ for every strong neuron v. ◀

▷ Claim 25. For every weak neuron $v \in A_{i,0}(u)$, it holds that $w(v,u) < b(u)$.

Proof. Consider such weak neuron $v \in A_{i,0}(u)$. When running $\mathcal{N}$ with initial state $\bar{\sigma}_1$ and latency function ℓ_1, by Claim 23(2), v fires once in the interval T_j, i.e., in the first round jL. By the construction of ℓ_1 this spike arrives in round $(j+1)L$. Therefore, if $w(v,u) \geq b(u)$, then u is activated in round $(j+1)L$, implying that $u \in A_{j+1,1}$, contradiction to the definition of u. We therefore conclude that $w(v,u) < b(u)$ for every weak neuron $v \in A_{i,0}(u)$. ◀

Let ω, s be the number of weak (resp., strong) neurons in $A_{i,0}(u)$. By Claims 25 and 23(2), all active weak neurons in T_i fire only in the first round of that block. By the definition of the latency function, all these spikes are scheduled to the first ω rounds in T_i, and therefore none of them is scheduled to arrive on the first round of T_{i+1}. This implies that u fires in that round due to the spikes generated by its strong neighbors.
By Claim 24, $w(v,u) < b(u)/(L-1)$ for each such strong neighbor v of u. This in particular implies that u is a weak neuron, and by Claim. 23 it did not fire in the last round of T_i. By the definition of the latency function, the spikes generated by such strong neighbors are divided almost evenly among $L-\omega$ rounds, up to the first round of T_{i+1}. Each round gets at most $s\cdot(\omega/(L-\omega)+1)$, which is strictly less than $b(u)$ by Eq. (2). Leading to contradiction for the assumption that $u \in A_{i+1,0}$.

Since ℓ_1 is a good latency function when starting with $x = 1$, we have that z never fires and thus $z \notin \bigcup A_{i,1}$. By applying invariant (I2) on the output neuron z for every round $\tau \geq 0$, we get that $z \notin \bigcup A_{i,0}$. By using Claim 23, we get that z never fires with ℓ_0 and $x = 0$. Contradiction to the fact that $\mathcal{N}$ solves $NOT(x)$.

A.2 Time Lower Bound for Computing $NOT(x)$

In this section we show the following.

▶ **Lemma 26.** *Every network $\mathcal{N}$ that computes $NOT(x)$ in the L-bounded asynchronous setting requires $\Omega(L^3)$ rounds.*

By Lemma 12, we restrict attention to a simple network $\mathcal{N} = \mathcal{N}_{simple}$ with one input neuron x that computes $NOT(x)$. Similarly to the size lower bound, we define two *conflicting* latency functions ℓ_0 and ℓ_1, such that if ℓ_1 is good when $x_0 = 1$, then the output neuron z of $\mathcal{N}$ fires after $\Omega(L^3)$ rounds in the simulation with the latency function ℓ_0 and $x_0 = 0$.

- The simulation with the latency function ℓ_0 is partitioned into consecutive blocks of L rounds, $T_i = [iL, iL + (L-1)]$ for every $i \in \mathbb{N}$.
- The simulation with the latency function ℓ_1 is based on the notion of *important* and *unimportant* rounds. Consider the L-round interval $T_k = [k \cdot L, k \cdot L + (L-1)]$ for $k \in \mathbb{N}$. Among the first $L/2$ rounds, there is an important round once every 16 rounds, and the rest are unimportant. Furthermore, each of the last $L/2$ rounds of the interval are unimportant. I.e., the important rounds in the interval are $\{kL + 16j \mid 16j < L/2, j \in \mathbb{N}\}$. Denote by τ_i the i^{th} important round in the simulation. Note that by definition $\tau_{i+1} - \tau_i \leq L/2$. In our arguments, the configuration of the network in the i^{th} important round τ_i of the simulation with ℓ_1 and $x_0 = 1$ will be compared against the configuration in round iL (i.e., the first round of the block T_i) in the simulation with ℓ_0 and $x_0 = 0$.
- Active subsets of neurons: For every $i \in \mathbb{N}$, let $A_{0,i}$ be the firing neurons (hence *active*) of round $i \cdot L$ (the first round of the block T_i) in the simulation of $\langle \mathcal{N}, \sigma_0, \ell_0 \rangle$. Similarly, let $A_{1,i}$ be the firing neurons in round τ_i of the simulation of $\langle \mathcal{N}, \sigma_1, \ell_1 \rangle$. Also define $A'_{b,i} = A_{b,i} \setminus \bigcup_{j \leq i-1} A_{b,j}$, the neurons that fire for the first time in "round" i.
- For every neuron u, $b \in \{0,1\}$ and $i \in \mathbb{N}$, let $A_{b,i}(u) = A_{b,i} \cap N_{in}(u)$, $A'_{b,i}(u) = A'_{b,i} \cap N_{in}(u)$ where $N_{in}(u)$ is the set of incoming neighbors of u.
- For a subset of neurons $V' \subseteq V$ and a neuron u, let $w(V', u) = \sum_{v \in V'} w(v, u)$. Moreover, let $\mathbf{S}(V')$ and $\mathbf{W}(V')$ be the strong[9] and weak (respectively) neurons subsets of V'.

A.2.1 Defining the latency functions ℓ_0 and ℓ_1

Throughout, a spike event is represented by a triplet $\langle v, u, \tau \rangle$ where $v \in N_{in}(u)$ fires in round τ. Since the functions are nice, the latency values for the self spikes $\langle u, u, \tau \rangle$ for every u and τ are set to 1. For technical reasons, it is more convenient to start the simulations in round -1, rather than in round 0. For this first round -1, let $\ell_b((v, u), -1) = 1$ for every u and every $v \neq x$, and $\ell_b((x, u), -1) = L$ for every u and $b \in \{0, 1\}$. As a result, the positive spikes (by any $v \neq x$) fired in round -1 arrive to their destination in round 0, and the negative spikes of x arrive in round $L-1$. We now define the latency values for the remaining spikes.

[9] Recall that a neuron u is strong if $w(u, u) \geq b(u)$ and it is weak otherwise.

Defining the function ℓ_0. Note that when $x_0 = 0$, x never fires and thus there is no need to define ℓ_0 values for the spikes of x. We define ℓ_0 iteratively in a block by block manner. Here we do not accumulate spikes and spikes generated in the i^{th} block T_i will arrive by the first round of the $(i+1)^{th}$ block T_{i+1}. Fix a block $T_i = [iL, iL + (L-1)]$ for $i \geq 0$ and assume that the latency values ℓ_0 for all prior spikes in rounds $\tau < iL$ have already been fixed. Thus the active set $A_{0,i}$ can be determined. First, the algorithm checks if there is a way to spread all the spikes generated in rounds of T_i among the interval $[iL+2, (i+1)L]$, in a way that guarantees that u will not fire in *any* of the rounds this interval. In particular, *no* spike is scheduled to arrive in round $iL+1$. Otherwise, all spikes generated in this block are scheduled to arrive in round $(i+1)L$ (the first round of the $(i+1)^{th}$ block).

Defining the function ℓ_1. The definition of the function ℓ_1 is more involved. Unlike the function ℓ_0 in which all spikes generated in block T_i are scheduled by round $(i+1)L$, here the setting is slightly more sensitive. Specifically, the scheduling algorithm of ℓ_1 will make sure that non-self spikes arrive to their destination only in important rounds.

Spikes by the input (inhibitory neuron) x: All spikes from x are scheduled to arrive in the last round of the blocks T_i, namely, in rounds of the form $i \cdot L + (L-1)$. Formally, for every spike $\langle x, u, \tau \rangle$ where $\tau = i \cdot L + (L-1)$ for some $i \in \mathbb{N}$, let $\ell_1((x,u),\tau) = L$ thus arriving in round $\tau + L = (i+1)L + L - 1$. For every $\tau \in [iL, iL + (L-2)]$, let $\ell_1((x,u),\tau) = (iL + (L-1)) - \tau$, thus arriving in round $iL + (L-1)$ as desired.

Spikes by $v \neq x$: The latency values are defined in a round by round fashion, such that for every important round τ_i, every neuron u gets activated if possible. Otherwise the arrival of the spikes towards u are postponed (when possible) to the next important round τ_{i+1}. The spikes generated at non-important rounds will be always delayed to the next important round. This is always possible as the distance to the next important round is at most $L/2$. For a subset of spikes S, let $w(S) = \sum_{\langle v,u,\tau \rangle \in S} w(v,u)$ be the total weight of the spikes in S. For every important round τ_i, we will maintain a list of pending spikes $\mathcal{R}_{\tau_i}(u)$ towards u that were not yet scheduled. In every step $\tau \geq 0$, the algorithm will schedule the spikes generated in this round. If the round τ is important, then the algorithm will also make decisions regarding the set of pending spikes $\mathcal{R}_{\tau_i}(u)$.

We will keep the invariant that at the beginning of step τ, the latency value of all spikes scheduled to arrive by round τ has already been determined. As we will see, the non-self spikes will always be scheduled to arrive in important rounds. As a result, a neuron u fires in an unimportant round τ iff u is strong and it fired in round $\tau - 1$. Initially, for every neuron u, the algorithm adds every non-self spike $\langle v, u, -1 \rangle$ to $\mathcal{R}_{\tau_i}(u)$. For every $\tau \geq 0$, we consider the following algorithm.

- All self-spikes $\langle u, u, \tau' \rangle$ are given a latency value of $\ell(u, u, \tau') = 1$.
- **Handling important rounds τ_i.** Consider a neuron u. If u fired in round $\tau_i - 1$, add the self-spike $\langle u, u, \tau_i - 1 \rangle$ to the pending spike set $\mathcal{R}_{\tau_i}(u)$. If the total weight of its pending spikes (towards u) is sufficiently large to make u fire, all the non-self spikes are scheduled to arrive in τ_i. Formally, if $w(\mathcal{R}_{\tau_i}(u)) \geq b(u)$, schedule all these spikes to round τ_i by setting $\ell(v, u, \tau') = \tau_i - \tau'$ for every spike $\langle v, u, \tau' \rangle \in \mathcal{R}_{\tau_i}(u)$.
  Otherwise, if the total weight of pending spikes is small, i.e., $w(\mathcal{R}_{\tau_i}(u)) < b(u)$, the non-self spikes are deferred to the next important round τ_{i+1} if possible (i.e., if the latency does not exceed its upper bound L). Formally, for every non-self pending spike $\langle v, u, \tau' \rangle \in \mathcal{R}_{\tau_i}(u)$, if $\tau_{i+1} - \tau' > L$ then let $\ell((v,u),\tau') = L$ (i.e., $\langle v, u, \tau' \rangle$ cannot be further deferred). Otherwise, add $\langle v, u, \tau' \rangle$ to the pending spike set $\mathcal{R}_{\tau_{i+1}}(u)$ of the next important round τ_{i+1}.

Finally, all spikes generated in round τ_i are also (safely) added to the pending list $\mathcal{R}_{\tau_{i+1}}(u)$.

- **Handling unimportant rounds.** The non-self spikes towards u generated in round τ are added to the pending spike set $\mathcal{R}_{\tau_{i+1}}(u)$ of the next important round τ_{i+1} (after round τ).

First observe that the function ℓ_1 is valid: All self-spikes have a latency value of 1. Moreover, the non-self spikes have a latency value in $[1, L]$. To see this observe that for unimportant round τ, a non-self spike $\langle v, u, \tau \rangle$ is added to the pending list $\mathcal{R}_{\tau_{i+1}}(u)$ where τ_{i+1} is the next important round after τ. Due to the fact that $\tau_{i+1} - \tau_i \leq L/2$, this assignment is valid. In addition, the pending spikes $\langle v, u, \tau \rangle \in \mathcal{R}_{\tau_i}(u)$ are deferred to τ_{i+1} only if $\tau_{i+1} - \tau \leq L$.

A.2.2 Proof of Lemma 26

The key lemma that establishes Lemma 26 is the following:

▶ **Lemma 27.** *For every neuron $u \neq x$ with $u \in A'_{0,i}$ for $i < L^2/1024$, there exists some $i' \leq i$ such that $u \in A'_{1,i'}$.*

By the correctness of the simple network $\mathcal{N}$, the output neuron z should not fire when $x_0 = 1$ and with the latency function ℓ_1. In other words, $z \notin A_{1,i'}$ for any i'. By Lemma 27, we get that z can only be in $A'_{0,j}$ for some $j \geq L^2/1024$, hence firing when $x_0 = 0$ only after $\Omega(L^3)$ rounds. We start with the following simple observation.

▷ Observation 28. In the simulation of $\langle \mathcal{N}, \bar{\sigma}_0 \rangle$ with ℓ_0, it holds for every i that: (i) each strong neuron $s \in A_{0,i}$ fires in every round of block T_i; (ii) each weak neuron $\omega \in A_{0,i}$ which is not x fires only in the first round of block T_i; and (iii) every neuron $v \notin A_{0,i}$ does not fire in any round of block T_i.

Proof. (i). In the simulation with ℓ_0 with $x_0 = 0$ there are no inhibiting spikes, and if a strong neuron s fires in some round, it will keep on firing for the rest of the simulation.

(ii). By the definition of the latency function ℓ_0, no spikes from incoming neighbors of the weak neuron ω arrive in round $iL + 1$, the second round of block T_i. We will prove by induction on $\tau \in [iL + 1, iL + (L - 1)]$ that ω does not fire in round τ. For the base of the induction, since $\omega \neq x$ is excitatory and weak, it holds that $0 \leq w(\omega, \omega) < b(\omega)$, thus ω does not fire in round $iL + 1$. Assume that the claim holds up to round $\tau \geq iL + 1$ and consider round $\tau + 1$. Since ω did not fire in round τ by the induction assumption, it does not receive a self spike in round $\tau + 1$. By the definition of the function ℓ_0, the non-self spikes that arrive in round $\tau + 1 < (i + 1)L$ cannot make ω fire. Thus ω does not fire in $\tau + 1$ and (ii) holds.

(iii). Let $v \notin A_{0,i}$, i.e., v did not fire in round iL. Since v does not receive negative spikes in round iL (as the spikes of x are always scheduled to the last round of the blocks). We can then conclude that $b(v) > 0$. Since in round $iL + 1$, it receives no self-spike and no other spike, it also did not fire in round $iL + 1$. The argument then follows inductively in the same manner as in (ii). ◀

We next state the following claim which is crucial to complete the key lemma.

▷ Claim 29. Fix a neuron $u \in A'_{0,i}$ such that for every $v \in A_{0,i-1}$ it holds that $w(v, u) < b(u)$. Then the total weight of spikes fired towards u in block T_{i-1} is at least $L \cdot b(u)/8$.

We first complete the proof of Lemma 27 and only then prove Claim 29.

Proof of Lemma 27. The proof is shown by induction on the block i. For the base case of $i = 0$, note that the initial states and the latency functions for the neurons $V \setminus \{x\}$ in both simulations are the same, and that spikes from x (that exist only in the simulation with ℓ_1) arrive only in round $L - 1$. This implies that in both simulations the same neurons (except for x) are active in round $\tau = 0$, hence $A_{0,0} = A_{1,0} \setminus \{x\}$. Now consider the block T_i for $1 \leq i < L^2/1024$. Let $u \in A'_{0,i}$, i.e. u fires for the first time in round iL in the simulation with ℓ_0 and $x_0 = 0$.

Case 1: There exists a previously firing dominant incoming neighbor: First assume that u has some incoming neighbor $v \in A_{0,i-1}$ with $w(v, u) \geq b(u)$. By definition $v \in A'_{0,j}(u)$ for some $j \leq i - 1$, and then by the induction assumption $v \in A'_{1,i'}$ for some $i' \leq j \leq i - 1$. By definition of the latency function ℓ_1, since $w(v, u) \geq b(u)$, the total weight of spikes from incoming neighbors will be sufficient to activate u in the next important round, $\tau_{i'+1}$. Therefore, $u \in A_{1,i'+1}$, which implies $u \in A'_{1,i''+1}$ for some $i'' \leq i' + 1$. Since $i'' \leq i' + 1 \leq i$ the condition holds.

Case 2: All previously firing incoming neighbors are not dominant: By applying Claim 29 on u and block T_i, we get that the total weight of spikes fired towards u in block T_{i-1} is at least $L \cdot b(u)/8$. Due to Observation 28, we get

$$L \cdot w(\mathbf{S}(A_{0,i-1}(u)), u) + w(\mathbf{W}(A_{0,i-1}(u)), u) \geq \frac{L}{8} \cdot b(u).$$

By the definition of $A'_{0,j}$ and the induction assumption, it holds that

$$A_{0,i-1} \subseteq \bigcup_{j \leq i-1} A'_{0,j} \subseteq \bigcup_{i' \leq i-1} A'_{1,i'}.$$

We now consider the simulation with $x_0 = 1$ and the latency function ℓ_1, and partition all the rounds until τ_i into k blocks of L rounds (expect perhaps the last one). Formally, for every $j \leq k - 2$, let $B_j = [jL, jL + (L - 1)]$ and let $B_{k-1} = [(k - 1)L, \tau_{i-1} + 15]$. Denote by $\mathbf{S}(B_j)$ and $\mathbf{W}(B_j)$ the strong and weak (respectively) incoming neighbors of u that fire in some round of B_j. Using these notations, we can write

$$\sum_{j=0}^{k-1} L \cdot w(\mathbf{S}(B_j), u) + w(\mathbf{W}(B_j), u) \geq \frac{L}{8} \cdot b(u).$$

Case 2.1: Most of the weight is in the last block. We first assume that

$$L \cdot w(\mathbf{S}(B_{k-1}), u) + w(\mathbf{W}(B_{k-1}), u) \geq \frac{L}{16} \cdot b(u).$$

Consider the algorithm that defines ℓ_1, and recall that $\mathcal{R}_{\tau_{i'}}(u)$ is the set of pending spikes that were not yet scheduled when the algorithm considered the important round τ_i. The interesting case is when u did not fire in any round of B_{k-1}. In such a case, all the spikes generated towards u in the rounds of B_{k-1} were added to the pending list of $\mathcal{R}_{\tau_i}(u)$. Note that each strong neuron $v \in \mathbf{S}(B_{k-1})$ fires at least 16 spikes in B_{k-1}, since $\tau_i - \tau_{i-1} = 16$. Furthermore, each $v \in \mathbf{W}(B_{k-1})$ fires at least one spike in B_{k-1}. Moreover, the gap between any $\tau_{i'} \in B_{k-1}$ and τ_i is at most L rounds, so they do not exceed the maximal latency in τ_i. Altogether, we get that

$$\begin{aligned} & w(\mathcal{R}_{\tau_i}(u)) \geq 16 \cdot w(\mathbf{S}(B_{k-1}), u) + w(\mathbf{W}(B_{k-1}), u) \geq \\ & \frac{16}{L} \cdot (L \cdot w(\mathbf{S}(B_{k-1}), u) + w(\mathbf{W}(B_{k-1}), u)) \geq b(u) \ . \end{aligned} \tag{3}$$

Therefore u fires in τ_i and $u \in A_{1,i'}$ for some $i' \le i$ as desired.

Case 2.2: Most of the weight is in the first $k-1$ blocks. It remains to consider the complementary case where

$$\sum_{j=0}^{k-2} L \cdot w(\mathbf{S}(B_j), u) + w(\mathbf{W}(B_j), u) \ge \frac{L}{16} \cdot b(u).$$

Since $i < L^2/1024$ and each block B_j for $j \le k-2$ consists of $L/32$ important rounds, we have $k \le \dfrac{L^2/1024}{L/32} = \dfrac{L}{32}$. Therefore, by an averaging argument there exists B_j for $j \le k-2$ satisfying that:

$$L \cdot w(\mathbf{S}(B_j), u) + w(\mathbf{W}(B_j), u) \ge 2 \cdot b(u). \tag{4}$$

First observe that every strong neuron $s \in \mathbf{S}(B_j)$ fires for at least $L/2$ rounds in this block. The reason is that there is a gap of $L/2$ rounds between the last important rounds of B_j and the round where the inhibiting spike from x arrives. During this time interval every strong neuron in $\mathbf{S}(B_j)$ keeps on firing. Now, assume that u does not fire in any round of B_j, and denote the first important round of B_{j+1} by $\tau_{i'}$. Again, consider the algorithm that defines ℓ_1. Since u did not fire in any round of the block B_j, all the spikes that are fired towards u in B_j are in the residual set $\mathcal{R}_{\tau_{i'}}(u)$. Therefore by Eq. (4), we get that $w(\mathcal{R}_{\tau_{i'}}(u)) \ge (L/2) \cdot w(\mathbf{S}(B_j), u) + w(\mathbf{W}(B_j), u) \ge b(u)$, and u fires in $\tau_{i'}$. Therefore, we get that u fires either in some important round of B_j or in $\tau_{i'}$. In both cases there is a round $\tau_{i''}$ with $i'' \le i$ such that $u \in A_{1,i''}$. This implies $u \in A'_{1,i''}$ for $i'' \le i$, and the condition holds. ◀

Finally, it remains to prove Claim 29.

Proof of Claim 29. Recall that $\mathbf{S}(A_{0,i-1}(u))$ and $\mathbf{W}(A_{0,i-1}(u))$ are the strong and weak (respectively) incoming neighbors of u that fire in block T_{i-1}. If $w(\mathbf{S}(A_{0,i-1}), u) \ge b(u)/8$, then by Observation 28 the total spike weight fired in block $i-1$ is at least $L \cdot b(u)/8$, and we are done. Therefore, it remains to consider the case where $w(\mathbf{S}(A_{0,i-1}), u) < b(u)/8$ and $w(\mathbf{W}(A_{0,i-1})) < L \cdot b(u)/8$. We will show that in this case, there is a way to schedule all spikes fired towards u in block T_{i-1} to arrive in rounds $[(i-1)L+2, iL]$, such that u does not get activate in any of these rounds. By the definition of ℓ_0, we get that u does not get activated in any of the rounds $[(i-1)L+2, iL]$, and in particular $u \notin A'_{0,i}$, thus a contradiction.

First observe that $b(u) > 0$ since u did not fire in round $(i-1)L$ (as $u \notin A_{0,i-1}$) and it did not receive any negative spike in that round (as all negative spikes arrive in the last rounds of the blocks). We next show that all the spikes generated in block T_{i-1} can be scheduled in rounds $[(i-1)L+2, iL]$ without making u fire in any of these rounds. Since the scheduling algorithm of ℓ_0 works in this manner, we will get a contradiction to the fact that $u \in A_{0,i}$.

Scheduling spikes from weak neighbors. Let $F_{\mathbf{W}} = \{\langle v, u, (i-1)L\rangle \mid v \in \mathbf{W}(A_{0,i-1})\}$ be the spikes of weak neighbors fired in the block T_{i-1}. Recall that by Observation 28, these weak neurons fire only in the first round. Since these spikes are fired in round $(i-1)L$, they can arrive in any of the rounds $[(i-1)L+2, iL]$. As the total weight of the weak spikes is at most $Lb(u)/8$, we show that we can schedule them in a greedy manner into at most $L/2-2$ rounds while keeping the total weight in each such round to strictly less than $b(u)$. We traverse the weak spikes one by one, and start throwing them into rounds in $[(i-1)L+2, iL-1]$. We add a spike to round τ as long as the total weight of weak spikes scheduled to it is at most $b(u)/2$. If the addition of the next weak spike raises the weight to above $b(u)$ it is deferred to the next round $\tau+1$. Let τ' be the last round to which the weak spikes are schedules. Since in each $\tau \in [(i-1)L+2, \tau'-1]$ the total weight of weak spikes is at least $b(u)/2$, we get that $\tau' \leq (i-1)L+L/4+3 \leq (i-1)L+L/2$ as desired.

Scheduling spikes from strong neighbors. We next turn to show that also the strong spikes can be scheduled in a balanced manner in the remaining $L/2$ slots of the block T_i without activating the neuron u. Let $F_{\mathbf{S}} = \{\langle v, u, \tau\rangle \mid v \in \mathbf{S}(A_{0,i-1}), \tau \in T_{0,i-1})\}$ be the spikes of strong neighbors fired in block T_{i-1}. For a spike $\langle v, u, \tau\rangle \in F_{\mathbf{S}}$ with $\tau \leq (i-1)L + (L/2-1)$, schedule $\langle v, u, \tau\rangle$ to arrive in round $\tau + L/2 + 1$. For $\langle v, u, \tau\rangle \in F_{\mathbf{S}}$ with $\tau \geq (i-1)L + L/2$, schedule $\langle v, u, \tau\rangle$ to arrive in round $\tau + 1$. In this way, due to Observation 28, u receive two spikes from each $v \in \mathbf{W}(A_{0,i-1})$ in each round $\tau \in [(i-1)L + L/2 + 1, iL]$. Since $w(\mathbf{S}(A_{0,i-1})) < b(u)/8$, we get that the total weight of spikes that u receives in each of these rounds is less than $b(u)/4$, and therefore u does not get activated. Overall, all spikes generated in the block T_{i-1} are scheduled by ℓ_0 without activating the neuron u in any of the rounds $[(i-1)L+2, iL]$, contradiction to the fact that $u \in A_{i,0}$. The claim follows. ◀

B Missing Proofs for the Positive Results

B.1 Synchronization of Boolean Gates

Proof of Observation 13. The network is as follows: connect each input neuron x_i to the output neuron z by an edge of weight $w(x_i, z) = 1$, and let the bias of z be $b(z) = 1$. First note that if all input neurons x_i did not fire in round 0, then $\text{pot}(z, \tau) = -1$ for all τ, and z will not fire. If a neuron x_i fires in round τ, since the latency of each edge is at most L, there is a round $\tau' \in [\tau+1, \tau+L]$ in which the spike from x_i arrives to z. Thus, in round τ', the weighted incoming sum to z is at least 1, therefore $\text{pot}(z, \tau') \geq 0$ and z fires in round τ'. ◀

The Complete Proof of Lemma 14: We analyze the correctness of the network $\mathsf{NOT}_{\mathsf{sync}}$. We begin by proving the following auxiliary claim.

▷ Claim 30. If all the intermediate neurons $v_0, \ldots v_L$ fire starting round τ for at least $L(L+1)$ rounds, then there exists a round $\tau' \in [\tau+1, \tau+L(L+1)]$ in which the output neuron z fires (i.e., regardless of the latencies of the edges).

Proof. For every $i \in \{0, 1 \ldots, L(L+1)\}$, denote the L-length interval $T_i = [\tau + i \cdot L, \tau + (i+1)L - 1]$. In addition, define $\widetilde{T} = T_0 \cup ... \cup T_{L+1}$. Let q_i be the number of spikes that were fired in the interval of T_i but received by z in the next interval T_{i+1}. Note that since the maximum edge latency is L, in the worst case the spikes of interval T_i must arrive to z by the end of the next interval T_{i+1}. We next prove by induction on i that either z fires by the end of the interval T_i, or $q_{i+1} \geq i \cdot L$. For the base of the induction, consider $i = 0$. If z did not fire in some round during T_0, we claim that $q_1 \geq L$. Since all the $L+1$ neurons fire in every round during the interval, overall $L(L+1)$ many spikes where fired. By the fact that z did not fire during T_0, we have that in each of these rounds, it received at most L spikes. This implies that z received at most L^2 many spikes during T_0, and therefore at least $q_1 \geq L$ many spikes will be received by z in the interval T_1. Assume that the claim holds up to $i-1$ and consider the i^{th} interval. If z fired by the end of the i^{th} interval T_i, we are done. Otherwise, by induction assumption for $i-1$, we have that $q_i \geq i \cdot L$. In addition, all these q_i spikes must be received at z during the interval T_i. Then, in interval T_i we again have a total of $L(L+1)$ fresh spikes by the neurons $v_0, \ldots v_L$. This creates a total of $L(L+1) + i \cdot L$ spikes. As z did not fire in T_i, it received at most L^2 many spikes, leaving at least $q_{i+1} \geq L(L+1) + i \cdot L - L^2 \geq (i+1)L$ spikes for the next interval. This completes the proof of the induction step.

Overall, for $i = L$, we have that either z fired by the end of the interval T_L, or that $q_{L+1} \geq L(L+1)$. In the latter case, since all these spikes must arrive during the last interval T_{L+1}, by the pigeonhole principle there must be a round in this interval in which z received at least $L+1$ spikes and fire. This completes the proof of the claim. ◀

Proof of Lemma 14. Due to Claim 30, it remains to show that if x did not fire in round 0, then there must be a starting round τ, in which all the neuron $v_0, \ldots v_L$ fire for at least $L(L+1)$ rounds.

If x did not fire, then neither the memory neuron m nor the reset neuron r fire during the execution. Since we assume that v^* fires in round 0, it must hold that all these neurons fire starting some round $\tau \in [5L^2, 5L^3 + 2L]$. This holds due to the self-loops on the neurons $v_0, \ldots v_L$, and the chain of length $5L^2$. By Claim 30, there exists a round $\tau' \in [\tau + 1, \tau + L(L+1)]$ in which z fires.

We next show that if x fired in round 0, then z would not fire in any round. The key observation is as follows:

▷ Observation 31. In order for z to fire in some round τ, it must receive spikes from *at least* two different v_i neurons.

Proof. To see this, note that since the maximum edge latency is L, in round τ, z can receive spikes only from the L previous rounds $\tau - L, \ldots, \tau - 1$. In particular, a single neuron can be accounted for at most L many spikes received by z in a given round. Finally, since the bias of z is $L+1$, and all edge weights are 1, we conclude that z must receive spikes from at least two neurons in order to fire. ◀

When x fires in round 0, the memory neuron m fires from round $\tau_m \in [1, L+1]$ ahead, due to its self-loop. Hence, starting round $\tau_r \in [2, 2L+2]$, the reset neuron r starts firing at least *once* in every interval of $2L$ rounds. Recall that each v_i gets a negative spike from the inhibitor r and positive spike from the neuron $c_{5iL} \in C$. We next show that each neuron v_i gets inhibited at least L rounds *before* the activation of the neurons v_{i+1}. As a result, at any point of time, there will be no two neurons v_i and v_j such that z received both of their spikes in the same round. By induction on i, the first intermediate neuron v_0 has an incoming edge from $c_0 = v^*$, and thus it begins to fire in some round $\tau' \in [0, L]$. Due to the negative edge from the reset neuron r, it stops firing before round $3L+2$. Since v_1 has an incoming edge from c_{5L}, it starts firing only after round $5L+1$, and therefore z starts receiving spikes from v_1 only starting round $5L+2$. Assume the claim is correct for neurons $v_0, \ldots, v_{i-1}$ and consider neuron v_i. If the neuron v_i starts firing in round τ_i, by round $\tau_i + 2L$ it is inhibited by r. Because v_i starts to fire after receiving a spike from $c_{5 \cdot i \cdot L}$ and v_{i+1} starts firing after receiving a spike from $c_{5(i+1)L}$, neuron v_{i+1} begins to fire only after round $\tau_i + 4L$, at least L rounds after v_i is inhibited.

Hence, z cannot receive input from two different neurons v_i, v_j at the same round, and the claim follows by combining Observation 31. Finally, the next observation plays a rule in the subsequent constructions.

▷ Observation 32. The correctness still holds even if the chain starts to fire at some round $\tau > 0$. In this case the output neuron z fires in some round $t \in [\tau + 1, \tau + \Theta(L^3)]$.

Proof of Observation 32. The correctness of the observation follows from the fact that the input neuron x activates the memory neuron m, that keeps on firing (i.e., presenting the state of x) due to its self-loop. Thus all arguments in Lemma 14 still hold in case the chain starts to fire in any later round. ◀

B.2 Synchronization of a Boolean Circuit, Proof of Lemma 15

The Construction. Given a Boolean circuit $\mathcal{A}$ of OR / NOT gates $g_1, \ldots, g_m$ of depth d, we describe a construction of an analogous neural network $\mathcal{N}$ with a similar execution. For every g_i, let $\mathsf{Sync}(g_i)$ be the synchronized sub-network of the gate g_i. Specifically, for a NOT gate (resp., OR) g_i, the sub-network $\mathsf{Sync}(g_i)$ is taken from Lemma 14 (resp., Observation 13). Recall, that for a NOT gate g_i, its syncronized sub-network $\mathsf{Sync}(g_i)$ contains a chain of neurons where the head of the chain c_0 will be denoted hereafter by v_i^*. The network $\mathcal{N}$ consists the following components:

1. Input neurons $x_1, \ldots, x_n$, and output neurons $z_1, \ldots, z_k$, that serve as the input and the output for the network $\mathcal{N}$.
2. A chain $C = [c_0, \ldots, c_q]$ containing $q + 1 = \alpha d L^3 + 1$ neurons, where α is a constant satisfying that $\alpha L^3 \geq 5L^3 + L$. For every $i \geq 0$, the neuron c_i has bias $b(c_i) = 1$. Moreover, for every $i \geq$ the neuron c_i has a positive incoming edge from c_{i-1} with weight $w(c_{i-1}, c_i) = 1$. Our simulation starts with neuron c_0 firing.
3. A $\mathsf{Sync}(g_i)$ network (using Lemma 14 and Observation 13 respectively) for every gate g_i.

The connections between these components are as follows:

1. For every gate g_i in the first layer, the input for its synchronized sub-network $\mathsf{Sync}(g_i)$ is given by $x_{i,1}, \ldots, x_{i,k_i}$, namely, the input bits of the gate g_i in the circuit $\mathcal{A}$.
2. For every gate g_i in layer $j \geq 2$, denote by $g_{i,1}, \ldots, g_{i,k_i}$ the input of the gate g_i in the circuit $\mathcal{A}$. In the network $\mathcal{N}$, the input to the sub-network $\mathsf{Sync}(g_i)$ are the output neurons of the sub-networks $\mathsf{Sync}(g_i), \mathsf{Sync}(g_{i_1}), \ldots, \mathsf{Sync}(g_{i_k})$.
3. The output gates of the network $\mathcal{N}$ are the output neurons of the sub-networks $\mathsf{Sync}(o_1), \ldots, \mathsf{Sync}(o_k)$, where $o_1, \ldots, o_k$ are the output gates of the circuit $\mathcal{A}$.
4. Finally, the synchronized sub-networks of the NOT gates are connected to the chain C as follows. For each NOT gate g_i in every layer j, the $(j\alpha L^3)^{th}$ neuron $c_{j\alpha L^3}$ in the chain has an outgoing edge to v_i^* with weight 1 (where v_i^* is the head of the internal chain in $\mathsf{Sync}(g_i)$), since the bias of v_i^* is 1, a spike from $c_{j\alpha L^3}$ makes v_i^* fire.

Figure 3 illustrates the construction for a circuit with 4 NOT and OR gates of depth 3. We note that one can shave an L-factor in the size and time overhead of lemma 15, by reusing the synchronization chain for all the Boolean gates in the network. For clarity of explanation, we defer this improvement to the full paper.

Correctness. Let V be the total set of neurons in $\mathcal{N}$, and let $\ell : V \times V \times \mathbb{N} \to [1, L]$ be a fixed (arbitrary) nice latency function. First note that in the global chain C, each of the neurons fires once, in a sequential manner. Recall, that we assume that the starter c_0 fires in round $\tau_0 = 0$. For every $j \in \{1, \ldots, d\}$, let τ_j be the round in which $c_{j\alpha L^3}$ fires (i.e., the spike from $c_{j\alpha L^3 - 1}$ is received at $c_{j\alpha L^3}$ in round $\tau_j - 1$).

For every gate g_i in the circuit $\mathcal{A}$ in layer $j \geq 1$, denote by $out(g_i, \mathcal{A})$ the final state of g_i after receiving its inputs in the circuit $\mathcal{A}$. In addition, let q_i be the output neuron in the sub-network $\mathsf{Sync}(g_i)$, and let $\sigma_t(q_i, \mathcal{N})$ be the state of the neuron in round t when simulating the network $\mathcal{N}$. Our goal is to show that for every g_i, its corresponding output q_i in the network $\mathcal{N}$, has the same "output" as g_i in the circuit $\mathcal{A}$.

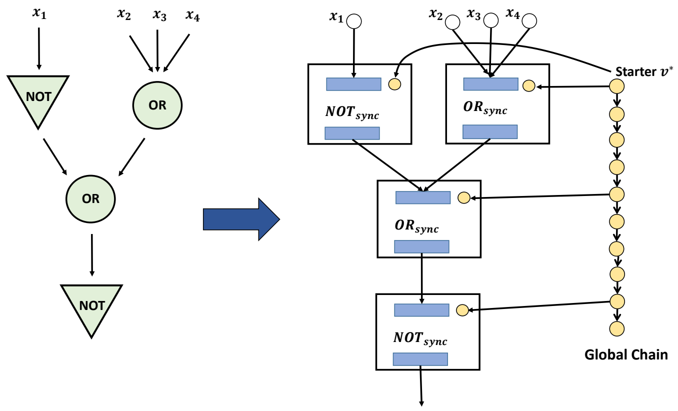

Figure 3 The transformation of the circuit on the left with 4 inputs and 3 layers. For each gate we add the corresponding synchronized sub-network, where we connect the input and output neurons of the sub-network according to the original circuit. In addition we introduce a global chain that activates the sub-networks in each layer after the previous layers have already finished the computation. The first neuron in the global chain is set to be the starter neuron which fires in the beginning of the simulation.

▷ Claim 33. For every layer $j \in \{1, \ldots, d\}$ and every gate g_i in layer j of circuit $\mathcal{A}$, its holds that: (i) If $out(g_i, \mathcal{A}) = 0$, then $\sigma_t(q_i, \mathcal{N}) = 0$ for every t, and (ii) If $out(g_i, \mathcal{A}) = 1$, then there exists $t \in [\tau_{j-1} + 1, \tau_j]$ such that $\sigma_t(q_i, \mathcal{N}) = 1$.

Proof. We prove by induction on the layer j. For $j = 1$, recall that the input neurons $x_{i,1}, \ldots, x_{i,k_i}$ of the sub-network $\mathsf{Sync}(g_i)$ are the input neurons of the network $\mathcal{N}$. Therefore, in round 0 in the simulation of $\mathcal{N}$, the sub-network $\mathsf{Sync}(g_i)$ has the same input as gate g_i in the circuit $\mathcal{A}$. Assume first that g_i is a NOT gate. Then the spike of the starter neuron c_0 arrived at the head chain v_i^* by round L. Combining with Observation 32 we get that if $out(g_i, \mathcal{A}) = 1$ then $\sigma_t(q_i, \mathcal{N}) = 1$ for some $t \in [L, L + 5L^3] \subseteq [1, \alpha L^3]$. In addition, if $out(g_i, \mathcal{A}) = 0$ then $\sigma_t(q_i, \mathcal{N}) = 0$ for every t. The case where g_i is an OR gate is even simpler and follows by Observation 13. Since the path from c_0 to $c_{\alpha L^3}$ in the chain C is of length αL^3, we have that $\tau_1 \geq \alpha L^3$. Therefore $[1, \alpha L^3] \subseteq [\tau_0 + 1, \tau_1]$, and the claim holds for $j = 1$.

For the induction step, let $j \geq 2$, and assume correctness up to layer $j - 1$. We now prove the claim for layer j. Let g_i be a gate in layer j. By Observation 32, the important thing to take care of regarding a NOT gate g is to make sure that its inputs have the correct states (i.e., as the corresponding states in $\mathcal{A}$) by the time that the head of the chain v_i^* in $\mathsf{Sync}(g)$ has received the spike from $c_{(j-1)\alpha L^3}$. Denote by $q_{i,1}, \ldots, q_{i,k_i}$ the output neurons of the sub-networks $\mathsf{Sync}(g_{i,1}), \ldots, \mathsf{Sync}(g_{i,k_i})$. By the induction assumption, for each $g_{i,h}$, if $out(g_{i,h}, \mathcal{A}) = 0$ then $\sigma_t(q_{i,h}) = 0$ for every t, and otherwise $\sigma_t(q_{i,h}) = 1$ for some $t \leq \tau_{j-1}$. Since the neurons $q_{i,1}, \ldots, q_{i,k_i}$ are the input neurons of g, it holds that the sub-network $\mathsf{Sync}(g)$ gets the same input as the input of g in $\mathcal{A}$, by round τ_{j-1} of the simulation of $\mathcal{N}$.

Now, assume that g_i is a NOT gate. Then, by round $\tau_{j-1} + L$, the head of the chain v_i^* has recieved the spike from $c_{(j-1)\cdot\alpha\cdot L^3}$. Combining with Observation 32, when $out(q_i, \mathcal{N}) = 1$, it holds that $\sigma_t(q_i, \mathcal{N}) = 1$ for some $t \in [\tau_{j-1} + 1, \tau_{j-1} + L + 5L^3]$. In addition, when $out(q_i, \mathcal{N}) = 0$ then $\sigma_t(q_i, \mathcal{N}) = 0$ for every t. Again, since the path from $c_{(j-1)\alpha L^3}$ to $c_{j\alpha L^3}$ in the chain C is of length αL^3, we have that $\tau_j \geq \tau_{j-1} + L + 5L^3$, and the claim follows. The case where g_i is an OR gate follows in a similar way by Observation 13. ◀

Lemma 15 follows by using Claim 33 with $j = d$, and noting that each output neuron z_i in $\mathcal{N}$ is the output neuron $q_{i'}$ for some sub-network $\mathsf{Sync}(g_{i'})$ where $g_{i'}$ is a gate in layer d of $\mathcal{A}$. This completes the correctness and the bound on the time overhead. We finally bound the size of the network. The network $\mathcal{N}$ consists of a chain of $O(dL^3)$ neurons, and a $\mathsf{Sync}(g_i)$ sub-network of size $O(L^2)$ for each gate g_i in $\mathcal{A}$. Therefore, there are overall $O(dL^3 + mL^2)$ auxiliary neurons.

B.3 Synchronization of a Single Deterministic Threshold Gate

We now turn to consider the synchronized implementation of a single deterministic threshold gate and prove Lemma 16.

Thanks to a result of [30], we can assume without loss of generality that the weights and bias values can be represented using binary vectors of length $\lceil \Delta \log \Delta \rceil$. Hastad [10] also showed that this bound is tight. In addition, we can also assume without loss of generality that $b(z) \geq 0$. The key part is to implement the single threshold gate by a Boolean circuit. This requires small adaptations from existing results in the area, specifically we will use the following known facts.

▷ Fact 34. [31, 38][Iterated Addition] Given two input binary vectors $\bar{x} = [x_1, \ldots, x_\Delta]$ and $\bar{y} = [y_1, \ldots, y_\Delta]$, there exists a Boolean circuit with $\text{poly}(\Delta)$ gates and $O(1)$ depth that outputs the binary representation of $\text{dec}(\bar{x}) + \text{dec}(\bar{y})$.

▶ **Corollary 35.** *[Multiple Iterated Addition] Given Δ input binary vectors $\bar{x}_1, \ldots, \bar{x}_\Delta$ where $\bar{x}_i \in \{0,1\}^m$ for some integer $m \geq 1$, there exists a Boolean circuit with $\text{poly}(\Delta, m)$ gates and $O(\log \Delta)$ depth that outputs the binary representation of $\sum_i \text{dec}(\bar{x}_i)$.*

▷ Observation 36 (Comparison). Given two input binary vectors $\bar{x} = [x_1, \ldots, x_\Delta]$ and $\bar{y} = [y_1, \ldots, y_\Delta]$, there exists a Boolean circuit with $\text{poly}(\Delta)$ gates and $O(1)$ depth that outputs 1 iff $\text{dec}(\bar{x}) \geq \text{dec}(\bar{y})$.

We are now ready to implement a threshold gate by a small depth Boolean circuit of polynomial size. This lemma explains the dependency in the largest in-degree Δ of the final synchronized solution.

▶ **Lemma 37.** *Given a threshold gate g with Boolean inputs $x_1, \ldots, x_\Delta$ with weights $w_1, \ldots, w_\Delta$, and an output neuron z with bias $b(z)$, there exists a Boolean circuit that computes g (i.e., outputs 1 iff $\sum w_i \cdot x_i \geq b(z)$) using $\text{poly}(\Delta)$ gates and depth $O(\log \Delta)$.*

Proof. Each input x_i is connected to $\ell = \lceil \Delta \cdot \log \Delta \rceil$ neurons $w_{i,1}, \ldots, w_{i,\ell}$, where the edge weight $w(x_i, w_{i,j})$ is 1 if the j^{th}-bit in w_i is 1 and 0 otherwise. We set the bias values to be $b(w_{i,j}) = 1$. Thus, the outgoing edge weights of x_i encode the binary representation of the weight w_i. As a result, once x_i fires in round τ, after at most L rounds, $w_{i,j}$ fires iff the j^{th} bit in the representation of w_i is 1. As we will see, those $\Delta^2 \cdot \log \Delta$ neurons $w_{1,1}, \ldots, w_{\Delta,\ell}$ will serve as the input layer to the circuit. In addition, we also represent the bias of z using ℓ neurons $b_1, \ldots, b_\ell$ that encode the binary representation of $b(z)$: the bias of $b_j = 1$ if the j^{th} bit in $b(z)$ is 0, and $b_j = -1$ otherwise. Let $\bar{x}_{pos} = \{x_i \mid w_i \geq 0\}$ and $\bar{x}_{neg} = \{x_i \mid w_i < 0\}$. In the same manner, let $W_{pos} = \sum\{w_i \mid w_i \geq 0\}$ and $W_{neg} = \sum\{|w_i| \mid w_i < 0\}$. We will use the Multiple Iterated Addition circuit of Corollary 35 to compute the binary representation of W_{pos} and $W_{neg} + b(z)$. Finally, we use the Comparison circuit of Observation 36 to compare those values, such that the output will be 1 iff $W_{pos} \geq W_{neg} + b(z)$, hence computing the function of the threshold gate. ◀

The final synchronous implementation of g is obtained by applying Lemma 15 on $\mathcal{C}$, i.e., $\mathsf{Sync}(g) \leftarrow \mathsf{Sync}(\mathcal{C})$. The construction uses a total $O(\log \Delta \cdot L^3 + \text{poly}(\Delta) \cdot L^2)$ auxiliary neurons, and computation time of $O(L^4 \cdot \log \Delta)$ rounds. This completes the proof of Lemma 16.

B.4 Probabilistic Threshold Gate

B.4.1 Description of the Boolean Circuit

The construction of the boolean circuit $\mathcal{A}$ approximating a probabilistic threshold gate is achieved using two main steps. First we sample an almost uniform random variable, then we use the sampled value in order to approximate a sample from the Logistic distribution.

Step 1: Sampling from the Almost Uniform Distribution. We introduce $k = 4\log(1/\epsilon)$ uniformly random gates, denoted as $r_1, \ldots r_k$. Hence, $\text{dec}(\bar{r})$ encodes an integer number that is uniformly sampled between 0 and $\lceil(1/\epsilon)^4\rceil$. In addition, we introduce k input bits (with fixed value) $a_1, \ldots a_k$ such that $\text{dec}(\bar{a}) = \lceil(1/\epsilon)^4\rceil$. Thus, the value $r' = \text{dec}(\bar{r})/\,\text{dec}(\bar{a})$ is sampled uniformly at random from the set $\{0, \epsilon^4, 2\epsilon^4, 3\epsilon^4, \ldots 1\}$.

Step 2: Sampling from the Almost Logistic Distribution. Next, we transform the sample r' from Step 1 into a sample from an almost Logistic distribution. This is done by using the method of inverse transform sampling. In our context, for a sample r u.a.r in $[0,1]$, the value $b + \ln(r/(1-r))$ is a sample from the Logistic distribution with mean b and scale 1. To compute the expression $b + \ln(r'/(1-r'))$ using a Boolean circuit, we approximate the $\ln(x)$ function (up to $\pm\,\text{poly}(\epsilon)$) using the first $O(\log 1/\epsilon)$ terms of the Taylor expansion around a point x_0 where $0 \le x_0 - x \le 1/2$.

▶ **Definition 38** (ϵ-Approximation of the $\ln(x)$ Function)**.** *Given $x > 0$ and a positive integer k, let $\widehat{\ln}_k(x)$ be the* ln*-approximation of x obtained by computing the first k terms of the Taylor expansion around a point x_0, where $0 \le x_0 - x \le 1/2$. When k is clear from the content we may omit it and simply write $\widehat{\ln}(x)$.*

The task of sampling from the (almost) Logistic distribution then boils into computing $f(r') = \widehat{\ln}_k(r'/(1-r'))$ with $k = \lceil 4\log 1/\epsilon \rceil$. We first use a Boolean circuit to distinguish between the case where $r' \le 1 - r'$, and the complementary case. Using the vectors $\bar{r}$ and $\bar{a}$, this can be done using integer operations and comparison as $r'/(1-r') = \text{dec}(\bar{r})/(\text{dec}(\bar{a}) - \text{dec}(\bar{r}))$. When $r' > 1 - r'$, we calculate $f(-r')$, and then either add or subtracts it from the bias b respectively.

In what follows, assume that $r' \le 1 - r'$, and therefore $r'/(1-r') \in [0,1]$. To pick the point x_0 around which the Taylor approximation is expended, we let $x_0 = 1/2$ when $r'/(1-r') \le 1/2$, and $x_0 = 1$ otherwise. This latter condition can also be easily checked with a Boolean circuit.

To finally be able to compute the function $\widehat{\ln}_k(x)$ using a Boolean circuit, we must ensure that all our operations are applied on *integers*. Therefore, instead of computing $f(r')$, we will be actually computing $q \cdot f(r')$ for some large enough constant q that guarantees that $q \cdot f(r')$ is an integer. Specifically, letting $q = (\text{dec}(\bar{a}) - \text{dec}(\bar{r}))^k$ does the job as the function $\widehat{\ln}_k(x)$ is a polynomial of degree k. This factor of q would not affect the correctness of the computation as it will be canceled out later on. Using the circuit for iterated addition [38, 31] and fast multipliers [9], we can compute $q \cdot f(r')$ using only integer addition and multiplication. The output of the final Boolean circuit is $\bar{y}$ where $\text{dec}(\bar{y}) = q \cdot b + q \cdot f(r')$. In the analysis section, we show that $\text{dec}(\bar{y})/q$ is sampled from a distribution that is $\text{poly}(\epsilon)$-close to the Logistic distribution with mean b.

Putting all Together: The Output Circuit. Let $w_1, \ldots, w_\Delta$ be the weights of the probabilistic threshold gate g. To cancel out the multiplication of q in the output bias value from the previous step, we multiply all the incoming weights by q as well. We can then use the construction of Lemma 16 for a deterministic threshold gate with weights $w'_1 = q \cdot w_1, \ldots, w'_\Delta = q \cdot w_\Delta$ and bias $b'' = \text{dec}(\bar{y})$. This completes the description of the construction.

B.4.2 Analysis and Proof of Lemma 17

We now turn to prove Lemma 17 and start with several auxiliary claims.

▷ Claim 39. Let $r_1, r_2 \in [0,1]$ such that $|r_1 - r_2| \le \epsilon^2$ and $\epsilon \le r_1 \le 1-\epsilon$, then

$$|\ln(r_1/(1-r_1)) - \ln(r_2/(1-r_2))| \le 2\epsilon \ .$$

Proof. By the definition of r_1 and r_2 we get the following inequalities:

$$\begin{aligned}
|\ln(r_1/(1-r_1)) - \ln(r_2/(1-r_2))| &= |\ln(r_1) - \ln(1-r_1) - \ln(r_2) + \ln(1-r_2)| \\
&\le |\ln(\frac{r_1+\epsilon^2}{r_1})| + |\ln(\frac{1-r_1+\epsilon^2}{1-r_1})| \\
&\le |\ln(1+\epsilon^2/r_1)| + |\ln(1+\epsilon^2/(1-r_1))| \\
&\le 2\ln(1+\epsilon) \le 2\cdot\epsilon \ ,
\end{aligned}$$

where the last inequality is due to the Taylor expansion of $\ln(1+x)$ around 0. ◀

Recall that given $x > 0$ and an integer $k > 0$, $\widehat{\ln}_k(x)$ is the ln-approximation of x obtained by computing the first k terms of the Taylor expansion around a point x_0 where $0 \le x_0 - x \le 1/2$.

▷ Claim 40. Fix $r_1, r_2 \in [0,1]$ such that $|r_1 - r_2| \le \epsilon^2$ and $\epsilon \le r_1 \le 1-\epsilon$, denote $\widehat{b}_1 = b + \ln(r_1/(1-r_1))$ and $\widehat{b}_2 = b + \widehat{\ln}(r_2/(1-r_2))$. Then, $|\widehat{b}_1 - \widehat{b}_2| \le 3\epsilon$.

Proof. Fix $x \in (0,1)$. Since $\widehat{\ln}(x)$ is obtained by using the first k terms in the Taylor expansion of $\ln(x)$ around x_0, we have that $|\ln(x) - \widehat{\ln}(x)| = \frac{1}{x_0^k} \cdot \frac{(x-x_0)^k}{k} \cdot \eta^k$, where $\eta \in [x, x_0]$. Since $x_0 \ge x$, also $x_0 \ge \eta$. As $|x - x_0| \le 1/2$, we get that $|\ln(x) - \widehat{\ln}(x)| \le (1/2)^k$. By plugging $k = \Theta(\log 1/\epsilon)$, we have that $|\ln(x) - \widehat{\ln}(x)| \le \epsilon$ for every x.

Thus, combining with Claim 39 we conclude the following:

$$\begin{aligned}
|\widehat{b}_1 - \widehat{b}_2| &= |b + \ln(r_1/(1-r_1)) - b - \widehat{\ln}(r_2/(1-r_2))| \qquad (5)\\
&\le |\ln(r_1/(1-r_1)) - \ln(r_2/(1-r_2)) + \epsilon| \\
&\le \epsilon + |\ln(r_1/(1-r_1)) - \ln(r_2/(1-r_2))| \le 3\cdot\epsilon \ .
\end{aligned}$$

◀

▷ Claim 41. Consider two threshold gates g_1, g_2 with the same weighted sum and bias values $b_1 \le b_2$ such that $b_2 - b_1 \le \epsilon$. Then $|\Pr[g_1 = 1] - \Pr[g_2 = 1]| \le \sqrt{\epsilon}$.

Proof. Let W be the weighted incoming sum to both g_1 and g_2. The probability that g_1 outputs 1 is $1/(1+e^{-(W-b_1)})$, and the probability that g_2 outputs 1 is $1/(1+e^{-(W-b_2)})$. The following holds:

$$\begin{aligned}\Pr[g_2=1] &= 1/(1+e^{-(W-b_2)}) \geq 1/(1+e^{-(W-b_1-\epsilon)}) = 1/(1+e^{\epsilon}e^{-(W-b_1)}) \\ &\geq \frac{1}{e^{\epsilon}\cdot(1+e^{-(W-b_1)})} \geq \frac{1}{(1+\sqrt{\epsilon})\cdot(1+e^{-(W-b_1)})} \\ &= (1-\sqrt{\epsilon})(1/(1+e^{-(W-b_1)})) \geq 1/(1+e^{-(W-b_1)}) - \sqrt{\epsilon} = \Pr[g_1=1]-\sqrt{\epsilon}\ .\end{aligned}$$

In the third inequality we use the fact that $e < (1+\sqrt{\epsilon})^{\frac{1}{\epsilon}}$ and thus $e^{\epsilon} < 1+\sqrt{\epsilon}$. On the other hand, since $b_2 \geq b_1$ it holds that

$$\Pr[g_2=1] = 1/(1+e^{-(W-b_2)}) \leq 1/(1+e^{-(W-b_1)}) = \Pr[g_1=1]\ .$$

Hence, we conclude that $|\Pr[g_2=1]-\Pr[g_1=1]| \leq \sqrt{\epsilon}$ as required. ◀

Analysis of Step 1. In the first step of the construction, since each uniformly random gate r_i is 1 with probability 1/2, the value $\mathrm{dec}(\bar{r})$ is a uniform sample in $\{0,1\ldots,(1/\epsilon)^4\}$. Therefore, $r' = \mathrm{dec}(\bar{r})/\mathrm{dec}(\bar{a}) = \epsilon^4\cdot\mathrm{dec}(\bar{r})$ is sampled uniformly at random from $\{0,\epsilon^4,2\epsilon^4,3\epsilon^4,\ldots 1\}$. By a simple coupling argument, sampling r' is equivalent to the process of sampling a uniform random variable $r_1 \in [0,1]$ and rounding it to the closest value of the form $i\cdot\epsilon^4$ for some integer i. In this manner, these two samples have an additive distance of at most ϵ^4.

Analysis of Step 2. Denote the probability z outputs 1 by q, and the probability u outputs 1 by p. Recall that g is the probabilistic gate and g' is the output gate of the Boolean circuit that approximates g.

In the second step, we compute $\mathrm{dec}(\bar{y}) = q\cdot(b+f(r'))$ where $q=(\mathrm{dec}(\bar{a})-\mathrm{dec}(\bar{r}))^k$ and $f(r') = \widehat{\ln}(r'/(1-r'))$. Then g' outputs 1 iff $\mathrm{dec}(\bar{y}) \leq W\cdot q$, or simply iff $b+f(r')\leq W$. Given that $r'\in[2\epsilon^2,1-2\epsilon^2]$ by Claim 40, $b'=b+f(r')$ satisfies that $|b^*-b'|\leq 3\epsilon^2$ where b^* is a true sample from the Logistic distribution with mean b. Therefore, the following holds.

$$\begin{aligned}\Pr[g'=1 \ \mid\ r'\in[2\epsilon^2,1-2\epsilon^2]] &= \Pr[W\geq b' \mid r'\in[2\epsilon^2,1-2\epsilon^2]] \\ &\leq \Pr[W+3\epsilon^2\geq b^* \mid r'\in[2\epsilon^2,1-2\epsilon^2]] \\ &= 1/(1+e^{-(W-b+3\epsilon^2)}),\end{aligned}$$

and in addition

$$\Pr[g'=1]\geq \Pr[W-3\epsilon^2\geq b^* \mid r'\in[2\epsilon^2,1-2\epsilon^2]] = 1/(1+e^{-(W-b-3\epsilon^2)}).$$

Recall that $\Pr[g=1]=\frac{1}{1+e^{-(W-b)}}$. By claim 41 we conclude that $|\Pr[g'=1]-\Pr[g=1]|\leq 3\epsilon$.

We note that $r'\in[2\epsilon^2,1-2\epsilon^2]$ with probability at least $1-4\epsilon^2$. Hence, we conclude that:

$$\Pr[g'=1]\leq \Pr[g'=1 \ \mid\ r'\in[2\epsilon^2,1-2\epsilon^2]]+4\epsilon^2 \leq \Pr[g=1]+3\epsilon+4\epsilon^2 = p+\Theta(\epsilon)\ ,$$

and on the other hand:

$$\begin{aligned}\Pr[g'=1] &\geq (1-4\epsilon^2)\Pr[g'=1 \ \mid\ r'\in[2\epsilon^2,1-2\epsilon^2]] \\ &\geq (1-4\epsilon^2)(\Pr[g=1]-3\epsilon)\geq \Pr[g=1]-\Theta(\epsilon)\ .\end{aligned}$$

Thus, $|\Pr[g=1]-\Pr[g'=1]| = O(\epsilon)$ as required.

Complexity. We assume that the bias and weights of the given probabilistic threshold gate g are polynomial in $1/\epsilon$. We first claim that with high probability of $1-\Theta(\epsilon)$, the approximate bias sampled from the almost Logistic distribution in also bounded by $\mathrm{poly}(1/\epsilon)$.

▷ Claim 42. Given that $|\mu| = \text{poly}(1/\epsilon)$, for a random variable x drown from the logistic distribution with mean μ it holds that $|x| = \text{poly}(1/\epsilon)$ with probability greater than $1 - \epsilon$.

Proof. By the definition of the Logistic CNF function it holds that

$$\Pr[x > 2\ln(1/\epsilon) + \mu] = 1 - \frac{1}{1 + e^{-2\ln(1/\epsilon) - \mu + \mu}} = \frac{\epsilon^2}{1 + \epsilon^2} < \epsilon^2 .$$

On the other hand

$$\Pr[x \leq -2\ln(1/\epsilon) + \mu] = \frac{1}{1 + e^{2\ln(1/\epsilon) - \mu + \mu}} = \frac{1}{1 + 1/\epsilon^2} < \epsilon^2 .$$

◀

Thus we can assume from now on that all integer numbers can be representing using $O(\text{poly}(1/\epsilon))$ bits. Using circuits for fast integers multiplication as described in [9] and iterated addition [38, 31], there exists a Boolean circuit computing $W \cdot q$ as well as $b \cdot q$ using $\text{poly}(\Delta, \log(1/\epsilon))$ gates and $\text{poly}(\log \Delta, \log(1/\epsilon))$ depth. When computing the polynomial $q \cdot \widehat{\ln}(\frac{r'}{1-r'})$ (of total degree $2k$), calculating each term requires $O(\log k)$ multiplicity operations. Since we have k summands, in total we use $k \cdot \log k$ multiplicity operations, each requires $O(k \cdot \log k \cdot 2^{O(\log^* k)})$ gates (and depth), and $\log k$ addition operations. The comparison circuits uses $\text{poly}(\log 1/\epsilon)$ gates and depth, and the final threshold gate circuit requires $\text{poly}(\Delta, \log 1/\epsilon)$ gates and depth $\text{poly}(\log \Delta, \log 1/\epsilon)$. We conclude that the Boolean circuit has $\text{poly}(\Delta, \log(1/\epsilon))$ gates and depth of $\text{poly}(\log \Delta, \log(1/\epsilon))$.

B.4.3 Synchronizing a Probabilistic Threshold Gate

In order to construct a synchronized neural network computing the Boolean Circuit described in Lemma 17, we use the construction for synchronized Boolean circuits as described in Lemma 15. We are left with describing the implementation of the random bits $\bar{r}$ and the constant bits $\bar{a}$.

- In order to represent $\bar{a}$, we introduce k neurons $a_1, \ldots, a_k$. If the i^{th} bit in the binary representation of $\text{dec}(\bar{a}) = (1/\epsilon)^4$ equals 1 we set the bias of a_i to be $b(a_i) = -1$ and otherwise we set the bias to be $b(a_i) = 1$. As a result, the neurons that represent the bits that are 1 in the binary representation fire on every round, and the other neurons idle thought the execution.
- In order to represent $\bar{r}$ we introduce k *spiking* neurons $r_1, \ldots, r_k$. For the computation to succeed, we need to sample each random variable r_i only once. Therefore, each neuron r_i has a very large bias $b(r_i) = \text{poly}(1/\epsilon)$ and an incoming edge from the starter neuron s with weight $w(s, r_i) = b(r_i)$. As a result, as long as r_i did not receive a spike from s, with high probability it does not fire. On the other hand, when neuron r_i receives a spike from the starter neuron v^* it fires with probability $1/2$.

B.5 The Complete Synchronization Scheme

Finally, we describe the synchronizer for a given neural network and prove Theorem 4. We start by describing the construction for a network of deterministic threshold gates. The adaptation to a network of spiking neurons is quite straightforward as discussed in the end of the section. The construction has two parts: a global pulse generator that can be used to synchronize many networks, and an adaptation of the given network $\mathcal{N}$ into a network $\mathsf{Sync}(\mathcal{N})$, see Figure 2. The *pulse generator* is implemented by a directed cycle

$PG = [c_0, \ldots, c_k]$ of length $k = O(L^4 \log \Delta)$. All neurons in PG have bias $b(c_i) = 1$. In addition, for every $i \geq 1$ neuron c_i has an incoming edge from neuron c_{i-1} with weight $w(c_{i-1}, c_i) = 1$, and the first neuron c_0 has an incoming edge from the last neuron c_k with weight $w(c_0, c_k) = 1$. The last neuron of the chain c_k will declare the end of each phase. We assume throughout that the simulation starts by a spike of the starter $v^* = c_0$.

Modifications to the Network $\mathsf{Sync}(\mathcal{N})$. The input layer and output layer in $\mathsf{Sync}(\mathcal{N})$ are *exactly* as in $\mathcal{N}$. We will now focus on the set of *auxiliary* neurons V in $\mathcal{N}$. The network $\mathsf{Sync}(\mathcal{N})$ contains the vertices V of the original network $\mathcal{N}$, and in addition, for each neuron $v_i \in V$ we add the following components to the network:

- A synchronized sub-network $\mathsf{Sync}(v_i)$ using Lemma 16 implementing the threshold gate defined by neuron v_i. The input neurons to the sub-network $\mathsf{Sync}(v_i)$ are the incoming neighbors of v_i in $\mathcal{N}$. The first neuron v_i^* in the internal chain of the sub-network $\mathsf{Sync}(v_i)$ has an incoming edge from the L^{th} neuron of PG cycle, namely c_L with weight 1 and bias $b(v_i^*) = 1$. Denote the output of the sub-network $\mathsf{Sync}(v_i)$ by v_i^{out}.
- An AND module AND_i whose output neuron is v_i. This module is implemented by a circuit of $\mathsf{OR}_{\mathsf{sync}}$ and $\mathsf{NOT}_{\mathsf{sync}}$ gates with three layers (using simple De-morgan rule). The AND_i module receives input from the neuron v_i^{out} and from the $(\alpha L^4)^{th}$ neuron in PG, $c_{\alpha L^4}$ where α is a large enough constant. The internal chains of the AND_i circuit receive input from neuron c_β in PG where $\beta = \alpha L^4 + L$, making sure the circuit begins the execution *after* receiving all its inputs[10].

Modifications to the Circuit Synchronization of Sec. 4.1. So far, we handled the synchronization of circuits. In order to handle general networks (e.g., that contains self-loops and recurrent edges), we need to apply small adaptations to the synchronized sub-networks of Sec. 4.1. Specifically, unlike circuits, in the execution of a network, certain neurons (or gates) might be activated several times. To be able to re-use the sync. sub-networks throughout the execution, we need to reset the states kept by their self-loops.

We therefore adapt the construction of the sync. sub-network presented in Section 4.1 to reset themselves at the end of their computation. For each $\mathsf{NOT}_{\mathsf{sync}}$ gate, we augment its internal chain by $3 \cdot L^2$ neurons, and the last neuron of this chain is connected to an inhibitor neuron v_r. The inhibitor v_r has outgoing edges of weight $-\infty$ to all neurons in the sub-network. Due to Claim 30, it holds that the inhibition by v_r (i.e., the round in which v_r fires) occurs *after* the output neuron has already fired. Observe that the timing of the inhibition by v_r is set in a way that guarantees that all gates in the sub-network will be idle from that point on (i.e., there will be no delayed spikes that arrive after this inhibition). For the sub-network $\mathsf{OR}_{\mathsf{sync}}$ which do not contain self-loops, no adaptation is needed.

B.6 Correctness

Throughout, we fix a synchronous execution Π_{sync} and an asynchronous execution Π_{async}. For every neuron v and phase p, define the beginning of phase p of v in the asynchronous execution $(r(v,p))$ as the round in which the p^{th} spike of c_0 is fired. I.e., the p^{th} phase of v is the time interval $[r(v,p), r(v,p+1))$. For every round p, let $V^+_{\mathsf{sync}}(p)$ be the set of neurons that fire in round p in Π_{sync} (i.e., the neurons with positive entries in σ_p). Similarly, let $V^+_{\mathsf{async}}(p)$ be the set of neurons that fire during phase p.

[10] We say that the circuit receives its input, if every gate in the first layer has received the signals from its incoming input.

▶ **Lemma 43.** *The networks* $\mathsf{Sync}(\mathcal{N})$ *and* $\mathcal{N}$ *have similar executions.*

In order to show the networks $\mathsf{Sync}(\mathcal{N})$ and $\mathcal{N}$ have similar executions, we show by induction on round (resp., phase) p that $V^+_{\mathsf{sync}}(p) = V^+_{\mathsf{async}}(p)$. For $p = 1$, let $V^+_{\mathsf{sync}}(0)$ be the neurons that fired at the beginning of the simulation in round 0. We will show that every neuron $v_i \in V$ fires in phase 1 iff $v_i \in V^+_{\mathsf{sync}}(1)$. For $v_i \in V$, the spikes from its incoming neighbors in $V^+_{\mathsf{sync}}(0)$ reach the sub-network $\mathsf{Sync}(v_i)$ by round L. The global chain in $\mathsf{Sync}(v_i)$ is then activated by the neuron c_L in some round $\tau \in [L, L^2]$. Therefore, by Lemma 16 there exists a constant γ such that v_i^{out} fires in some round $\tau_i \in [2, \tau + \gamma \cdot \log \Delta \cdot L^4]$ iff the output of the threshold function corresponding to v_i is 1, meaning that $v_i \in V^+_{\mathsf{sync}}(1)$. We next note that the first layer of the sub-network AND_i consists of two $\mathsf{NOT}_{\mathsf{sync}}$ sub-networks with input from $c_{\alpha \cdot L^4}$ and v_i^{out}. Hence, by Observation 32 as long as AND_i receives the information from $c_{\alpha \cdot L^4}$ and v_i^{out} before the activation of the global chain of the network AND_i in some round τ^* its output neuron fires by round $\tau^* + O(L^4)$ iff both v_i^{out} and $c_{\alpha \cdot L^4}$ fired.

The global chain of AND_i is activated by neuron c_β for $\beta = \alpha \cdot L^4 + L$ and therefore is indeed activated *after* AND_i receives the spike from $c_{\alpha L^4}$. In addition, we choose α such that $\alpha L^4 > L^2 + \gamma \cdot \log \Delta \cdot L^4$. Therefore the neuron c_β fires *after* round $\tau_i + L$, i.e. after AND_i received the spike from v_i^{out} as well. We conclude that v_i fires in some round $\tau'' \in [\beta, O(L^4)]$ iff v_i^{out} fires in round τ_i. We choose k to be large enough to make sure that c_k fires after round τ'' and therefore all neurons in $V^+(1)$ fired during the first phase.

Next, we assume that $V^+_{\mathsf{sync}}(p) = V^+_{\mathsf{async}}(p)$ and consider phase $p+1$. Let τ_p be the round that c_0 fired at the beginning of phase p and let τ_{p+1} be the round in which c_0 fired at the beginning of phase $p+1$. In addition, we denote the round in which $c_{\alpha L^4}$ fired during phase p by τ_α. By the induction assumption, neuron v_i fires between round τ_p and round τ_{p+1} iff $v_i \in V^+_{\mathsf{sync}}(p)$. Moreover, since the activation of the sub-network AND_i is performed by neuron c_β, every $v_i \in V^+_{\mathsf{sync}}(p)$ fires after round τ_α. We choose α to be large enough such that by round τ_α, all sub-networks $\mathsf{Sync}(v_i)$ have been reset due to the modification in the circuit synchronization. Hence, for neuron $v_i \in V$, the spikes from its incoming neighbors in $V^+_{\mathsf{async}}(p)$ reach $\mathsf{Sync}(v_i)$ after the sub-network has already been reset. Thus, when the global chain of the sub-network $\mathsf{Sync}(v_i)$ is activated by the neuron c_L in round $\tau_L \in [\tau_{p+1} + L, \tau_{p+1} + L^2]$, the sub-network $\mathsf{Sync}(v_i)$ received spikes from the incoming neighbors of v_i in $V^+_{\mathsf{async}}(p)$. Combining with Lemma 16 we conclude that v_i^{out} fires in round $\tau_i \in [\tau_{p+1} + L, \tau_{p+1} + L^2 + \gamma \cdot \log \Delta \cdot L^4]$ iff $v_i \in V^+_{\mathsf{sync}}(p+1)$. Thus, when neuron c_β fires in phase $p+1$, the sub-network AND_i has received the spikes from both v_i^{out} and $c_{\alpha L^4}$. Since the global chain of AND_i is activated by the neuron c_β, we conclude that v_i fires in some round $\tau^* \in [\tau_{p+1} + \beta, \tau_{p+1} + \Theta(L^4)]$, iff $v_i \in V^+_{\mathsf{sync}}(p+1)$. Choosing k to be large enough, τ^* occurs before c_k fires and ends the phase.

Synchronization of a Spiking Neural Network. We next explain the adaptation of the construction given a network of spiking neurons $\mathcal{N}$. Let n be the number of auxiliary neurons in $\mathcal{N}$ and let t be the number of rounds. Each spiking neuron implemented by a probabilistic threshold gate can be made synchronized using Cor. 19 where we use an error parameter of $\epsilon = 1/\operatorname{poly}(n, t)$. Thus, The network $\mathsf{Sync}(\mathcal{N})$ consists of $\operatorname{poly}(\Delta, \log n \cdot t) \cdot L^4 \cdot n$ auxiliary neurons and uses $\operatorname{poly}(\log \Delta, \log n \cdot t) \cdot L^5$ rounds.

To compare the simulation of the given spiking neural network $\mathcal{N}$ and the synchronized network $\mathsf{Sync}(\mathcal{N})$, we fix the randomness used by $\mathcal{N}$ throughout the simulation and use these coins when simulated the network $\mathsf{Sync}(\mathcal{N})$. For neuron $v \in V$ and round $\tau \geq 1$, by Cor. 19, with probability at least $1 - 1/\operatorname{poly}(n \cdot t)$ it holds that $v \in V^+_{\mathsf{sync}}(\tau)$ iff $v \in V^+_{\mathsf{async}}(\tau)$. By applying the union bound over all n neurons and t rounds of the simulation, we conclude that with high probability $\mathcal{N}$ and $\mathsf{Sync}(\mathcal{N})$ have similar executions.

C Synchronization in the Node-Delay Model

C.1 Network Dynamics in the Node Delay Setting

Network evolution proceeds in *seconds*, namely, a sufficiently small time unit. For a given integer $T \geq 1$, the dynamics is specified by a node-delay function $t : V \to \mathbb{N}_{\leq T}$ interpreted as follows: the round duration on each neuron v consists of $t(v)$ seconds. Specifically, the i^{th} round of v is defined by the time interval $R_i(v) = [(i-1)t(v)+1, i \cdot t(v)]$ for every $i \geq 1$. All spikes are assumed to arrive with a delay of a single second[11]. For the neuron v and integer i, the set of spikes received at v during its i^{th} round is given by

$$A(v,i) = \{(u, j \cdot t(u)) \ \mid \ j \cdot t(u) + 1 \in R_i(v)\}.$$

The state of v in its i-round (i.e., at the second $i \cdot t(v)$) is given by:

$$\text{pot}(v,i) = \sum_{(u,j \cdot t(u)) \in A(v,i)} w(u,v) \cdot \sigma_j(v) \ - \ b(v) \ \text{ and } \ \sigma_i(v) = 1 \text{ iff } \text{pot}(v,i) \geq 0 \ . \tag{6}$$

If v is a probabilistic threshold gate then it fires in second $i \cdot t(v)$ with probability $p(v,i) = \frac{1}{1+e^{-\text{pot}(v,i)}}$.

▶ **Definition 44** (The T-bounded Node-Delay Setting)**.** *We are given a network $\mathcal{N}$ and an integer T. It is assumed the network contains a special neuron, the* starter*, that fires in the first round of the simulation. The dynamic is determined by a node-delay function $t : V \to \mathbb{N}_{\leq T}$. This function t can be chosen arbitrarily.*

▶ **Definition 45** (Computation of a Boolean Function in the T-bounded Node-Delay Setting)**.** *Let $f : \{0,1\}^n \to \{0,1\}^k$ be a Boolean function. A network $\mathcal{N}$ with n input neurons $x_1, \ldots, x_n$ and k output neurons $z_1, \ldots, z_k$* computes *f in this setting if for every T-bounded function $t : V \to \mathbb{N}_{\leq T}$ and for every fixed possible assignment to the input neurons $b_1, \ldots, b_n$ the following holds: (i) If $f_i(b_1, \ldots, b_n) = 1$, then there exists a round in which z_i fires, where $f_i(\cdot)$ is the i^{th} bit in the output of f. (ii) If $f_i(b_1, \ldots, b_n) = 0$ then z_i does not fire throughout the entire execution.*

Synchronizers for the Node-Delay. A synchronizer ν is an algorithm that gets as input a network $\mathcal{N}$ and integer T, and outputs a network $\mathcal{N}' = \mathsf{sync}_V(\mathcal{N}, T)$ that contains all the neurons of $\mathcal{N}$, plus additional auxiliary neurons. One of the auxiliary neurons in $\mathcal{N}'$ is a *starter* neuron that fires in the *first* round of the simulation. The network $\mathcal{N}'$ works in the asynchronous setting and should have *similar execution* to $\mathcal{N}$ in the sense that for every neuron $v \in V(\mathcal{N})$, the firing pattern of v in the asynchronous network should be similar to the one in the synchronous network. The output network $\mathcal{N}'$ simulates each round of the network $\mathcal{N}$ by a phase.

▶ **Definition 46** (Phases)**.** *We partition the execution of $\mathcal{N}'$ into phases $1, 2, \ldots,$ using a function $r : V(\mathcal{N}) \times \mathbb{N} \to \mathbb{N}$ that defines the beginning of phase p. Hence, the p^{th} phase is the round interval $[r(v,p), r(v,p+1))$.*

[11] As discussed in the introduction, this model can be generalized to support both edge-delays and node-delays, to isolate the node-delay effect we assume that all edges have latency of 1.

▶ **Definition 47** (Similar Executions (Deterministic Networks)). *The* synchronous *execution* Π *of a deterministic network* $\mathcal{N}$ *is specified by a list of states* $\Pi = \{\sigma_1, \ldots, \}$ *where each* σ_i *is a binary vector describing the firing status of the neurons in round* i*. The* asynchronous *execution of the network* $\mathcal{N}' = \mathsf{sync}_V(\mathcal{N}, T)$ *with a node-delay function* $t : V \to \mathbb{N}_{\leq T}$ *denoted by* $\Pi'(t)$ *is defined analogously only when applying the asynchronous dynamic. The execution* $\Pi'(t)$ *is divided into phases according the a function* $r : V(\mathcal{N}) \times \mathbb{N} \to \mathbb{N}$.

The network $\mathcal{N}$ *and the pair* $\langle \mathcal{N}', t \rangle$ *have a* ***similar execution*** *if* $V(\mathcal{N}) \subseteq V(\mathcal{N}')$*, and in addition, a neuron* $v \in V(\mathcal{N})$ *fires in round* p *in the execution* Π *iff* v *fires during phase* p *in* $\Pi'(t)$.

The networks $\mathcal{N}$ *and* $\mathcal{N}'$ *are* ***similar*** *if* $\mathcal{N}$ *and* $\langle \mathcal{N}', t \rangle$ *have a similar execution for every node-delay function* t.

As for the edge-delay model, the extension for randomized networks is made by fixing the random bits in the simulation of the input network.

C.2 Reduction to the Edge-Delay Model: A Simulation Result

Given a neural network $\mathcal{N}$ and an integer parameter T, our goal is to construct a network $\mathcal{N}_R = \mathsf{sync}_V(\mathcal{N}, T)$ in the T-bounded node-delay model that behaves similarly to $\mathcal{N}$, i.e., that $\mathcal{N}$ and $\mathcal{N}_R$ are similar according to Definition 47.

Given the network $\mathcal{N}$ and the delay bound T, we start by computing the network $\mathcal{N}_{\mathcal{L}} = \mathsf{sync}_E(\mathcal{N}, L)$ with $L = 5T^2$. The desired $\mathcal{N}_R = \mathsf{sync}_V(\mathcal{N}_T)$ is obtain by changing some of the edge weights in $\mathcal{N}_{\mathcal{L}}$. Our proof of correctness is based on similarity between a network in the node-delay model and a network in the edge-delay model.

Similarity between the networks $\mathcal{N}_R$ and $\mathcal{N}_{\mathcal{L}}$. Fix integer parameters T, L. Given an edge-delay network $\mathcal{N}_{\mathcal{L}}$, a latency function $\ell : E(\mathcal{N}_{\mathcal{L}}) \times \mathbb{N} \to \mathbb{N}_{\leq L}$, a node-delay network $\mathcal{N}_R$ on the same neuron set and a node-delay function $t : V(\mathcal{N}_R) \to \mathbb{N}_{\leq T}$, we want to define similarity between the simulations $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$ and $\langle \mathcal{N}_R, t \rangle$, where both simulations use the same initial configuration.

This notion of similarity is based on defining different time scales in each of the simulations. Specifically, for every $i \geq 1$ and neuron $u \in V$ the time window $R_i(u)$ will be the time that u collects spikes for its round i in the simulation of $\langle \mathcal{N}_R, t \rangle$. Moreover, for every $i \geq 0$ the time window $L_i(u)$ correspond to the firing period of round i of u in the simulation of $\langle \mathcal{N}_R, t \rangle$, where

$$R_i(u) = [(i-1) \cdot t(u) + 1, i \cdot t(u)] \text{ and } L_i(u) = [i \cdot T \cdot t(u), i \cdot T \cdot t(u) + (T \cdot t(u) - 1)].$$

Furthermore, for every second τ_R in the simulation of $\langle \mathcal{N}_R, t \rangle$ we will have the corresponding block $B_{\tau_R} = [\tau_R \cdot T, \tau_R \cdot T + (T-1)]$ in the simulation of $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$. For the simulation of $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$ define for every neuron u and $i \geq 0$:

$$\sigma_i(u, \mathcal{N}_{\mathcal{L}}) = \begin{cases} 1 & u \text{ is strong and } u \text{ fires in every } \tau_{\mathcal{L}} \in L_i(u) \\ 1 & u \text{ is weak and } u \text{ fires in } \tau_{\mathcal{L}} \in L_i(u) \text{ only for } \tau_{\mathcal{L}} = i \cdot T \cdot t(u) \\ 0 & u \text{ never fires in } L_i(u) \\ \emptyset & \text{Otherwise.} \end{cases}$$

For the simulation of $\langle \mathcal{N}_R, t \rangle$ define for every neuron u and $i \geq 0$:

$$\sigma_i(u, \mathcal{N}_R) = \begin{cases} 1 & u \text{ fires in round } i \text{ of } u \text{ (i.e. in the second } i \cdot t(u)) \\ 0 & \text{Otherwise.} \end{cases}$$

▶ **Definition 48.** *The simulations $\langle \mathcal{N}_R, t\rangle, \langle \mathcal{N}_\mathcal{L}, \ell\rangle$ are* similar*, denoted as $\langle \mathcal{N}_R, t\rangle \sim \langle \mathcal{N}_\mathcal{L}, \ell\rangle$, if for every neuron u and $i \geq 0$ it holds that $\sigma_i(u, \mathcal{N}_\mathcal{L}) = \sigma_i(u, \mathcal{N}_R)$.*

A network $\mathcal{N}_\mathcal{L}$ in the L-bounded edge-delay model and a network $\mathcal{N}_R$ in the T-bounded node-delay model are similar*, denoted by $\mathcal{N}_\mathcal{L} \sim \mathcal{N}_R$, if for every node-delay function $t : V(\mathcal{N}_R) \to \mathbb{N}_{\leq T}$ there exists a latency function $\ell : E(\mathcal{N}_\mathcal{L}) \times \mathbb{N} \to \mathbb{N}_{\leq L}$ such that $\langle \mathcal{N}_R, t\rangle \sim \langle \mathcal{N}_\mathcal{L}, \ell\rangle$.*

The key simulation lemma used in the synchronization scheme is as follow:

▶ **Lemma 49.** *Given a network $\mathcal{N}_\mathcal{L}$ in the L-bounded edge delay model such that:*
1. *$b(u) > 0$ for every neuron u.*
2. *Every weak neuron v has no self-loop.*
3. *There is no edge from a strong neuron to a strong neuron.*
4. *Every negative edge has weight $-\infty$.*
5. *For every neuron u, either any excitatory incoming neighbor of u is weak, or any excitatory incoming neighbor of u is strong.*
6. *Let v be a strong incoming neighbor of a neuron u, and let f be an inhibitor. Then if f has an edge to v, it also has an edge to u.*

Then there exists a network $\mathcal{N}_R$ in the T-bounded node-delay model with $T \leq \sqrt{L/5}$ with $V(\mathcal{N}_R) = V(\mathcal{N}_\mathcal{L})$ such that $\mathcal{N}_R$ and $\mathcal{N}_\mathcal{L}$ are similar.

Defining the node-delay network $\mathcal{N}_R$. The network $\mathcal{N}_R$ is exactly as $\mathcal{N}_\mathcal{L}$, up to small adaption of the weights. Denote by $w_\mathcal{L} : V \to \mathbb{R}$ the weight function of the network $\mathcal{N}_\mathcal{L}$. Define the weight function w_R of $\mathcal{N}_R$ as

$$w_R(v,u) = \begin{cases} T \cdot w_\mathcal{L}(v,u) & v \neq u,\ v \text{ is strong,} \\ w_\mathcal{L}(v,u) & \text{Otherwise.} \end{cases}$$

Correctness. We will show that $\mathcal{N}_\mathcal{L}$ and $\mathcal{N}_R$ are similar. Fix a node-delay function $t : V \to \mathbb{N}_{\leq T}$. First, we define the corresponding latency function ℓ and prove it is valid, i.e. that ℓ is nice and $\ell(v,u,\tau) \in [1, L]$ for every neurons v, u and round τ. Then, we restate Lemma 49 in order to prove its correctness by induction on the round.

Definition of the latency function ℓ. First, set the latency of self-spikes to be of value 1. For a neuron u, we say that u is *weak-incoming* if any excitatory incoming neighbor of u is weak, and we say that u is *strong-incoming* if any excitatory incoming neighbor of u is strong. Note that by property 5, every neuron u is either weak-incoming or strong-incoming. For a strong-incoming neuron u, an inhibitor v and $\tau_\mathcal{L} \geq 0$, set $\ell(v,u,\tau_\mathcal{L}) = 2T^2 + 1$. Now consider the remaining spikes, which are either positive spikes, or spikes to a weak-incoming neuron u.

For every $\tau_\mathcal{L} \geq 0$ define the latency value for the spike event $\langle v, u, \tau_\mathcal{L}\rangle$ as follows. Let j be an integer satisfying that $\tau_\mathcal{L} \in L_j(v)$, and let i be such that $j \cdot t(v) + 1 \in R_i(u)$, hence $(v, j \cdot t(v)) \in A(u,i)$.

If v is weak, then for $\tau_\mathcal{L} = j \cdot T \cdot t(v)$ set $\ell(v,u,\tau_\mathcal{L}) = i \cdot T \cdot t(u) - \tau_\mathcal{L}$. That is, the spike $\langle v,u,\tau_\mathcal{L}\rangle$ is scheduled to arrive in the first round of $L_i(u)$. For $\tau_\mathcal{L} > j \cdot T \cdot t(v)$, set $\ell(v,u,\tau_\mathcal{L}) = 1$. Otherwise, if v is strong, consider the following argument. For every second $\tau_\mathcal{L}$ in the edge-latency simulation, let τ_R be the second in the node-delay simulation such that $\tau_\mathcal{L} \in B_{\tau_R}$.

- **Case (I):** there exists a second in $[\tau_R + 1, \tau_R + 2T]$ such that u fires in the node-delay simulation, let τ'_R be the first such second. Set $\ell(v,u,\tau_\mathcal{L}) = \tau'_R \cdot T - \tau_\mathcal{L}$, that is schedule $\langle v,u,\tau_\mathcal{L}\rangle$ to arrive in round $\tau'_R \cdot T$.

- **Case (II):** case I does not apply, and there is an inhibitor f which is an incoming neighbor of u, and a second $\tau'_R \in [\tau_R - T, \tau_{\mathcal{L}} + 2T]$ such that f fires in τ'_R in the node-delay simulation. Then for such τ'_R, set $\ell(v, u, \tau_{\mathcal{L}}) = \tau'_R \cdot T + (2T^2 + 1) - \tau_R$, that is schedule $\langle v, u, \tau_{\mathcal{L}} \rangle$ to arrive in round $\tau'_R \cdot T + (2T^2 + 1)$.
- **Case (III):** neither case (I) nor case (II) apply. Set $\ell(v, u, \tau_{\mathcal{L}}) = 1$.

The intuition is that for a positive spike in the edge-delay simulation, we look for a round such that u is supposed to fire in the next $2T^2$ rounds. If we cannot find one, we want to send the spike to a round that we know it will not activate u. This is a round in which u receives a negative spike (since negative spikes are of weight $-\infty$). If such round also does not exist, it implies that the total weight of positive incoming neighbors of u that fired in round $\tau_{\mathcal{L}}$ is low, and we can schedule all these spikes to arrive together in $\tau_{\mathcal{L}} + 1$ without activating u. We next show that ℓ is valid.

▷ Claim 50. ℓ is a valid latency function for $\mathcal{N}_{\mathcal{L}}$.

Proof. First, since all self-spikes have latency value 1, ℓ is nice. For a negative spike $\langle v, u, \tau_{\mathcal{L}} \rangle$ such that u is strong-incoming, it holds that $\ell(v, u, \tau) = 2T^2 + 1 < L$. Therefore we are left to show validity for positive spikes, and for negative spikes that are fired towards a weak-incoming neuron. Consider a spike $\langle v, u, \tau_{\mathcal{L}} \rangle$, and let j be an integer satisfying that $\tau_{\mathcal{L}} \in L_j(v)$. Furthermore, let i be an integer such that $j \cdot t(v) + 1 \in R_i(u)$.

Next, assume that v is weak. We distinguish between two cases depending whether $\tau_{\mathcal{L}}$ is the first round in the block or not. For $\tau_{\mathcal{L}} = i \cdot T \cdot t(v)$ we have $\ell(v, u, \tau_{\mathcal{L}}) = i \cdot T \cdot t(u) - \tau_{\mathcal{L}}$. Recall that $R_i(u) = [(i-1) \cdot t(u) + 1, i \cdot t(u)]$, thus $j \cdot t(v) + 1 \le i \cdot t(u)$, and $\ell(v, u, \tau_{\mathcal{L}}) = T \cdot (i \cdot t(u) - j \cdot t(v)) \ge T \ge 1$. Furthermore $j \cdot t(v) + 1 \ge (i-1) \cdot t(u) + 1$, hence $i \cdot t(u) - j \cdot t(v) \le t(u) \le T$, and $\ell(v, u, \tau_{\mathcal{L}}) \le T \cdot (i \cdot t(u) - j \cdot t(v)) \le L$. Otherwise, i.e. for $\tau_{\mathcal{L}} \ge i \cdot T \cdot t(v)$, it holds that $\ell(v, u, \tau_{\mathcal{L}}) = 1$, and thus $\ell(v, u, \tau_{\mathcal{L}}) \in [1, L]$.

It remains to consider the case where v is strong. Let τ_R be the second in the node-delay simulation such that $\tau_{\mathcal{L}} \in B_{\tau_R}$. Consider the definition of ℓ for a spike $\langle v, u, \tau_{\mathcal{L}} \rangle$. In case (I), we have $\ell(v, u, \tau_{\mathcal{L}}) = \tau'_R \cdot T - \tau_{\mathcal{L}}$, and since $\tau'_R \in [\tau_R + 1, \tau_R + 2T]$ it holds that $1 \le \tau'_R \cdot T - \tau_{\mathcal{L}} \le 2T^2 < L$. In case (II), since $\tau'_R \in [\tau_R - T, \tau_R + 2T]$, we have that $\ell(v, u, \tau_{\mathcal{L}}) = \tau'_R \cdot T + (2T^2 + 1) - \tau_{\mathcal{L}} \in [1, 5T^2]$. Finally, in case (III) we simply have $\ell(v, u, \tau_{\mathcal{L}}) = 1$. Hence, in all cases it holds that $\ell(v, u, \tau_{\mathcal{L}}) \in [1, L]$. ◀

In order to show that $\langle \mathcal{N}_R, t \rangle \sim \langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$, we restate the condition for similarity in the following lemma. We then prove the lemma by induction on the round $\tau_{\mathcal{L}}$.

▶ **Lemma 51** (Restating Lemma 49). *For every round $\tau_{\mathcal{L}} \ge 0$ of the simulation $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$ and for every neuron u, let i be such that $\tau_{\mathcal{L}} \in L_i(u)$. Then:*

1. *If $\sigma_i(u, \mathcal{N}_R) = 1$:*
   - *If $\tau_{\mathcal{L}} = i \cdot T \cdot t(u)$ then u fires in $\tau_{\mathcal{L}}$.*
   - *If $\tau_{\mathcal{L}} > i \cdot T \cdot t(u)$ then u fires iff u is strong.*
2. *If $\sigma_i(u, \mathcal{N}_R) = 0$ then u does not fire in $\tau_{\mathcal{L}}$.*

For the base case $\tau_{\mathcal{L}} = 0$, the correctness follows the fact that both simulations have the same starting configuration. Now, let $\tau_{\mathcal{L}} \ge 1$ and assume correctness for every $\tau'_{\mathcal{L}} \le \tau_{\mathcal{L}} - 1$. Fix a neuron u and let i be an integer such that $\tau_{\mathcal{L}} \in L_i(u)$. We start with a useful auxiliary claim.

▷ Claim 52. Let u be a weak-incoming neuron, v an incoming neighbor of u, and $\tau'_{\mathcal{L}} \ge 0$. Furthermore, let j be such that $\tau'_{\mathcal{L}} \in L_j(v)$, and i such that $j \cdot t(v) + 1 \in R_i(u)$. Then the spike $\langle v, u, \tau'_{\mathcal{L}} \rangle$ occurs and arrives to u in round $\tau_{\mathcal{L}} = i \cdot T \cdot t(u)$ in the simulation $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$ iff

$\tau'_{\mathcal{L}} = j \cdot t(v)$ and the spike $\langle v, u, j \cdot t(v)\rangle$ occurs and arrives to u in $R_i(u)$ in the simulation $\langle \mathcal{N}_R, t\rangle$.

Proof of Claim 52. Since u is weak-incoming v is weak, then by the induction assumption for $\tau'_{\mathcal{L}}$ and the definition of ℓ, the spike event $\langle v, u, \tau'\rangle$ occurs and arrives in round $\tau_{\mathcal{L}}$ iff there exists j such that $\tau'_{\mathcal{L}} = j \cdot T \cdot t(v)$ and $\sigma_j(v, \mathcal{N}_R) = 1$. This happens iff in the simulation of $\langle \mathcal{N}_R, t\rangle$ the spike event $\langle v, u, j \cdot t(v)\rangle$ occurs and arrives to u in $R_i(u)$. ◀

We split the proof of Lemma 51 into two cases.

Case 1: u is weak-incoming. Assume $\tau_{\mathcal{L}} = i \cdot T \cdot t(u)$, we want to show that u fires in round $\tau_{\mathcal{L}}$ iff $\sigma_i(u, \mathcal{N}_R) = 1$. By Claim 52, we get that the mapping $\langle v, u, j \cdot T \cdot t(v)\rangle \mapsto \langle v, u, j \cdot t(v)\rangle$ is a bijection between the set of non self-spikes that u receives in $\tau_{\mathcal{L}}$ in the simulation $\langle \mathcal{N}_{\mathcal{L}}, \ell\rangle$ and the set of non-self spikes that u receives in $R_i(u)$ in the simulation $\langle \mathcal{N}_R, t\rangle$. As for self-spikes, note that if u is weak it has no self-loop. If u is strong, then by the induction assumption u fires in $\tau_{\mathcal{L}} - 1$ iff $\sigma_{i-1}(u, \mathcal{N}_R)$. Thus, u receives the self-spike $\langle u, u, \tau_{\mathcal{L}} - 1\rangle$ in $\tau_{\mathcal{L}}$ iff it receives the self-spike $\langle u, u, (i-1) \cdot T \cdot t(u)\rangle$ in $R_i(u)$. Since $w_{\mathcal{L}}(v, u) = w_R(v, u)$ for every weak neuron v and for $v = u$, we get that the total spike weight that u receives in $\tau_{\mathcal{L}}$ equals to the total spike weight it receives in $R_i(u)$. Thus, u fires in round $\tau_{\mathcal{L}}$ iff $\sigma_i(u, \mathcal{N}_R) = 1$.

Now, assume $\tau_{\mathcal{L}} > i \cdot T \cdot t(u)$ and that either v is weak, or v is strong and $\sigma_i(u, \mathcal{N}_R) = 0$. We want to show that u does not fire. Note that if v is weak then it has no self-loop, and if v is strong and $\sigma_i(u, \mathcal{N}_R) = 0$ then by the induction assumption for $\tau_{\mathcal{L}} - 1$, u does not fire in $\tau_{\mathcal{L}} - 1$. Thus, in both cases u does not receive a self-spike in $\tau_{\mathcal{L}}$. Furthermore, u has no strong neighbors, therefore by Claim 52 u does not receive any positive spikes from incoming neighbors. Since $b(u) > 0$, u does not fire in $\tau_{\mathcal{L}}$.

Finally, assume $\tau_{\mathcal{L}} > i \cdot T \cdot t(u)$, and assume u is strong and $\sigma_i(u, \mathcal{N}_R) = 1$. We want to show that u fires. Note that by Claim 52, u does not receive a negative spike in $\tau_{\mathcal{L}}$. Furthermore, since $\sigma_i(u, \mathcal{N}_R) = 1$ by the induction assumption for $\tau_{\mathcal{L}} - 1$, u fires in $\tau_{\mathcal{L}} - 1$ and therefore u receives a self-spike in $\tau_{\mathcal{L}}$. Since $w_{\mathcal{L}}(u, u) \geq b(u)$, u fires in $\tau_{\mathcal{L}}$.

Case 2: u is strong-incoming. By the properties of $\mathcal{N}_{\mathcal{L}}$ there is no edge between strong neurons, and weak neurons have no self-loop. Hence u is weak and has no self-loop. We handle separately the following sub-cases:

Case 2.1: $\sigma_i(u, \mathcal{N}_R) = 1$ and $\tau_{\mathcal{L}} = i \cdot T \cdot t(u)$. We want to show that u fires in $\tau_{\mathcal{L}}$. Let $\langle v, u, j \cdot t(v)\rangle$ be a positive spike in the simulation $\langle \mathcal{N}_R, t\rangle$ that arrives to u in $R_i(u)$, and let $\tau'_{\mathcal{L}}$ be one of the T rounds $[j \cdot T \cdot t(v), j \cdot T \cdot t(v) + (T-1)]$. Since v is strong then by the induction assumption for $\tau'_{\mathcal{L}}$ v fires in $\tau'_{\mathcal{L}}$, and therefore the spike event $\langle v, u, \tau'_{\mathcal{L}}\rangle$ occurs in the simulation $\langle \mathcal{N}_{\mathcal{L}}, \ell\rangle$. We now show that $\langle v, u, \tau'_{\mathcal{L}}\rangle$ arrives to u in $\tau_{\mathcal{L}}$, according to the definition of ℓ for spikes from strong neurons.

Since $\sigma_i(u, \mathcal{N}_R) = 1$, u fires in the second $r_R = i \cdot t(u)$ in the simulation $\langle \mathcal{N}_R, t\rangle$. Note that $j \cdot t(v) + 1 \in R_i(u)$ implies that $i \cdot t(u) - j \cdot t(v) \leq T$. Hence in particular $i \cdot t(u) \in [j \cdot t(v) + 1, j \cdot t(v) + 2T]$. Let $r'_R \in [j \cdot t(v) + 1, j \cdot t(v) + 2T]$ with $r'_R < i \cdot t(u)$. Note that $r'_R \in R_i(u)$, therefore r'_R is not an end of a round of u. Hence u does not fire in r'_R. Therefore the second $r_R = i \cdot t(u)$ is the first second in $[j \cdot t(v) + 1, j \cdot t(v) + 2T]$ that u fires, and due to the definition of ℓ the spike $\langle v, u, \tau'_{\mathcal{L}}\rangle$ arrives in round $\tau_{\mathcal{L}}$.

Now, let f be an inhibitory incoming neighbor of u. By the definition of ℓ, a spike from f to u can arrive only in a round of the form $\tau'_R \cdot T + T^2 + 1$ for some second τ'_R, which is not a multiplicity of T. Note that $\tau_{\mathcal{L}} = i \cdot T \cdot t(u)$ is a multiplicity of T. Thus u does not receive a negative spike in $\tau_{\mathcal{L}}$.

We get that in round $\tau_{\mathcal{L}}$, u receives only positive spikes in $\tau_{\mathcal{L}}$, and for every positive spike

$\langle v, u, j \cdot t(v)\rangle$ that arrives to u in $R_i(u)$ and every $\tau'_{\mathcal{L}} \in [j \cdot T \cdot t(v), j \cdot T \cdot t(v) + (T-1)]$, u receives a spike $\langle v, u, \tau'_{\mathcal{L}}\rangle$. Since $w_R(v,u) = T \cdot w_{\mathcal{L}}(v,u)$ for every strong v and $[j \cdot T \cdot t(v), j \cdot T \cdot t(v) + (T-1)]$ contains T rounds, we get that the total spike weight that u receives in $\tau_{\mathcal{L}}$ is at least the total spike weight it receives in $R_i(u)$ in the node-delay simulation. Since $\sigma_i(u, \mathcal{N}_R) = 1$, u receives in $R_i(u)$ a spike weight of at least $b(u)$, which implies the same for round $\tau_{\mathcal{L}}$ in the edge-delay simulation. Thus u fires in round $\tau_{\mathcal{L}}$.

Case 2.2: $\sigma_i(u, \mathcal{N}_R) = 0$ or $\tau_{\mathcal{L}} > i \cdot T \cdot t(u)$. We want to show that u does not fire in $\tau_{\mathcal{L}}$. Towards contradiction, assume that it does. First note that if u receives no positive spikes in $\tau_{\mathcal{L}}$, then since $b(u) > 0$ u does not fire in $\tau_{\mathcal{L}}$. Otherwise, let $\langle v, u, \tau'_{\mathcal{L}}\rangle$ be a positive spike that arrives to u in round $\tau_{\mathcal{L}}$. Recall that since v is strong, there are three cases for defining the latency value of $\langle v, u, \tau'_{\mathcal{L}}\rangle$.

We will now show that $\langle v, u, \tau'_{\mathcal{L}}\rangle$ belongs to case (II). It does not belong to case (I), since it does not hold that $\sigma_i(u) = 1$ and $\tau_{\mathcal{L}} = i \cdot T$ If we are in case (II), then there exists an inhibitor f which is connected to u that fired in second τ'_R in the node-delay simulation that arrived in $\tau_{\mathcal{L}}$, i.e. such that $\tau_{\mathcal{L}} = \tau'_R \cdot T + (T^2 + 1)$. By the induction assumption for $\tau'_R \cdot T$, u fires in round $\tau'_R \cdot T$ in the edge-delay simulation, and since u is strong-incoming then by the definition of ℓ the spike $\langle f, u, \tau'_R \cdot T\rangle$ arrives to u in round $\tau'_R \cdot T + (T^2 + 1) = \tau_{\mathcal{L}}$. Since negative spikes are of weight $-\infty$, u does not fire in $\tau_{\mathcal{L}}$. Therefore, $\langle v, u, \tau'_{\mathcal{L}}\rangle$ belongs to case (III).

By the definition of case (III), $\langle v, u, \tau'_{\mathcal{L}}\rangle$ was generated in round $\tau'_{\mathcal{L}} = \tau_{\mathcal{L}} - 1$. Let j be such that $\tau_{\mathcal{L}} - 1 \in L_j(v)$, and let τ_R such that $\tau_{\mathcal{L}} - 1 \in B_{\tau_R}$. Furthermore, let v be an excitatory incoming neighbor that fires in $\tau_{\mathcal{L}} - 1$, let j be such that $\tau_{\mathcal{L}} - 1 \in L_j(v)$, and let τ_R such that $\tau_{\mathcal{L}} - 1 \in B_{\tau_R}$. Our goal is to show that v fires in $[\tau_R + 1, \tau_R + T]$ in the node-delay simulation, by showing that it receives enough positive spikes from its neighbors in this interval.

Let f be an inhibitor that has an edge to v. By the network properties f also has an edge to u, and since we are not in case (II) in the definition of ℓ, f does not fire in the interval $[\tau_R - T, \tau_{\mathcal{L}} + 2T]$ in the node-delay simulation. This implies that v does not receive a negative spike in $[\tau_R - T + 1, \tau_R + 2T + 1]$. Notice that $\tau_R \in [j \cdot t(v), (j+1) \cdot t(v) - 1]$, and since $t(v) \leq T$ we get

$$R_{j+1}(v) = [j \cdot t(v) + 1, (j+1) \cdot t(v)] \subseteq [\tau_R - T + 1, \tau_R + 2T + 1].$$

Therefore, v does not receive a negative spike in $R_{j+1}(v)$.
By the induction assumption for $\tau_{\mathcal{L}} - 1$ we have $\sigma_j(v, \mathcal{N}_R) = 1$. Together with the fact that v is strong and receives no negative spikes in $R_{j+1}(v)$, we get that $\sigma_{j+1}(v, \mathcal{N}_R) = 1$, i.e. v fires in the node-delay simulation in the second $(j+1) \cdot t(v)$. This implies that u receives a spike from v in $(j+1) \cdot t(v) + 1$, which is inside the interval $[\tau_R + 1, \tau_R + T]$. If so, let W the total weight of the incoming neighbors of u that fired in round $\tau_{\mathcal{L}} - 1$. Since u fires in round $\tau_{\mathcal{L}}$, it holds that $W \geq b(u)$. We will show this implies that u fires in some round in $[\tau_R + 1, \tau_R + 2T]$, which contradicts the fact that none of the arriving spikes belong to case I.

We showed that for every neuron v that fires in $\tau_{\mathcal{L}} - 1$ in the edge-latency simulation, u receive a spike from v in some round $\tau'_R \in [\tau_R + 1, \tau_R + T]$ in the node-delay simulation. By the definition of w_R it holds that $w_R(v,u) = T \cdot w_{\mathcal{L}}(v,u)$, and therefore we get

$$\sum_{\tau'_R = \tau_R + 1}^{\tau_R + T} W_{\tau'_R} \geq T \cdot W.$$

By an averaging argument there is a second $\tau'_R \in [\tau_R + 1, \tau_R + T]$ with $W_{\tau'_R} \geq (T \cdot W)/T = W$.

Let i' be an integer such that $\tau'_R \in R_{i'}(u)$. Therefore u receive in $R_{i'}(u)$ a total positive spike weight of at least $W \geq b(u)$. Furthermore, since no spike belongs to case C.2, u do not receive a negative spike in $R_{i'}(u) \subseteq [\tau_R + 1, \tau_R + 2T]$. Thus, u fires in the second $i' \cdot t(u) \in [\tau_R + 1, \tau_R + 2T]$, a contradiction.

C.3 The Complete Synchronization Scheme

We are now ready to complete the proof of Theorem 5. We consider a neural network $\mathcal{N}$ and an integer parameter T. Set $L = 5T^2$ and let $\mathcal{N}_{\mathcal{L}} = \mathsf{sync}_E(\mathcal{N}, L)$ be the synchronized network of $\mathcal{N}$ in the L-bounded node-delay model. We will now show that $\mathcal{N}_{\mathcal{L}}$ satisfies the properties in the conditions of Lemma 49.

Showing that $\mathcal{N}_{\mathcal{L}}$ satisfies the properties of Lemma 49.

Note that by the definition of the edge-delay synchronization scheme given in Section 4.3, every neuron $u \in \mathcal{N}_{\mathcal{L}}$ is contained in one of the following modules: (1) an $\mathsf{OR}_{\mathsf{sync}}$ or $\mathsf{NOT}_{\mathsf{sync}}$ subnetwork (Section 4.1), (2) a chain of a threshold gate which is a implemented as a boolean circuit subnetwork (Section B.3), or (3) the chain of the global pulse generator (Section 4.3). By the definitions of these modules, properties 1 and 2 hold. Moreover, together with the fact that edges between the modules connect only weak excitatory neurons, we also get property 3. Furthermore, note that the only inhibitors in the network are r and v_r neurons (which is later added in 4.3) in the $\mathsf{NOT}_{\mathsf{sync}}$ module, and all their edges have weight $-\infty$. Therefore, property 4 is satisfied.

The remaining properties 5 and 6 are relevant only for strong neurons. Therefore, consider the $\mathsf{NOT}_{\mathsf{sync}}$ module (Lemma 14), which is the only module that contains strong neurons. By the module definition, there are two possible types of strong neurons: (i) the memory neuron m, that is only connected to the reset neuron r; and (ii) intermediate neuron v_i, that is only connected to the output neuron z. In case (i), v has only one incoming inhibitor, which is the neuron v_r that resets the whole network after it finishes. Thus v_r also has an edge to r. In case (ii), v has two incoming inhibitors, v_r and r, which both have an edge to z. Therefore property 6 holds. Furthermore, both r and z have no edges from weak neurons. Hence, property 5 holds.

Indeed, $\mathcal{N}_{\mathcal{L}}$ satisfies the conditions of Lemma 49, and therefore there exists a network $\mathcal{N}_R$ in the T-bounded node delay model which is similar to $\mathcal{N}_{\mathcal{L}}$. We are left to show the transitivity of similarity, i.e. that if $\mathcal{N} \sim \mathcal{N}_{\mathcal{L}}$ and $\mathcal{N}_{\mathcal{L}} \sim \mathcal{N}_R$, also $\mathcal{N} \sim \mathcal{N}_R$.

Showing transitivity of similarity.

Let t be a node-delay function for $\mathcal{N}_R$. First, by the similarities of the networkss we get $V(\mathcal{N}) = V(\mathcal{N}_{\mathcal{L}}) = V(\mathcal{N}_R)$. Moreover, by the definition of $\mathcal{N}_{\mathcal{L}} \sim \mathcal{N}_R$ there exists a latency function ℓ for $\mathcal{N}_{\mathcal{L}}$ such that $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle \sim \langle \mathcal{N}_R, t \rangle$. Let Π be the execution of $\mathcal{N}$, $\Pi_{\mathcal{L}}$ be the execution of $\langle \mathcal{N}, \ell \rangle$, and Π_R be the execution of $\langle \mathcal{N}, t \rangle$. Let the interval $[r_{\mathcal{L}}(v,p), r_{\mathcal{L}}(v,p+1))$ be the p^{th} phase of $\Pi_{\mathcal{L}}$, and define $[r_R(v,p), r_R(v,p+1))$ as the p^{th} phase of Π_R, where the definition of $r_R(v,p)$ is as follows. Let $L^*_p(v)$ be the earliest block $L_i(v)$ whose first round τ^*_p is contained in phase p of $\Pi_{\mathcal{L}}$, then $r_R(v,p) = \tau^*_p / T$. We wish to prove the following claim.

▷ Claim 53. For every neuron v and $p \geq 0$, v fires in round p of Π iff v fires in phase p of Π_R.

First, note that by the construction of $\mathcal{N}_{\mathcal{L}} = \mathsf{sync}_(\mathcal{N}, \mathcal{L})$, every neuron $v \in V(\mathcal{N})$ can fire only after the chain neuron $c_{\alpha L^4 + L}$ fires. Since $\alpha L^4 > t(v) \cdot T$ this implies that v does not fire in the first $t(v) \cdot T$ rounds of each phase in $\Pi_{\mathcal{L}}$. We prove the two directions of the claim.

- Assume neuron v fires in round p in Π. Because $\mathcal{N} \sim \langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$ there is a round $\tau_{\mathcal{L}}$ in phase p of $\Pi_{\mathcal{L}}$ where v fires. Since v does not fire in the first $t(v) \cdot T$ rounds of each phase we have $\tau_{\mathcal{L}} \geq r_{\mathcal{L}}(v,p) + t(v) \cdot T$. Since $L_j(v)$ consists of $t(v) \cdot T$ rounds, the first round of $L_j(v)$ is in phase p. Therefore, $j \cdot t(v) \cdot T \geq \tau^*$, and therefore $j \cdot t(v) \geq r_R(v,p)$. Furthermore we have that $j \cdot t(v)$ is not in phase $p+1$. Hence also $j \cdot (v) < r_R(v, p+1)$, i.e. $j \cdot t(v)$ is in phase p of Π. Due to the similarity $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle \sim \langle \mathcal{N}_R, t \rangle$, since v fires in $L_j(v)$ it also fires in $j \cdot t(v)$. Hence v fires in phase p of Π_R.
- Assume that v fires in phase p in Π_R. Assume this happens in round τ_R, then $\tau_R \geq \tau_p^*/T$. Thus $j \cdot T \cdot t(v) \geq \tau_p^* \geq r_{\mathcal{L}}(v,i)$. Furthermore, $j \cdot t(v) < \tau_p^*/T$ implies that round $j \cdot T \cdot t(v)$ was before phase $p+1$ of $\Pi_{\mathcal{L}}$. Therefore $j \cdot T \cdot t(v)$ is in phase p of $\Pi_{\mathcal{L}}$. By the similaritiy $\langle \mathcal{N}_{\mathcal{L}}, \ell \rangle \sim \langle \mathcal{N}_R, t \rangle$ we have that v fires in round $j \cdot T \cdot t(v)$ in $\Pi_{\mathcal{L}}$. Hence v fires in phase p of $\Pi_{\mathcal{L}}$. By the similarity $\mathcal{N} \sim \langle \mathcal{N}_{\mathcal{L}}, \ell \rangle$ we get that v fires in round p of Π.