Counting to Ten with Two Fingers: Compressed Counting with Spiking Neurons

Yael Hitron Merav Parter*†

Abstract

We consider the task of measuring time with probabilistic threshold gates implemented by bio-inspired spiking neurons. In the model of *spiking neural networks*, network evolves in discrete rounds, where in each round, neurons fire in pulses in response to a sufficiently high membrane potential. This potential is induced by spikes from neighboring neurons that fired in the previous round, which can have either an excitatory or inhibitory effect.

Discovering the underlying mechanisms by which the brain perceives the duration of time is one of the largest open enigma in computational neuro-science. To gain a better algorithmic understanding onto these processes, we introduce the *neural timer* problem. In this problem, one is given a time parameter t, an input neuron x, and an output neuron y. It is then required to design a minimum sized neural network (measured by the number of auxiliary neurons) in which every spike from x in a given round i, makes the output y fire for the subsequent t consecutive rounds.

We first consider a deterministic implementation of a neural timer and show that $\Theta(\log t)$ (deterministic) threshold gates are both sufficient and necessary. This raised the question of whether randomness can be leveraged to reduce the number of neurons. We answer this question in the affirmative by considering neural timers with spiking neurons where the neuron y is required to fire for t consecutive rounds with probability at least $1-\delta$, and should stop firing after at most $2t$ rounds with probability $1-\delta$ for some input parameter $\delta \in (0,1)$. Our key result is a construction of a neural timer with $O(\log\log 1/\delta)$ spiking neurons. Interestingly, this construction uses only *one* spiking neuron, while the remaining neurons can be deterministic threshold gates. We complement this construction with a matching lower bound of $\Omega(\min\{\log\log 1/\delta, \log t\})$ neurons. This provides the first separation between deterministic and randomized constructions in the setting of spiking neural networks.

Finally, we demonstrate the usefulness of compressed counting networks for *synchronizing* neural networks. In the spirit of distributed synchronizers [Awerbuch-Peleg, FOCS'90], we provide a general transformation (or simulation) that can take any synchronized network solution and simulate it in an asynchronous setting (where edges have arbitrary response latencies) while incurring a small overhead w.r.t the number of neurons and computation time.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Emails: {yael.hitron,merav.parter}@weizmann.ac.il.

†Supported in part by the BSF-NSF grants.

1 Introduction

Understanding the mechanisms by which brain experiences time is one of the major research objectives in neuroscience [MHM13, ATGM14, FSJ+15]. Humans measure time using a global clock based on standardized units of minutes, days and years. In contrast, the brain perceives time using specialized neural clocks that define their own time units. Living organisms have various other implementations of biological clocks, a notable example is the circadian clock that gets synchronized with the rhythms of a day.

In this paper we consider the algorithmic aspects of measuring *time* in a simple yet biologically plausible model of *stochastic spiking neural networks* (SNN) [Maa96, Maa97], in which neurons fire in discrete pulses, in response to a sufficiently high membrane potential. This model is believed to capture the spiking behavior observed in real neural networks, and has recently received quite a lot of attention in the algorithmic community [LMP17a, LMP17b, LMP17c, LM18, LMPV18, PV19, CCL19]. In contrast to the common approach in computational neuroscience and machine learning, the focus here is not on general computation ability or broad learning tasks, but rather on specific algorithmic implementation and analysis.

The SNN network is represented by a directed weighted graph $G = (V, A, W)$, with a special set of neurons $X \subset V$ called *inputs* that have no incoming edges, and a subset of *output* neurons[1] $Y \subset V$. The neurons in the network can be either deterministic threshold gates or probabilistic threshold gates. As observed in biological networks, and departing from many artificial network models, neurons are either strictly inhibitory (all outgoing edge weights are negative) or excitatory (all outgoing edge weights are positive). The network evolves in discrete, synchronous *rounds* as a Markov chain, where the firing probability of every neuron in round τ depends on the firing status of its neighbors in the preceding round $\tau - 1$. For probabilistic threshold gates this firing is modeled using a standard sigmoid function. Observe that an SNN network is in fact, a *distributed network*, every neuron responds to the firing spikes of its *neighbors*, while having no global information on the entire network.

Remark. In the setting of SNN, unlike classical distributed algorithms (e.g., LOCAL or CONGEST), the algorithm is fully specified by the *structure* of the network. That is, for a given network, its dynamic is fully determined by the model. Hence, the key complexity measure here is the size of the network measured by the number of auxiliary neurons[2]. For certain problems, we also care for the tradeoff between the size and the computation time.

1.1 Measuring Time with Spiking Neural Networks

We consider the algorithmic challenges of measuring time using networks of threshold gates and probabilistic threshold gates. We introduce the *neural timer* problem defined as follows:

> Given an input neuron x, an output neuron y, and a time parameter t, it is required to design a small neural network such that any firing of x in a given round invokes the firing of y for exactly the next t rounds.

In other words, it is required to design a succinct timer, activated by the firing of its input neuron, that alerts when exactly t rounds have passed.

A trivial solution with t auxiliary neurons can be obtained by taking a directed chain of length t (Fig. 1): the head of the chain has an incoming edge from the input x, the output y has incoming edges from the input x, and all the other t neurons on the chain. All these neurons are simple OR-gates, they fire in round τ if at least one of their incoming neighbor fired in round $\tau - 1$. Starting with the firing of x in round 0, in each round i, exactly one neuron, namely the i^{th} neuron on the chain fires, which makes y keep on firing for exactly t rounds until the chain fades out. In this basic solution, the network spends one neuron that counts +1 and dies. It is

[1] In contrast to the definition of *circuits*, we do allow output neurons to have outgoing edges and self loops. The requirement will be that the value of the output neurons converges over time to the desired solution.

[2] I.e., neurons that are not the input or the output neurons.

noteworthy that the neurons in our model are very simple, they do not have any memory, and thus cannot keep track of the firing history. They can only base their firing decisions on the firing of their neighbors in the *previous* round.

With such a minimal model of computation, it is therefore intriguing to ask how to beat this linear dependency (of network size) in the time parameter t. Can we count to ten using only two (memory-less) neurons? We answer this question in the affirmative, and show that even with just simple deterministic threshold gates, we can measure time up to t rounds using only $O(\log t)$ neurons. It is easy to see that this bound is tight when using deterministic neurons (even when allowing some approximation). The reason is that $o(\log t)$ neurons encode strictly less than t distinct configurations, thus in a sequence of t rounds, there must be a configuration that re-occurs, hence locking the system into a state in which y fires forever.

Theorem 1 (Deterministic Timers)**.** *For every input time parameter $t \in \mathbb{N}_{>0}$, (1) there exists a deterministic neural timer network $\mathcal{N}$ with $O(\log t)$ deterministic threshold gates, (2) any deterministic neural timer requires $\Omega(\log t)$ neurons.*

This timer can be easily adapted to the related problem of *counting*, where the network should output the number of spikes (by the input x) within a time window of t rounds.

Does Randomness Help in Time Estimation? Neural computation in general, and neural spike responses in particular, are inherently stochastic [Lin09]. One of our broader scope agenda is to understand the power and limitations of randomness in neural networks. Does neural computation become *easier* or *harder* due to the stochastic behavior of the neurons?

We define a randomized version of the neural timer problem that allows some slackness both in the approximation of the time, as well as allowing a small error probability. For a given error probability $\delta \in (0, 1)$, the output y should fire for at least t rounds, and must stop firing after at most $2t$ rounds[3] with probability at least $1 - \delta$. It turns out that this randomized variant leads to a considerably improved solution for $\delta = 2^{-O(t)}$:

Theorem 2 (Upper Bound for Randomized Timers)**.** *For every time parameter $t \in \mathbb{N}_{>0}$, and error probability $\delta \in (0, 1)$, there exists a probabilistic neural timer network $\mathcal{N}$ with $O(\min\{\log\log 1/\delta, \log t\})$ deterministic threshold gates plus* additional *random spiking neuron.*

Our starting point is a simple network with $O(\log 1/\delta)$ neurons, each firing independently with probability $1 - 1/t$. The key observation for improving the size bound into $O(\log\log 1/\delta)$ is to use the *time axis*: we will use a *single* neuron to generate random samples over time, rather than having *many* random neurons generating these samples in a *single* round. The deterministic neural counter network with time parameter of $O(\log 1/\delta)$ is used as a building block in order to gather the firing statistics of a single spiking neuron. In light of the $\Omega(\log t)$ lower bound for deterministic networks, we get the first separation between deterministic and randomized solutions for error probability $\delta = \omega(1/2^t)$. This shows that randomness can help, but up to a limit: Once the allowed error probability is exponentially small in t, the deterministic solution is the best possible. Perhaps surprisingly, we show that this behavior is tight:

Theorem 3 (Lower Bound for Randomized Timers)**.** *Any SNN network for the neural timer problem with time parameter t, and error $\delta \in (0, 1)$ must use $\Omega(\min\{\log\log 1/\delta, \log t\})$ neurons.*

Neural Counters. Spiking neurons are believed to encode information via their firing rates. This underlies the *rate coding* scheme [Adr26, TM97, GKMH97] in which the spike-count of the neuron in a given span of time is interpreted as a *letter* in a larger alphabet. In a network of memory-less spiking neurons, it is not so clear how to implement this rate dependent behavior. How can a neuron convey a complicated message over time if its neighboring neurons remember only its recent spike? This challenge is formalized by the following neural counter problem: Given an input neuron x, a time parameter t, and $\Theta(\log t)$ output neurons represented by a vector $\bar{y}$, it is required to design a neural network such that the output vector $\bar{y}$ holds the binary

[3] Taking $2t$ is arbitrary here, and any other constant would work as well.

representation of the number of times that x fired in a sequence of t rounds. As we already mentioned this problem is very much related to the neural timer problem and can be solved using $O(\log t)$ neurons. Can we do better?

The problem of maintaining a *counter* using a small amount of space has received a lot of attention in the *dynamic streaming* community. The well-known Morris algorithm [Mor78, Fla85] maintains an approximate counter for t counts using only $\log\log t$ bits. The high-level idea of this algorithm is to increase the counter with probability of $1/2^{C'}$ where C' is the current read of the counter. The counter then holds the exponent of the number of counts. By following ideas of [Fla85], carefully adapted to the neural setting, we show:

Theorem 4 (Approximate Counting)**.** *For every time parameter t, and $\delta \in (0,1)$, there exists a randomized construction of approximate counting network using $O(\log\log t + \log(1/\delta))$ deterministic threshold gates plus an additional* single *random spiking neuron, that computes an $O(1)$ (multiplicative) approximation for the number of input spikes in t rounds with probability $1-\delta$.*

We note that unlike the deterministic construction of timers that could be easily adopted to the problem of neural counting, our optimized randomized timers with $O(\log\log 1/\delta)$ neurons cannot be adopted into an approximate counter network. We therefore solve the latter by adopting Morris algorithm to the neural setting.

Broader Scope: Lessons From Dynamic Streaming Algorithms. We believe that approximate counting problem provides just one indication for the potential relation between succinct neural networks and dynamic streaming algorithms. In both settings, the goal is to gather statistics (e.g., over time) using a small amount of space. In the setting of neural network there are additional difficulties that do not show up in the streaming setting. E.g., it is also required to obtain fast *update time*, as illustrated in our solution to the approximate counting problem.

1.2 Neural Synchronizers

The standard model of spiking neural networks assumes that all edges (synapses) in the network have a uniform response latency. That is, the electrical signal is passed from the presynaptic neuron to the postsynaptic neuron within a fixed time unit which we call a *round*. However, in real biological networks, the response latency of synapses can vary considerably depending on the biological properties of the synapse, as well as on the distance between the neighboring neurons. This results in an asynchronous setting in which different edges have distinct response time. We formalize a simple model of spiking neurons in the asynchronous setting, in which the given neural network also specifies a *response latency* function $\ell : A \to \mathbb{R}_{\geq 1}$ that determines the number of rounds it takes for the signal to propagate over the edge. Inspired by the synchronizers of Awerbuch and Peleg [AP90], and using the above mentioned compressed timer and counter modules, we present a general simulation methodology (a.k.a synchronizers) that takes a network $\mathcal{N}_{\mathsf{sync}}$ that solves the problem in the synchronized setting, and transform it into an "analogous" network $\mathcal{N}_{\mathsf{async}}$ that solves the same problem in the asynchronous setting.

The basic building blocks of this transformation is the neural time component adapted to the asynchronous setting. The cost of the transformation is measured by the overhead in the number of neurons and in the computation time. Using our neural timers leads to a small overhead in the number of neurons.

Theorem 5 (Synchronizer, Informal)**.** *There exists a synchronizer that given a network $\mathcal{N}_{\mathsf{sync}}$ with n neurons and maximum response latency[4] L, constructs a network $\mathcal{N}_{\mathsf{async}}$ that has an "analogous" execution in the asynchronous setting with a total number of $O(n+L\log L)$ neurons and a time overhead of $O(L^3)$.*

We note that although the construction is inspired by the work of Awerbuch and Peleg [AP90], due to the large differences between these models, the precise formulation and implementation of our synchronizers are quite different. The most notable difference between

[4] I.e., L correspond to the *length* of the longest round.

the distributed and neural setting is the issue of memory: in the distributed setting, nodes can aggregate the incoming messages and respond when all required messages have arrived. In strike contrast, our neurons can only respond (by either firing or not firing) to signals arrived in the *previous* round, and all signals from previous rounds cannot be locally stored. For this reason and unlike [AP90], we must assume a bound on the largest edge latency. In particular, we show that the size overhead of the transformed network $\mathcal{N}_{\mathsf{async}}$ must depend, at least logarithmically, on the value of the largest latency L.

Observation 1. *The size overhead of any synchronization scheme is* $\Omega(\log L)$.

Proof. We will show that implementing a simple NOT-gate in the asynchronous setting requires $\Omega(\log L)$ neurons. In the synchronous setting, one can easily implement a NOT-gate by connecting the input neuron to the output neuron with a negative weight and setting the bias of the output to 0.

Assume towards contradiction that there exists a deterministic network $\mathcal{N}$ with $N = o(\log L)$ neurons, an input neuron x, and an output neuron y that computes $y = NOT(x)$ within T rounds. Hence, if x fired in round 0, the output z should *not* fire in any of the rounds $[0, T]$, and if x does not fire, then there must a round $\tau \in [0, T]$ in which y fires. We set the latencies on the edges of $\mathcal{N}$ such that the outgoing edges from x have latency L, and all other edges have latency 1.

Consider an execution Π_{yes} where x fires in rounds $[0, T-L+1]$, and an execution Π_{no} where x do not fire at all. The initial states of all other neurons are set to 0 in both Π_{yes} and Π_{no}. By the correctness guarantee, in the execution Π_{yes}, the output neuron y do not fire in rounds $[1, L+1]$, and in the execution Π_{no} there exists a round $\tau \in [0, T]$ in which z fires. The *state* of the network in round t is described by an N-length vector indicating the firing neurons in that round. Note that because the network contain $N = o(\log L)$ neurons, the network have at most $L/2$ distinct states.

Since the latency on all outgoing edges from x is L, during rounds $[1, L]$ of execution Π_{yes}, the signal from x did not reach any other neuron. Hence, the states of all neurons but x during rounds $[1, L]$ of execution Π_{no} are identical to those of execution Π_{yes}. In other words, except for the state of x, the two executions are indistinguishable over the first L rounds. By the correctness of Π_{yes}, we have that y is idle during the first L round, and therefore it is also idle in Π_{no} during these rounds.

Since the network has at most $L/2$ distinct states, there must be a state s that occurs at least twice during rounds $[1, L]$ in both of the executions Π_{yes} and Π_{no}. In addition, in all the rounds between these two occurrences of s, the output y did not fire (as y was idle in the first L rounds). Due to the memory-less property of the neurons, we conclude that the execution Π_{no} is locked into a no-configuration in which y will never fire, contradicting the correctness of the network. □

This provably illustrates the difference in the overhead of synchronization between general distributed networks and neural networks. We leave the problem of tightening this lower bound (or upper bound) as an interesting open problem.

Additional Related Work on Async. Setting in Neural Networks and Logical Circuits. To the best of our knowledge, there are two main previous theoretical work on asynchronous neural networks. Maass [Maa94] considered a quite elaborated model for deterministic neural networks with *arbitrary* response *functions* for the edges, along with latencies that can be chosen by the network designer. Within this generalized framework, he presented a coarse description of a synchronization scheme that consists of various time modules (e.g., initiation and delay modules). Our work complements the scheme of [Maa94] in the simplified SNN model by providing a rigorous implementation and analysis for size and time overhead. Khun et al.

[KSPS10] analyzed the synchronous and asynchronous behavior under the stochastic neural network model of DeVille and Peskin [DP08]. Their model and framework is quite different from ours, and does not aim at building synchronizers.

Turning to the setting of logical circuits, there is a long line of work on the asynchronous setting under various model assumptions [AFM69, Hau95, Spa01, BM06, **?**] that do not quite fit the memory-less setting of spiking neurons.

Comparison with Concurrent Work [WL19]. Independently to our work, Wang and Lynch proposed a similar construction for the neural counter problem. Their work restricts attention to deterministic threshold gates and do not consider the neural timer problem and synchronizers which constitute the main contribution of our paper. We note that our approximate counter solution with $O(\log\log t + \log(1/\delta))$ neurons resolves the open problem stated in [WL19].

1.3 Preliminaries

We start by defining our model along with useful notation.

A Neuron. A *deterministic* neuron u is modeled by a *deterministic* threshold gate. Letting $b(u)$ to be the threshold value of u. Then it outputs 1 if the weighted sum of its incoming neighbors exceeds $b(u)$. A *spiking neuron* is modeled by a probabilistic threshold gate that fires with a sigmoidal probability $p(x) = \frac{1}{1+e^{-x}}$ where x is the difference between the weighted incoming sum of u and its threshold $b(u)$.

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 ℓ auxiliary neurons $Z = \{z_1, ..., z_\ell\}$. In a *deterministic* neural network (DNN) all neurons are deterministic threshold gates. 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, $\mathrm{b}(v) \in \mathbb{R}_{\geq 0}$ is the threshold value (activation bias). The weight function defining the synapses is restricted in two ways. 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 $w(v, u) \leq 0$ for every u, and if v is excitatory, then $w(v, u) \geq 0$ for every u.

Network Dynamics. 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 $u^\tau = 1$ if u fires (i.e., generates a spike) in round τ. Let u^0 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 ($\Pr[u^\tau = 1]$), calculated as:

$$\mathrm{pot}(u,\tau) = \sum_{v \in V} w_{v,u} \cdot v^{\tau-1} - b(u) \text{ and } p(u,\tau) = \frac{1}{1 + e^{-\frac{\mathrm{pot}(u,\tau)}{\lambda}}} \tag{1}$$

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 (due to scaling of edge weights and thresholds), thus we set $\lambda = 1$. In *deterministic* neural networks (DNN), each neuron u is a deterministic threshold gate that fires in round τ iff $\mathrm{pot}(u, \tau) \geq 0$.

Network States (Configurations). Given a network $\mathcal{N}$ (either a DNN or SNN) with N neurons, the configuration (or *state*) of the network in time τ denoted as s_τ can be described as an N-length binary vector indicating which neuron fired in round τ.

The Memoryless Property. The neural networks have a memoryless property, in the sense that each state depends only on the state of the previous round. In a DNN network, the state $s_{\tau-1}$ fully determines s_τ. In an SNN network, for every fixed state s^* it holds $\Pr[s_\tau =$

$s^* \mid s_1, ...s_{\tau-1}] = \Pr[s_\tau = s^* \mid s_{\tau-1}]$. Moreover for any $\tau, \tau', r > 0$, it holds that $\Pr[s_{\tau+r} = s^* \mid s_\tau] = \Pr[s_{\tau'+r} = s^* \mid s_{\tau'}]$.

Hard-Wired Inputs. We consider neural networks that *solve* a given parametrized problem (e.g., neural timer with time parameter t). The parameter to the problem can be either hard-wired in the network or alternatively be given as part of the input layer to the network. In most of our constructions, the time parameter is hard-wired. In some cases, we also show constructions with soft-wiring.

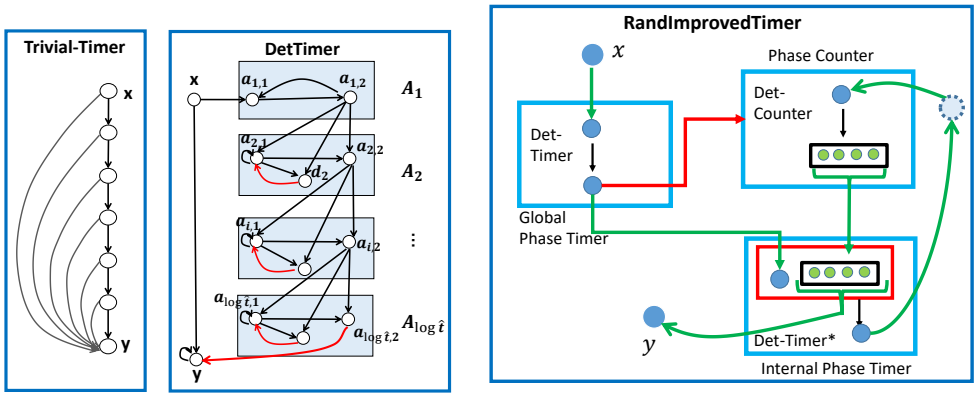

Figure 1: Illustration of timer networks with time parameter t. Left: The naïve timer with $\Theta(t)$ neurons. Mid: deterministic timer with $\Theta(\log t)$ neurons. Right: randomized timer with $O(\log \log 1/\delta))$ neurons, using the DetTimer modules with parameter $t' = \log 1/\delta$.

2 Deterministic Constructions of Neural Timer Networks

As a warm-up, we start by considering deterministic neural timers.

Definition 1 (Det. Neural Timer Network)**.** *Given time parameter t, a* deterministic *neural timer network $\mathcal{DT}$ is a network of threshold gates, with an input neuron x, an output neuron y, and additional auxiliary neurons. The network satisfies that in every round τ, $y^\tau = 1$ iff there exists a round $\tau > \tau' \geq \tau - t$ such that $x^{\tau'} = 1$.*

Lower Bound (Pf. of Thm. 1(2)). For a given neural timer network $\mathcal{N}$ with N auxiliary neurons, recall that the *state* of the network in round τ is described by an N-length vector indicating the firing neurons in that round. Assume towards contradiction that there exists a neural timer with $N \leq \log t - 1$ auxiliary neurons. Since there are at most 2^N different states, by the pigeonhole principle, there must be at least two rounds $\tau, \tau' \leq t - 1$ in which the state of the network is identical, i.e., where $s_\tau = s_{\tau'} = s^*$ for some $s^* \in \{0,1\}^N$. By the correctness of the network, the output neuron y fires in all rounds $\tau'' \in [\tau + 1, \tau' + 1]$. By the memoryless property, we get that $s_{\tau''} = s^*$ for $\tau'' = \tau + i \cdot (\tau' - \tau)$ for every $i \in \mathbb{N}_{\geq 0}$. Thus y continues firing forever, in contradiction that it stops firing after t rounds. Note that this lower bound holds even if y is allowed to stop firing in any finite time window.

A Matching Upper Bound (Pf. Thm. 1(1)). For ease of explanation, we will sketch here the description of the network assuming that it is applied only once (i.e., the input x fires once within a window of t rounds). Taking care of the general case requires slight adaptations[5], see Appendix A for the complete details.

At the high-level, the network consists of $k = \Theta(\log t)$ layers $A_1, \ldots, A_k$ each containing two excitatory neurons $a_{i,1}, a_{i,2}$ denoted as *counting* neurons, and one inhibitory neuron d_i. Each layer A_i gets its input from layer A_{i-1} for every $i \geq 2$, and A_1 gets its input from x. The role of each layer A_i is to count *two* firing events of the neuron $a_{i-1,2} \in A_{i-1}$. Thus the neuron $a_{\log t,2}$ counts $2^{\log t}$ rounds.

[5]I.e., whenever x fires again in a window of t rounds, one should reset the timer and starts counting t rounds from that point on.

Because our network has an update time of $\log t$ rounds (i.e., number of rounds to update the timer), for a given time parameter t, the construction is based on the parameter $\hat{t}$ where $\hat{t} + \log \hat{t} = t$.

- The first layer A_1 consists of two neurons $a_{1,1}, a_{1,2}$. The first neuron $a_{1,1}$ has positive incoming edges from x and $a_{1,2}$ with weights $w(x, a_{1,1}) = 3$, $w(a_{1,2}, a_{1,1}) = 1$, and threshold $b(a_{1,1}) = 1$. The second neuron $a_{1,2}$ has an incoming edge from $a_{1,1}$ with weight $w(a_{1,1}, a_{1,2}) = 1$ and threshold $b(a_{1,2}) = 1$. Because we have a loop going from $a_{1,1}$ to $a_{1,2}$ and back, once x fired $a_{1,2}$ will fire every two rounds.

- For every $i = 2 \dots \log \hat{t}$, the i^{th} layer A_i contains 3 neurons, two *counting* neurons $a_{i,1}$, $a_{i,2}$ and a reset neuron d_i. The first neuron $a_{i,1}$ has positive incoming edges from $a_{i-1,2}$, and a self loop with weight $w(a_{i-1,2}, a_{i,1}) = w(a_{i,1}, a_{i,1}) = 1$, a negative incoming edge from d_i with weight $w(d_i, a_{i,1}) = -1$, and threshold $b(a_{i,1}) = 1$. The second counting neuron $a_{i,2}$ has incoming edges from $a_{i-1,2}$ and $a_{i,1}$ with weight $w(a_{i-1,2}, a_{i,2}) = w(a_{i,1}, a_{i,2}) = 1$, and threshold $b(a_{i,2}) = 2$. The reset neuron d_i is an inhibitor copy of $a_{i-1,2}$ and therefore also has incoming edges from $a_{i-1,2}$ and $a_{i,1}$ with weight $w(a_{i-1,2}, d_i) = w(a_{i,1}, d_i) = 1$ and threshold $b(d_i) = 2$. As a result, $a_{i,1}$ starts firing after $a_{i-1,2}$ fires once, and $a_{i,2}$fires after $a_{i-1,2}$ fires twice. Then the neuron d_i inhibits $a_{i,1}$ and the layer is ready for a new count.

- The output neuron y has a positive incoming edge from x as well as a self-loop with weights $w(x, y) = 2$, $w(y, y) = 1$. In addition, it has a negative incoming edge from the last counting neuron $a_{\log \hat{t},2}$ with weight $w(a_{\log \hat{t},2}, y) = -1$ and threshold $b(y) = 1$. Hence, after x fires the output y continues to fire as long as $a_{\log \hat{t},2}$ did not fire.

- The last counting neuron $a_{\log \hat{t},2}$ also have negative outgoing edges to all counting neurons (neurons of the form $a_{i,j}$) with weight $w(a_{\log \hat{t},2}, a_{i,j}) = -2$. As a result, after the timer counts t rounds it is reset.

The key claim that underlines the correctness of Thm. 1(1) is as follows.

Claim 1. *If x fires in round t_0, for each layer i the neuron $a_{i,2}$ fires in rounds $t_0 + \ell \cdot 2^i + i - 1$ for every $\ell = 1 \dots \lfloor \hat{t}/2^i \rfloor$.*

Proof. The proof is by induction on i. For $i = 1$, once x fires in round t_0, neuron $a_{1,1}$ fires in round $t_0 + 1$ and $a_{1,2}$ fires in round $t_0 + 2$. Because there is a bidirectional edge between $a_{1,1}$ and $a_{1,2}$, the second counting neuron $a_{1,2}$ keeps firing every two rounds. Assume the claim holds for neuron $a_{i-1,2}$, and consider the i^{th} layer A_i. Recall that $a_{i,2}$ fires in round t' only if $a_{i,1}$ and $a_{i-1,2}$ fired in round $t' - 1$. The neuron $a_{i,1}$ fires one round after $a_{i-1,2}$ fires and keeps firing as long as d_i did not fire. By the induction assumption $a_{i-1,2}$ fired for the first time in round $2^{i-1} + i - 2$ and therefore $a_{i,1}$ starts firing in round $2^{i-1} + i - 1$. Note that in round $2^{i-1} + i - 1$ the neuron $a_{i-1,2}$ did not fire, and therefore the neurons $a_{i,2}$ and d_i can start firing only after $a_{i-1,2}$ fires again. Hence, only in round $2 \cdot 2^{i-1} + i - 2 + 1 = 2^i + i - 1$ the neurons $a_{i,2}$ and d_i fires for the first time. In the next round, because of the inhibition of d_i both counting neurons $a_{i,1}$ and $a_{i,2}$ do not fire and we can repeat the same arguments considering the next time the counting neurons $a_{i,1}$, $a_{i,2}$ fire.

We note that once the neuron $a_{\log \hat{t},2}$ fires for the first time in round $t_0 + 2^{\log \hat{t}} + \log \hat{t} - 1 = t_0 + \hat{t} + \log \hat{t} - 1$, it inhibits all the counting neurons. Hence, as long as x did not fire again, all counting neurons will be idle starting at round $t_0 + \hat{t} + \log \hat{t} = t_0 + t$. □

The complete proof is given in Appendix A.2.

Timer with Time Parameter. In Appendix A.3, we show a slight modified variant of neural timer denoted by $\mathsf{DetTimer}^*$ which also receives as input an additional set of $\log t$ neurons that encode the desired duration of the timer. This modified variant is used in our improved randomized constructions.

Neural Counters. In Appendix A.3 we also show a modification of the timer into a counter network DetCounter that instead of counting the number of rounds, counts the number of input spikes in a time interval of t rounds.

Lemma 1. *Given time parameter t, there exists a deterministic* neural counter *network which has an input neuron x, a collection of $\log t$ output neurons represented by a vector $\bar{y}$, and $O(\log t)$ additional auxiliary neurons. In a time window of t rounds, for every round τ, if x fired r_τ times in the last τ rounds, the output $\bar{y}$ encodes r_τ by round $\tau + \log r_\tau + 1$.*

This extra-additive factor of $\log r_\tau$ is due to the update time of the counter. In Appendix B, we revisit the neural counter problem and provide an *approximate* randomized solution with $O(\log\log t + \log(1/\delta))$ many neurons where δ is the error parameter. This construction is based on the well-known Morris algorithm (using the analysis of [Fla85]) for approximate counting in the streaming model.

3 Randomized Constructions of Neural Timer Networks

We now turn to consider randomized implementations. The input to the construction is a time parameter t and an error probability $\delta \in (0,1)$, that are hard-wired into the network.

Definition 2 (Rand. Neural Timer Network)**.** *A randomized neural timer $\mathcal{RT}$ for parameters $t \in \mathbb{N}_{>0}$ and $\delta \in (0,1)$, satisfies the following for a time window of* $\mathsf{poly}(t)$ *rounds.*

- *For every fixed firing event of x in round τ, with probability $1-\delta$, y fires in each of the following t rounds.*
- *$y^{\tau'} = 0$ for every round τ' such that $\tau' - \mathsf{Last}(\tau') \geq 2t$ with probability $1-\delta$, where $\mathsf{Last}(\tau') = \max\{i \leq \tau' \mid x^i = 1\}$ is the last round τ in which x fired up to round τ'.*

Note that in our definition, we have a success guarantee of $1-\delta$ for any fixed firing event of x, on the event that y fires for t many rounds after this firing. In contrast, with probability of $1-\delta$ over the entire span of $\mathsf{poly}(t)$ rounds, y *does not* fire in cases where the last firing of x was $2t$ rounds apart. We start by showing a simple construction with $O(\log 1/\delta)$ neurons Ideas along this line appear (somewhat implicitly) in the neural renaming network of [HLMP19].

3.1 Warm Up: Randomized Timer with $O(\log 1/\delta)$ Neurons

The network $\mathsf{BasicRandTimer}(t,\delta)$ contains a collection of $\ell = \Theta(\log 1/\delta)$ spiking neurons $A = \{a_1, \ldots, a_\ell\}$ that can be viewed as a *time-estimator population*. Each of these neurons have a positive self loop, a positive incoming edge from the input neuron x, and a positive outgoing edge to the output neuron y. See Figure 2 for an illustration. Whereas these a_i neurons are probabilistic spiking neurons[6], the output y is simply a threshold gate. We next explain the underlying intuition. Assume that the input x fired in round 0. It is then required for the output neuron y to fire for at least t rounds $1, \ldots, t$, and stop firing after at most $2t$ rounds with probability $1-\delta$. By having every neuron a_i firing (independently) w.p $(1-1/t)$ in each round given that it fired in the previous round[7], we get that a_i fires for t *consecutive* rounds with probability $(1-1/t)^t = 1/e$. On the other hand, it fires for $2t$ consecutive rounds with probability $(1-1/t)^{2t} = 1/e^2$. Since we have $\Theta(\log 1/\delta)$ many neurons, by a simple application of Chernoff bound, the output neuron y (which simply counts the number of firing neurons in A) can distinguish between round t and round $2t$ with probability $1-\delta$.

Detailed Construction. The network $\mathsf{BasicRandTimer}(t,\delta)$ has input neuron x, output neuron y, and $\ell = \Theta(\log 1/\delta)$ spiking neurons $A = \{a_1, \ldots, a_\ell\}$. We set the weights of the self loop of each a_i, and the weight of the incoming edge from x to be $w(x,a_i) = w(a_i,a_i) = \log(t-1) + b(a_i)$.

[6] A neuron that fires with a probability specified in Eq. (1)

[7] A neuron a_i that stops firing in a given round, drops out and would not fire again with good probability.

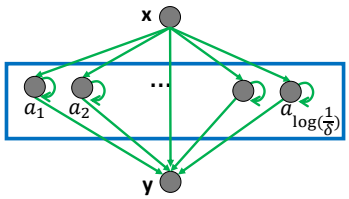

Figure 2: Illustration of the $\mathsf{BasicRandTimer}(t, \delta)$ network. Each neuron a_i fires with probability $1 - 1/t$ in round τ given that it fired in the previous round, and therefore fires for t consecutive rounds with constant probability. The output y fires if at least $1/(2e)$ fraction of the a_i neurons fired in the previous round.

The threshold value of a_i is set to $b(a_i) = \Theta(\log(t\ell/\delta))$. This makes sure that given a firing of either x or a_i in round τ, the probability that a_i fires in round $\tau + 1$ is $1 - 1/t$. In the complementary case (neither x nor a_i fired in round τ), a_i fires in round τ with probability at most $O(\delta/\mathsf{poly}(t\ell))$. For the output y, we let $w(a_i, y) = 1$ for each a_i, we set the weight from x to be $w(x, y) = \frac{\ell}{2e}$, and its threshold is $b(y) = \frac{\ell}{2e}$. This makes sure that y fires in round τ' if either x or at least $1/2e$ fraction of the a_i neurons fired in round $\tau' - 1$. We next analyze the construction.

Lemma 2 (Correctness)**.** *Within a time window of* $\mathsf{poly}(t)$ *rounds it holds that:*

- *For every fixed firing event of x in round τ, with probability $1 - \delta$, y fires in each of the following t rounds.*
- $y^{\tau'} = 0$ *for every round τ' such that $\tau' - \mathsf{Last}(\tau') \geq 2t$ with probability at least $1 - \delta$.*

Proof. When x fires in round τ_0, each neuron a_i fires for the following t consecutive rounds independently with probability $1/e$. Therefore, the expected number of neurons in A that fired for t consecutive rounds starting round $\tau_0 + 1$ is $\frac{\ell}{e}$. Using Chernoff bound upon picking a large enough constant c s.t $\ell = c \cdot \log(1/\delta)$, at least $\ell/2e$ auxiliary neurons fired for t consecutive rounds and therefore y fires in rounds $[\tau_0 + 2, \tau_0 + t]$ with probability $1 - \delta$. Since y has an incoming edge from x, it fires in round $\tau_0 + 1$ as well.

Next, recall that for each neuron $a_i \in A$, given that a_i or x did not fire in round τ, the probability that a_i fires in round $\tau + 1$ is at most $\delta/\mathsf{poly}(\ell t)$. Hence by union bound, in a window of $\mathsf{poly}(t)$ rounds, the probability there exists a neuron $a_i \in A$ that fired in round τ' but did not fire in round $\tau' - 1$ is at most $\delta/2$. Assuming no $a_i \in A$ fires unless it fired previously, each $a_i \in A$ fires for $2t$ consecutive rounds with probability $1/e^2$. Using Chernoff bound the probability at least $\frac{\ell}{2e}$ neurons from A fired for $2t$ consecutive rounds is at most $\delta/2$ (again we choose ℓ accordingly). Thus, we can conclude that the probability that there is a round τ' s.t $\tau' - \mathsf{Last}(\tau') \geq 2t$ in which $y^{\tau'} = 1$ is at most δ. □

3.2 Improved Construction with $O(\log \log 1/\delta)$ Neurons

We next describe an optimal randomized timer $\mathsf{ImprovedRandTimer}$ with an exponentially improved number of auxiliary neurons. This construction also enjoys the fact that it requires a *single* spiking neuron, while the remaining neurons can be deterministic threshold gates. Due to the tightness of Chernoff bound, one cannot really hope to estimate time with probability $1 - \delta$ using $o(\log(1/\delta))$ samples. Our key idea here is to generate the same number of samples by re-sampling one particular neuron over several rounds. Intuitively, we are going to show that for our purposes having $\ell = \log(1/\delta)$ neurons $a_1, \ldots, a_\ell$ firing with probability $1 - 1/t$ in a *given* round is *equivalent* to having a *single* neuron a^* firing with probability $1 - 1/t$ (independently) in a sequence of ℓ rounds.

Specifically, observe that the distinction between round t and $2t$ in the BasicRandTimer network is based only on the *number* of spiking neurons in a given round. In addition, the distribution on the number of times a^* fires in a span of ℓ rounds is equivalent to the distribution on the number of firing neurons $a_1, \ldots, a_\ell$ in a given round. For this reason, every phase of ImprovedRandTimer simulates a single round of BasicRandTimer. To count the number of firing events in ℓ rounds, we use the deterministic neural counter module with $\log \ell = O(\log \log 1/\delta)$ neurons.

We now further formalize this intuition. The network ImprovedRandTimer simulates each round of BasicRandTimer using a phase of $\ell' = \Theta(\log 1/\delta)$ rounds [8], but with only $O(\log \log 1/\delta)$ neurons. In the BasicRandTimer network each of the neurons a_i fires (independently) in each round w.p $1 - 1/t$. Once it stops firing in a given round, it basically drops out and would not fire again with good probability. Formally, consider an execution of the BasicRandTimer and let n_i be the number of neurons in A that fired in round i. In round $i+1$ of this execution, we have n_i many neurons each firing w.p $1 - 1/t$ (while the remaining neurons in A fire with a very small probability). In the corresponding $i+1$ phase of the network ImprovedRandTimer, the chief neuron a^* fires w.p $1 - 1/t'$ where $t' = \frac{t}{\ell'}$ for $n'_i \leq \ell$ consecutive rounds[9] where n'_i is the number of rounds in which a^* fired in phase i.

The dynamics of the network ImprovedRandTimer is based on discrete phases. Each phase has a fixed number of $\ell' = O(\ell)$ rounds, but has a possibly different number of *active rounds*, namely, rounds in which a^* attempts firing. Specifically, a phase i has an active part of n'_i rounds where n'_i is the number of rounds in which a^* fired in phase $i-1$. In the remaining $\ell' - n'_i$ rounds of that phase, a^* is idle. To implement this behavior, the network should keep track of the number of rounds in which a^* fires in each phase, and supply it as an input to the next phase (as it determines the length of the active part of that phase). For that purpose we will use the deterministic modules of neural timers and counters. The module DetCounter with time parameter $\Theta(\log 1/\delta)$ is responsible for counting the number of rounds that a^* fires in a given phase i. The output of this module at the end of the phase is the input to a DetTimer* module[10] in the beginning of phase $i+1$. In addition, we also need a *phase timer* module DetTimer with time parameter $\Theta(\log 1/\delta)$ that "announces" the end of a phase and the beginning of a new one. Similarly to the network BasicRandTimer, the output neuron y fires as long as a^* fires for at least $(1/2e)$ fraction of the rounds in each phase (in an analogous manner as in the BasicRandTimer construction). See Fig. 3 for an illustration of the network. Note that since we only use deterministic modules with time parameter $\Theta(\log 1/\delta)$, the total number of neurons (which are all threshold gates) will be bounded by $O(\log \log 1/\delta)$. We next give a detailed description of the network and prove Thm. 2.

Complete Proof of Thm. 2: We first describe the modules of the network ImprovedRandTimer that gets as input the time parameter t and error probability δ.

Network Modules:

- A *Global-Phase-Timer* module implemented by a (slightly modified) module of DetTimer(ℓ'). Due to the update time of DetCounter (lemma 1), we set the length of each phase to $\ell' = \ell + \log \ell$ where ℓ correspond to the number of spiking neurons in BasicRandTimer. Upon initializing this timer, the output neuron of this module fires *after* ℓ' rounds (instead of firing *for* every ℓ' rounds). This firing is the wake-up call for the network that a phase has terminated (ℓ' rounds have passed). This will activate some cleanup steps, and a subsequent "announcement" for the start of a new phase.

  To allow this module to inhibit as well as excite other neurons in the network, we will have

[8] Due to tactical reasons each phase consists of $\ell' = \ell + \log \ell$ rounds instead of ℓ.

[9] Note that because each phase takes $\ell' = \Theta(\log 1/\delta)$ rounds, we will need to count $t' = \frac{t}{\ell'}$ many phases. Thus a^* fires with probability $1 - 1/t'$ rather then w.p $1 - 1/t$.

[10] Here we use the variant of DetTimer in which the time is encoded in the input layer of the network.

two output (copy) neurons, one will be inhibitor and the other excitatory. The inhibitor activates a clean-up round (in order to clear the counting information from the previous phase). After one round, using a delay neuron the excitatory neuron safely announces the beginning of a new phase.

- A *Internal-Phase-Timer* module also implemented by a (yet a differently slightly modified) variant of DetTimer. The role of this module is to indicate to the random neuron a^* the number of rounds in which it should attempt firing in each phase. Recall that each phase i starts by an active part of length n_i in which a^* attempts firing w.p. $1 - 1/t'$ in each of these rounds. In the remaining $\ell' - n_i$ rounds till the end of the phase, a^* is idle. In each phase i, we then set the internal timer to n_i, this will activate a^* for n_i rounds. The time parameter n_i is given as input to this module. For that purpose, we use the DetTimer* variant in which the time parameter is given as an input. In our case, this input is supplied by the output layer of the counting module (describe next) at the end of phase $i-1$. In particular, at the end of the phase, the output of this counting module is fed into the input layer of the *Internal-Phase-Timer* module. Then, the information of this counting will be deleted from the counting module, ready to maintain the counting in the next phase.

  Since we would need to keep on providing the counting information throughout the entire phase, we augment the input layer of this module by self loops that keeps on presenting this information thought the phase.

- A *Phase-Counter* module implemented by the DetCounter network maintains the number of rounds in which a^* fires in the current phase. At the end of every i^{th} phase, the output layer of this module stores the number of rounds in which a^* fired in phase i. At the end of the phase, upon receiving a signal from the *Global-Phase-Timer*, the output layer copies its information to the input layer of the *Internal-Phase-Timer* module using an intermediate layer of neurons, and the information of the module is deleted (by inhibitory connections from the *Global-Phase-Timer* module).

Complete Description (Edge Weights, Bais Values, etc.)

- The neuron a^* has threshold $b(a^*) = \Theta(\log(\ell t/\delta))$, and a positive incoming edge from the output z_1 of the *Internal-Phase-Timer* module with weight $w(z_1, a^*) = \ln(t' - 1) + b(a^*)$. Therefore a^* fires with probability $1 - 1/t'$ if z_1 fired in the previous round, and w.h.p[11] will not fire otherwise.

- Each neuron in the output of the *Phase-Counter* has a positive outgoing edge to an intermediate *copy* neuron c_i with weight 1. In addition each c_i has a positive incoming edge from the *Global-Phase-Timer* excitatory output with weight 1, and threshold $b(c_i) = 2$. The copy neurons have outgoing edges to the input of the *Internal-Phase-Timer* and are used to copy the current count for the next phase.

- The inhibitor output of the *Global-Phase-Timer* has outgoing edges to all neurons in the *Internal-Phase-Timer* and *Phase-Counter* with weight -5. This is used to clean-up the out-dated counting information at the end of the phase.

- The excitatory output of the *Global-Phase-Timer* has an outgoing edge to a delay neuron d with weight 1 and threshold $b(d) = 1$. Hence, d fires one round after a phase ended, and alerts the beginning of the new phase. The neuron d has outgoing edges to the input of *Global-Phase-Timer* and *Internal-Phase-Timer* with large weight.

- The output neuron y has incoming edges from the time input neurons $q_1, \ldots, q_{\log \ell}$ of the *Internal-Phase-Timer* module each with weight $w(q_i, y) = 1$ and threshold $b(y) = \frac{\ell}{2e}$.

[11] Here high probability refers to probability of $1 - \delta/\mathsf{poly}(t)$.

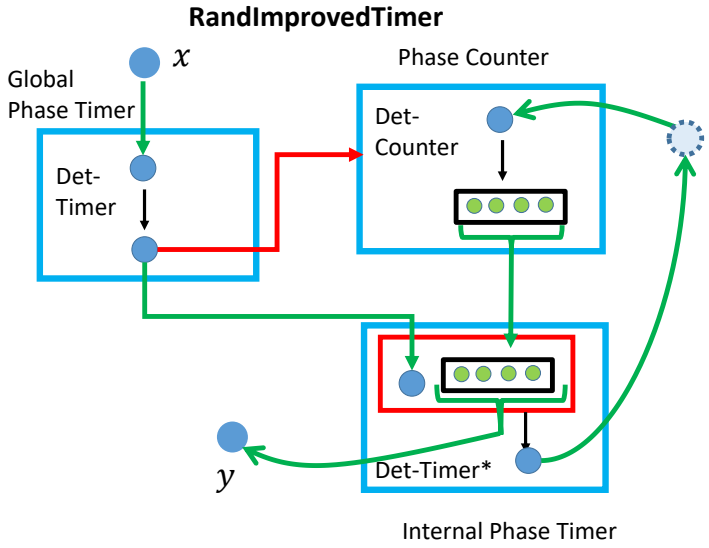

Figure 3: Schematic description of the randomized timer. In each module only the input layer and the output layer are shown. Excitatory (inhibitory) relations are shown in green (red) arrows. Each module is deterministic and has $\Theta(\log\log 1/\delta)$ threshold gates. The lower right module (*Internal-Phase-Timer*) uses the variant of the deterministic neural timer in which the time parameter is softly encoded in the input layer. This is crucial as the length of the $(i+1)^{th}$ active phase depends on the spike counts of a^* in phase i. This value is encoded by the output layer of the *Phase-Counter* module at the end of phase i. In contrast, the *Global-Phase-Timer* module uses the standard neural timer network (hard-wired), as the length of each phase is fixed.

Therefore y fires if a^* fired for at least $\frac{\ell}{2e}$ times in the previous phase. In addition, y has positive incoming edges from x and the delay neuron d of the *Global-Phase-Timer* module, each with weight $\frac{\ell}{2e}$. This insures that y also fires between phases.

- The *Global-Phase-Timer* input has an incoming edge from x with large weight, in order to initialize the timer when the input x fires. In addition, x has outgoing edges with large weight to the time input of the *Internal-Phase-Timer*, such that the decimal value of the input is set to ℓ.

All neurons except for a^* are threshold gates, see Figure 3 for a schematic description of the ImprovedRandTimer network.

Correctness. For simplicity we begin by showing the correctness of the construction assuming that there is a single firing of the input x during a period of $2t$ rounds. Taking care of the general case requires minor modifications that are described at the end of this section.

Our goal is to show that each *phase* of the ImprovedRandTimer network is equivalent to a *round* in the BasicRandTimer network. Toward that goal, we start by showing that the length of the active part in phase i has the same distribution as the number of neurons n_{i-1} that fire in round $i-1$ in BasicRandTimer(t',δ), where $t'=t/\ell'$. In the BasicRandTimer(t',δ) construction, let $\bar{B}_i$ be a random variable indicating the event that there exists a neuron $a\in A$ which fired in round $\tau\le i$ but a as well as x did not fire in round $\tau-1$. Similarly, for the ImprovedRandTimer(t,δ) construction, let $\bar{B}'_i$ be a random variable indicating the event that there exists a phase $\tau\le i$, where neuron a^* fired in an inactive round of phase τ. Note that in both constructions, the probability that a^* fired in an inactive round, and the probability that $a\in A$ fired given that it did not fire in the previous round is identical. Moreover, within a window of $\tau=\mathsf{poly}(t)$ rounds, by union bound both probabilities $\Pr[B_\tau]$ and $\Pr[B'_\tau]$ are at most $\delta/2$.

Let Y_i be a random variable for the number of neurons that fired in the i^{th} round in $\mathsf{BasicRandTimer}(t', \delta)$, and let X_i be the random variable for the number of rounds a^* fired during phase i in $\mathsf{ImprovedRandTimer}(t, \delta)$. In both constructions we assume that the input neuron x fired only in round 0.

Claim 2. *For any $k \geq 0$ and $i \geq 1$, $\Pr[X_i = k \mid \bar{B'}_i] = \Pr[Y_i = k \mid \bar{B}_i]$.*

Proof. By induction on i. For $i = 1$, given $\bar{B}_1$, $\bar{B'}_1$, the random variable X_1 as well as Y_1 are the sum of ℓ independent Bernoulli variables with probability $1 - 1/t'$ and therefore $X_1 = Y_1$. Assume $\Pr[X_i = k \mid \bar{B}_i] = \Pr[Y_i = k \mid \bar{B}_i]$ and we will show the equivalence for $i + 1$. First recall that in the $\mathsf{BasicRandTimer}(t', \delta)$ construction, each $a \in A$ fires with probability $1 - 1/t'$ given that it fired in the previous round. Moreover, conditioning on $\bar{B}_{i+1}$, we assumed that given that a did not fire in round i, it does not fire in round $i + 1$. Thus, for any k, j it holds that $\Pr[Y_{i+1} = k \mid Y_i = j, \bar{B}_{i+1}] = \binom{j}{k}(1 - 1/t')^k \cdot (1/t')^{j-k}$ (i.e., a binomial distribution). Similarly, in the $\mathsf{ImprovedRandTimer}(t, \delta)$ construction, since we assumed that a^* fires only in the active rounds of each phase, given that a^* fired j times in phase i, in phase $i + 1$ it holds that $\Pr[X_{i+1} = k \mid X_i = j, \bar{B'}_{i+1}] = \binom{j}{k}(1 - 1/t')^k \cdot (1/t')^{j-k}$. By the law of total probability we conclude that

$$\begin{aligned}
\Pr[X_{i+1} = k \mid \bar{B'}_{i+1}] &= \sum_{j=0}^{\ell} \Pr[X_{i+1} = k \mid X_i = j, \bar{B'}_{i+1}] \cdot \Pr[X_i = j \mid \bar{B'}_{i+1}] \\
&= \sum_{j=0}^{\ell} \Pr[X_{i+1} = k \mid X_i = j, \bar{B'}_{i+1}] \cdot \Pr[Y_i = j \mid \bar{B}_i] \\
&= \sum_{j=0}^{\ell} \binom{j}{k}(1 - 1/t')^k \cdot (1/t')^{j-k} \cdot \Pr[Y_i = j \mid \bar{B}_i] \\
&= \sum_{j=0}^{\ell} \Pr[Y_{i+1} = k \mid Y_i = j, \bar{B}_{i+1}] \cdot \Pr[Y_i = j \mid \bar{B}_i] = \Pr[Y_{i+1} = k \mid \bar{B}_{i+1}],
\end{aligned}$$

where the second equality is due to the induction assumption. □

Hence, by the correctness of the network $\mathsf{BasicRandTimer}(t', \delta)$, with probability at least $1-\delta$ the neuron a^* fires at least $\ell/2e$ times in each of the first t' phases. Since every phase consists of $\ell' = \ell + \log \ell$ rounds, y fires for at least $\ell' \cdot t' = t$ rounds w.h.p. On the other hand, with probability at most $\delta/2$ the neuron a^* fires in an inactive round during one of the first $2t'$ phases. Given that a^* fired only in active-rounds, we conclude that with probability at most $\delta/2$ the output y fires for at least $2t'$ phases. All together, with probability at least $1 - \delta$ the output y stops firing by round $2t' \cdot \ell' = 2t$.

Finally, we describe the small modifications needed to handle the case where x fires several times within a window of $2t$ rounds. Upon any firing of x, all modules get reset, and a new counting starts. To implement this reset, we connect the input neuron x to two additional neurons, an inhibitor neuron x_1, and an excitatory neuron x_2 where $w(x, x_1) = w(x, x_2) = 1$ and the thresholds are $b(x_1) = b(x_2) = 1$. The inhibitor x_1 has outgoing edges to all auxiliary neurons in the network with weight -4. The excitatory neuron x_2 has outgoing edges to the input of *Global-Phase-Timer* and the time input of the *Internal-Phase-Timer*, such that the decimal value is equal to ℓ with weights 6. Thus, after one round of cleaning-up, the network starts to account the last firing of x. The output y has incoming edges from x and x_2 each with weight $w(x, y) = w(x_1, y) = \frac{\ell}{2e}$, this makes y fire during the reset period.

3.3 A Matching Lower Bound

We now turn to show a matching lower bound with randomized spiking neurons. Assume towards contradiction there exists a randomized neural timer $\mathcal{N}$ for a given time parameter t

with $N = o(\log\log 1/\delta)$ neurons that succeeds with probability at least $1-\delta$. This implies that once x fired, y fires for t consecutive round with probability $1-\delta$. Moreover, there exists some constant $c \geq 2$ such that y stops firing after $(c-1)\cdot t$ rounds w.p $1-\delta$. Throwout the proof, we assume w.l.g that x fired in round 0. Recall that the state of the network in time τ denoted as s_τ can be described as an N-length binary vector. Since we have N many neurons, the number of distinct states (or configurations) is bounded by $S = 2^N = o(\log 1/\delta)$. We start by establishing useful auxiliary claims.

We first claim that because we have relatively small number of states, and the memoryless property discussed in section 1.3 in every window of t rounds there exists a state that occurs at least twice (and with sufficient distance). Let s^* be a state for which the probability that there exist rounds $t', t'' \leq t$ such that $\frac{t}{3\cdot S} \leq t' - t'' \leq t$ and $s_{t'} = s_{t''} = s^*$ is at least $1/S$.

Claim 3. *There exists such a state s^*.*

Proof. Note that because $N = o(\log\log 1/\delta)$ and $1/\delta \leq 2^{\mathsf{poly}(t)}$ it holds that $\frac{t}{3\cdot S} \geq 1$. We partition the interval $[0,t]$ into $2\cdot(S+1)$ balanced intervals, each of size $t/2(S+1)$. Because we have only S different states, in every execution of the network for t many rounds, there must be a state that occurs in at least two even intervals. Thus, there exists a state s^* for which the probability that s^* occurred in two even intervals is at least $1/S$. Because each interval is of size $t/2(S+1) \geq t/3S$ we conclude that the claim holds. □

Next we use the assumption that with probability at least $1-\delta$ the output y fires in rounds $[0,t]$ as well as the memoryless property to show that given that state s^* occurred in round t', with a sufficiently large probability, s^* occurs again with a long enough interval, and y fires in all rounds between the two occurrences of s^*. Let $p(t') = \Pr[\exists t'' \in [t'+t/(3S), t'+t], s_{t''} = s^* \text{ and } y^{t^*} = 1 \ \ \forall t^* \in [t', t''] \mid s_{t'} = s^*]$. By the memoryless property, we have:

Observation 2. *$p(t') = p(t'')$ for every t', t''.*

Define $p^* = p(1) = p(t')$ for any round t'. The next claim shows that p^* is sufficiently large.

Claim 4. *$p^* \geq 1/S - \delta$.*

Proof. Let A be an indicator random variable for the event that there exist $0 < t', t'' < t$ such that $t'' - t' \in [t/3S, t]$, $s_{t'} = s_{t''} = s^*$. Let B be the indicator random variable for the event that there exists $t^* \in [0,t]$ such that $y^{t^*} = 0$. By Claim 3, $\Pr[A] \geq 1/S$, and by the success guarantee of the network, $\Pr[B] \leq \delta$. Hence, by the union bound, we get that $\Pr[A \wedge \bar{B}] \geq 1/S - \delta$.

Let $A(t', t'')$ be the indicator random variable for the event that $s_{t'} = s_{t''} = s^*$ and $y^{t^*} = 1$ for every $t^* \in [t', t'']$. Let $F(t')$ be the indicator random variable for the event that s^* appears for the first time in round t'. Hence, we get

$$
\begin{aligned}
1/S - \delta \ &\leq \ \Pr[A \wedge \bar{B}] \leq \sum_{0<t'<t-t/(3S)} \Pr[F(t') \wedge \big(\exists t'' \in [t'+t/3S, t] \ \text{ s.t } \ s_{t''} = s^*\big) \wedge \bar{B}] \\
&\leq \sum_{0<t'<t-t/(3S)} \Pr[F(t') \wedge \big(\exists t'' \in [t'+t/3S, t] \ \text{ s.t } \ A(t', t'') = 1\big)] \\
&= \sum_{0<t'<t-t/(3S)} \Pr[F(t')] \cdot \Pr[\big(\exists t'' \in [t'+t/3S, t] \ \text{ s.t } \ A(t', t'') = 1\big) \mid F(t')] \\
&= \sum_{0<t'<t-t/(3S)} \Pr[F(t')] \cdot \Pr[\big(\exists t'' \in [t'+t/3S, t] \ \text{ s.t } \ A(t', t'') = 1\big) \mid s_{t'} = s^*] \\
&= \sum_{0<t'<t-t/(3S)} \Pr[F(t')] \cdot p^* \leq p^*,
\end{aligned}
$$

where the second inequality is by union bound over all possibilities for event A. The third equation is due to the memoryless property, the probability that the event occurred conditioning

on $F(t')$, is equivalent to conditioning on $s_{t'} = s^*$. The last equality follows by summing over a set of disjoint events $F(t')$. □

We are now ready to complete the proof of Theorem 3.

Proof of Thm. 3. We bound the probability that y fires in each of the first $c \cdot t$ rounds. Let C be the indicator random variable for the event that there exists a sequence of rounds $0 = \tau_0 < \tau_1 < \tau_2 < \cdots < \tau_{3c \cdot S}$ such that for every $i \geq 1$, it holds that:

- $y^{t^*} = 1$ for all $t^* \in [\tau_{i-1}, \tau_i]$.
- $s_{\tau_i} = s^*$.
- $\tau_i - \tau_{i+1} \in [t/3S, t]$.

Note that because $\tau_{i+1} - \tau_i \geq t/3S$, it holds that $\tau_{3c \cdot S} \geq c \cdot t$. Hence, the probability that y fires in each of the first $c \cdot t$ rounds is at least $\Pr[C]$. Next, we calculate the probability of the event C. Recall that given that $s_{\tau_i} = s^*$, the probability there exists a round $\tau_{i+1} \in [\tau_i + t/3S, \tau_i + t]$ for which $A(\tau_i, \tau_{i+1})$ is equal to $p(\tau_i) = p^*$. Moreover, by Claim 3 and the success guarantee, the probability there exists $0 < \tau_1 < t$ such that $A(0, \tau_1)$ is at least $1/S - \delta$. By Claim 4 and the memoryless property, we have:

$$\begin{aligned}\Pr[C] &= \Pr[\exists \tau_1 < t \ s.t \ A(0,\tau_1)] \cdot \prod_{i=1}^{3c \cdot S - 1} \Pr[\exists \tau_{i+1} \in [\tau_i + t/3S, \tau_i + t] \ s.t \ A(\tau_i, \tau_{i+1}) \ \mid \ s_{\tau_i = s^*}] \\ &\geq (1/S - \delta) \prod_{i=1}^{3c \cdot S - 1} p^* \geq (1/S - \delta)^{3cS} \ .\end{aligned}$$

Taking $N \leq (\log\log 1/\delta - \log\log\log 1/\delta) - \log 6c = \log(\frac{\log 1/\delta}{\log\log 1/\delta}) - \log 6c$, the number of different states is bounded by $S < \frac{\log 1/\delta}{6c \cdot \log\log 1/\delta}$. Thus the network fails with probability at least

$$(1/S - \delta)^{3c \cdot S} > \left(\frac{6 \cdot c \cdot \log\log 1/\delta}{\log 1/\delta} \cdot \frac{1}{2}\right)^{\frac{\log 1/\delta}{\log\log 1/\delta}} > \delta \ ,$$

in contradiction to the success guarantee of at least $1 - \delta$.

4 Applications to Synchronizers

The Asynchronous Setting. In this setting, the neural network $\mathcal{N} = \langle X, Z, Y, w, b \rangle$ also specifies a response latency function $\ell : A \to \mathbb{N}_{>0}$. For ease of notation, we normalize all latency values such that $\min_{e \in A} \ell(e) = 1$ and denote the maximum response latency by $L = \max_{e \in A} \ell(e)$. Supported by biological evidence [IB06], we assume that self-loop edges (a.k.a. autapses) have the minimal latency in the network, that $\ell((u,u)) = 1$ for self-edges (u,u). This assumption is crucial in our design[12]. Indeed the exceptional short latency of self-loop edges has been shown to play a critical role in biological network synchronization [MSJW15, FWW+18]. The dynamics proceeds in synchronous rounds and phases. The length of a round corresponds to the minimum edge latency, this is why we normalize the latency values so that $\min_{e \in A} \ell(e) = 1$. If neuron u fires in round τ, its endpoint v receives u's signal in round $\tau + \ell(e)$. Formally, a neuron u fires in round τ with probability $p(u, \tau)$:

$$\text{pot}(u,\tau) = \sum_{v \in X \cup Z \cup Y} w_{v,u} \cdot v^{\tau - \ell(u,v)} - b(u) \text{ and } p(u,\tau) = \frac{1}{1 + e^{-\frac{\text{pot}(u,\tau)}{\lambda}}} \tag{2}$$

[12] In a follow-up work, we actually show that this assumption is necessary for the existence of syncrnoizers even when $L = 2$.

Synchronizer. A synchronizer ν is an algorithm that gets as input a network $\mathcal{N}_{\mathsf{sync}}$ and outputs a network $\mathcal{N}_{\mathsf{async}} = \nu(\mathcal{N}_{\mathsf{sync}})$ such that $V(\mathcal{N}_{\mathsf{sync}}) \subseteq V(\mathcal{N}_{\mathsf{async}})$ where $V(\mathcal{N})$ denotes the neurons of a network $\mathcal{N}$. The network $\mathcal{N}_{\mathsf{async}}$ works in the asynchronous setting and should have *similar execution* to $\mathcal{N}_{\mathsf{sync}}$ in the sense that for every neuron $v \in V(\mathcal{N}_{\mathsf{sync}})$, the firing pattern of v in the asynchronous network should be similar to the one in the synchronous network. The output network $\mathcal{N}_{\mathsf{async}}$ simulates each round of the network $\mathcal{N}_{\mathsf{sync}}$ as a *phase*.

Definition 3 (Pulse Generator and Phases)**.** *A* pulse generator *is a module that fires to declare the end of each phase. Denote by $t(v,p)$ the (global) round in which neuron v receives the p^{th} spike from the pulse generator. We say that v is in phase p during all rounds $\tau \in [t(v,p-1), t(v,p)]$.*

Definition 4 (Similar Execution (Deterministic Networks))**.** *The* synchronous *execution Π_{sync} of a deterministic network $\mathcal{N}_{\mathsf{sync}}$ is specified by a list of states $\Pi_{\mathsf{sync}} = \{\sigma_1, \ldots, \}$ where each σ_i is a binary vector describing the firing status of the neurons in round i. The* asynchronous *execution of network $\mathcal{N}_{\mathsf{async}}$ denoted by Π_{async} is defined analogously only when applying the asynchronous dynamics (of Eq. (2)). The execution Π_{async} is divided into phases of fixed length. The networks $\mathcal{N}_{\mathsf{sync}}$ and $\mathcal{N}_{\mathsf{async}}$ have a* similar execution *if $V(\mathcal{N}_{\mathsf{sync}}) \subseteq V(\mathcal{N}_{\mathsf{async}})$, and in addition, a neuron $v \in V(\mathcal{N}_{\mathsf{sync}})$ fires in round p in the execution Π_{sync} iff v fires during phase p in Π_{async}.*

For simplicity of explanation, we assume that the network $\mathcal{N}_{\mathsf{sync}}$ is deterministic. However, our scheme can easily capture randomized networks as well (i.e., by fixing the random bits in the synchronized simulation and feeding it to the async. one).

4.1 Extension for Randomized Networks

For networks $\mathcal{N}_{\mathsf{sync}}$ that contain also probabilistic threshold gates, the notion of similar execution is defined as follows. Consider a fixed execution Π_{sync} of the network $\mathcal{N}_{\mathsf{sync}}$. In each round of simulating $\mathcal{N}_{\mathsf{sync}}$, the spiking neurons flip a coin with probability that depends on their potential. Once we are fixing those random coins used by the neurons in the execution Π_{sync}, the process becomes deterministic. Formally, for every round p and neuron v, let $R(v,p)$ be the set of random coins used by the neuron v in round p in the execution Π_{sync}. The firing decision of v in round p is fully determined given those bits. The asynchronous network $\mathcal{N}_{\mathsf{async}}$ contains a set of neurons V' that are analogous to the neurons in $\mathcal{N}_{\mathsf{sync}}$ and an additional set of deterministic threshold gates. When simulating this network, the neurons in V' will use the *same* random coins as those used by their corresponding neurons in Π_{sync}: in each phase p in the execution Π_{async}, the neuron v will be given the bits $R(v,p)$ and will base its firing decision using a deterministic function of its current potential, bias value and $R(v,p)$. This allows us to restrict attention to deterministic networks[13].

The Challenge. Consider a network of a threshold gate z with two incoming inputs: an excitatory neuron x, and an inhibitory neuron y. The weights are set such that z computes $X \wedge \bar{Y}$ thus it fires in round τ if x fired in round $\tau - 1$ and y did not fire. Implementing an $X \wedge \bar{Y}$ gate in the asynchronous setting is quite tricky. In the case where both x and y fire in round τ, in the synchronous network, z should not fire in round $\tau + 1$. However, in the asynchronous setting, if $\ell(x,z) < \ell(y,z)$, then z will mistakenly fire in round $\tau + \ell(x,z)$. This illustrates the need of enforcing a *delay* in the asynchronous simulation: the neurons should attempt firing only after receiving *all* their inputs from the previous phase. We handle this by introducing a pulse-generator module, that announces when it is safe to attempt firing.

To illustrate another source of challenge, consider the asynchronous implementation of an AND-gate $X \wedge Y$. If both x and y fire in round τ, then z fires in round $\tau + 1$ in the synchronous setting. However, if the latencies of the edges $\ell(x,z)$ and $\ell(y,z)$ are distinct, z receives the spike from x and y in *different* rounds, thus preventing the firing of z. Recall, that z has no

[13] Where the neurons in those networks are not necessarily threshold gates, but rather base their firing decision using *some* deterministic function

memory, and thus its firing decision is based only on the potential level in the previous round. To overcome this hurdle, in the transformed network, each neuron in the original synchronous network is augmented with 3 copy-neurons, some of which have self-loops. Since self-loops have latency 1, once a neuron with a self-loop fires, it fires in the next round as well. This will make sure that the firing states of x and y are kept on being presented to z for sufficiently many rounds, which guarantees the existence of a round where both spikes arrive.

While solving one problem, introducing self-loops into the system brings along other troubles. Clearly, we would not want the neurons to fire forever, and at some point, those neurons should get inhibition to allow the beginning of a new phase. This calls for a delicate *reset* mechanism that cleans up the old firing states at the end of each phase, only after their values have already being used. Our final solution consists of global synchronization modules (e.g., pulse-generator, reset modules) that are inter-connected to a modified version of the synchronous network. Before explaining those constructions, we start by providing a modified neural timer $\mathsf{DetTimer}_{\mathsf{async}}$ adapted to asynchronous setting. This timer will be the basic building block in our global synchronization infrastructures.

Asynchronous Analog of DetTimer. A basic building block in our construction is a variant of DetTimer to the asynchronous setting. Observe that the DetTimer implementation of Sec. 2 might fail miserably in the asynchronous setting, e.g., when the edges $(a_{i-1,2}, a_{i,2})$ have latency 2 for every $i \geq 2$, and the remaining edges have latency 1, the timer will stop counting after $\Theta(\log t)$ rounds, rather than after t rounds. In Appendix C.1, we show:

Lemma 3. *[Neural Timer in the Asynchronous Setting] For a given time parameter t, there exists a deterministic network $\mathsf{DetTimer}_{\mathsf{async}}$ with $O(L \cdot \log t)$ neurons, satisfying that in the asynchronous setting with maximum latency L, the output neuron fires at least $\Theta(t)$ rounds, and at most $\Theta(L \cdot t)$ rounds after each firing of the input neuron.*

Description of the Syncronizer. The construction has two parts: a global infrastructure, that can be used to synchronize many networks[14], and an adaptation of the given network $\mathcal{N}_{\mathsf{sync}}$ into a network $\mathcal{N}_{\mathsf{async}}$. The global infrastructures consists of the following modules:

- A *pulse generator* PG implemented by $\mathsf{DetTimer}_{\mathsf{async}}$ with time parameter $\Theta(L^3)$.
- A *reset* module R_1 implemented by a directed chain of $\Theta(L)$ neurons [15] with input from the output neuron of the PG module.
- A *delay* module D implemented by $\mathsf{DetTimer}_{\mathsf{async}}$ with time parameter $\Theta(L^2)$ and input from the output of of the PG module.
- Another *reset* module R_2 implemented by a chain of $\Theta(L)$ neurons with input from D.

The heart of the construction is the pulse-generator that fires once within a fixed number of $\ell \in [\Theta(L^3), \Theta(L^4)]$ rounds, and invokes a cascade of activities at the end of each phase. When its output neuron g fires, it activates the reset and the delay modules, R_1 and D. The second reset module R_2 will be activated by the delay module D. Both reset modules R_1 and R_2 are implemented by chains of length L, with the last neuron on these chains being an *inhibitor* neuron. The role of the reset modules is to *erase* the firing states of some neurons (in $\mathcal{N}_{\mathsf{async}}$) from the previous phase, hence their output neuron is an inhibitor. The timing of this clean-up is very delicate, and therefore the reset modules are separated by a delay module that prevents a premature operation. The total number of neurons in these global modules is $O(L \cdot \log L)$. We next consider the specific modifications to the synchronous network $\mathcal{N}_{\mathsf{sync}}$ (see Fig. 4).

Modifications to the Network $\mathcal{N}_{\mathsf{sync}}$. The input layer and output layer in N_{async} are *exactly* as in $\mathcal{N}_{\mathsf{sync}}$. We will now focus on the set of *auxiliary* neurons V in $\mathcal{N}_{\mathsf{sync}}$. In the network

[14] It is indeed believed that the neural brain has centers of synchronization.

[15] Each neuron in the chain has an incoming edge from its preceding neuron with weight 1 and threshold 1.

$\mathcal{N}_{\mathsf{async}}$, each $v \in V$ is augmented by three additional neurons $v_{\mathsf{in}}, v_{\mathsf{delay}}$ and v_{out}. The incoming (resp., outgoing) neighbors to v_{in} (resp., v_{out}) are the out-copies (resp., in-copies) of all incoming (resp., outgoing) neighboring neurons of v. The neurons $v_{\mathsf{in}}, v, v_{\mathsf{delay}}$ and v_{out} are connected by a directed chain (in this order). Both v_{delay} and v_{out} have self-loops.

In case where the original network $\mathcal{N}_{\mathsf{sync}}$ contains spiking neurons, the neuron v_{in} will be given the exact same firing function as v in Π_{sync}. That is, in phase p, v_{in} will be given the random coins[16] used by v in round p in Π_{sync}. The other neurons v, v_{delay} and v_{out} are deterministic threshold gates. The role of the *out-copy* v_{out} is to *keep on presenting* the firing status of v from the previous phase $p-1$ throughout the rounds of phase p. This is achieved through their self-loops. The role of the *in-copy* v_{in} is to simulate the firing behavior of v in phase p. We will make sure that v_{in} fires in phase p only if v fires in round p in Π_{sync}. For this reason, we set the incoming edge weights of v_{in} as well as its bias to be exactly the same as that of v in $\mathcal{N}_{\mathsf{sync}}$. The neuron v is an AND gate of its in-copy v_{in} and the PG output g. Thus, we will make sure that v fires at the *end* of phase p only if v_{in} fires in this phase as well. The role of the delay copy v_{delay} is to delay the update of v_{out} to the up-to-date firing state of v (in phase p). Since both neurons v_{delay} and v_{out} have self-loops, at the end of each phase, we need to carefully reset their values (through inhibition). This is the role of the reset modules R_1 and R_2. Specifically, the reset module R_1 operated by the pulse-generator inhibits v_{out}. The second reset module R_2 inhibits the delay neuron v_{delay} only after we can be certain that its value has already being "copied" to v_{out}. Finally, we describe the connections of the neuron v_{out}. The neuron v_{out} has an incoming edge from the reset module R_1 with a super-large weight. This makes sure that when the reset module is activated, v_{out} will be inhibited shortly after. In addition, it has a self-loop also of large weight (yet smaller than the inhibition edge) that makes sure that if v_{out} fires in a given round, and the reset module R_1 is not active, v_{out} also fires in the next round. Lastly, if v_{out} did not fire in the previous round, then it fires when receiving the spikes from *both* the delay module and from the delay copy v_{delay}. This will make sure that the firing state of v_{delay} will be copied to v_{out} only after the output of the delay module D fires.

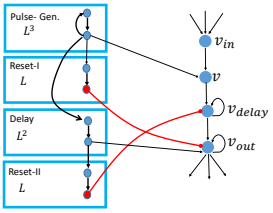

Figure 4: Illustration of the syncronizer modules. Left: global modules implemented by neural timers. Right: a neuron $v \in N_{\mathsf{sync}}$ augmented by three additional neurons that interact with the global modules.

4.2 Analysis of the Syncronizers.

Throughout, we fix a synchronous execution Π_{sync} and an asynchronous execution Π_{async}. For every round p, recall that $V^{+}_{\mathsf{sync}}(p)$ is the set of neurons that fire in round p in Π_{sync} (i.e., the neurons with positive entries in σ_p). In our simulation, we will make sure that each v in $\mathcal{N}_{\mathsf{async}}$ has the same firing pattern as its copy in $\mathcal{N}_{\mathsf{sync}}$.

Observation 3. *Consider a neuron v with incoming neighbors $u_1, \ldots, u_k$. If there is a round τ such that $u_1, \ldots, u_k$ fire in each round $\tau' \geq \tau$, v fires in every round $\tau'' \geq \tau + \max_{u_i} \ell(v, u_i)$.*

Lemma 4. *The networks $\mathcal{N}_{\mathsf{sync}}$ and $\nu(\mathcal{N}_{\mathsf{sync}}) = \mathcal{N}_{\mathsf{async}}$ have similar executions.*

Proof. We will show by induction on 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

[16] I.e., the random coins that are used to simulate the firing decision of v.

$v \in V$ fires at the end of phase 1 iff $v \in V^+_{\mathsf{sync}}(1)$. Without loss of generality, assume that g fired at the end phase 0 and begins the simulation in round 0. We begin with the following claim.

Claim 5. *For every $u \in V$, its in-copy u_{in} has a round $\tau_u \leq c_2 L^3 + L$ in which all its incoming neighbors in $V^+_{\mathsf{sync}}(0)$ fire (and the remaining neighbors do not fire), for a constant c_2.*

Proof. We first show that for $v \in V^+_{\mathsf{sync}}(0)$, the out-copy v_{out} fires when it receives a signal from the delay module D. Because each edge has latency at most L, by round L, neuron v has fired. Since the delay neuron v_{delay} has a self loop (with latency one), it starts firing in every round starting round $\tau_d \in [2, 2L]$ (until it is inhibited by the reset module R_2). Recall that the out-copy v_{out} is connected to the delay module D, and will fire only when receiving a spike from both the output neuron of D *and* the delay-neuron v_{delay}. We claim that v_{out} receives a signal from D and starts firing *after* it gets a reset from R_1. The reset module R_1 receives the signal from g by round L and starts counting L rounds. Thus, the output neuron of R_1 fires in some round $\tau'_{r_1} \in [L+1, L^2+L]$. This insures that by round L^2+2L the neuron v_{out} is inhibited by the output of R_1. The delay module D is implemented by $\mathsf{DetTimer}_{\mathsf{async}}$ with time parameter $2L^2$. Therefore, the output neuron of D fires in round $\tau_D \in [2L^2, 10L^3]$, ensuring that it fires only *after* v_{out} has been reset by the module R_1. Moreover, the reset module R_2 counts L rounds after receiving a signal from D. This ensures that the inhibitory output of R_2 starts inhibiting v_{delay} only *after* v_{out} has received the signal from D in round τ_{out}. Overall, we conclude that v_{out} fires in round $\tau_{\mathsf{out}} \in [c_1 \cdot L^2, c_2 \cdot L^3]$, for some constants c_1, c_2. Due to the self loop, v_{out} also fires in each round $\tau'' \geq \tau_{\mathsf{out}}$ in that phase. As a result for every $u \in V$, its in-copy u_{in} has a round $\tau_u \leq c_2 L^3 + L$ in which all its incoming neighbors in $V^+_{\mathsf{sync}}(0)$ fire. Note that for every neuron $v \notin V^+_{\mathsf{sync}}(0)$, non of its copy neurons $v_{\mathsf{out}}, v_{\mathsf{delay}}$ fire during the phase. □

Hence, u_{in} start firing in round τ_u only if u fires in round 1 in Π_{sync}, i.e., if $u \in V^+_{\mathsf{sync}}(1)$. We set the pulse-generator with time parameter $c_3 \cdot L^3$ for a large enough c_3 such that $c_3 \cdot L^3 > c_2 L^3 + 2L$. Since the out-copies keep on presenting the firing states of phase 0, u_{in} continues to fire in the last L rounds of the phase. Thus, when the pulse-generator spikes again, the neurons $v \in V^+_{\mathsf{sync}}(1)$ indeed fire as both g and v_{in} fired in the previous rounds.

Next, we assume that $V^+_{\mathsf{sync}}(p) = V^+_{\mathsf{async}}(p)$ and consider phase $p+1$. Let τ^* be the round that the PG fired at the end of phase p. We first show the following.

Claim 6. *For every $v \in V$, the neuron v_{delay} starts firing by round $\tau^* + 2L$, iff $v \in V^+_{\mathsf{sync}}(p)$.*

Proof. Recall that all delay copies are inhibited by the reset module R_2 at most $L^2 + 2L$ rounds after the delay module D has fired. We choose the time parameter of the PG to be large enough such that this occurs before the next pulse of PG in round τ^*. Hence, when phase p ended in round τ^*, all delay copies v_{delay} are idle. Because each edge has latency of at most L, by round $\tau^* + L$, all the neurons in $V^+_{\mathsf{sync}}(p)$ have fired (and by the assumption other neurons did not fire during phase p). As a result, the neuron v_{delay} starts firing by round $\tau^* + 2L$, iff $v \in V^+_{\mathsf{sync}}(p)$. □

We next show there exists a round in which the in-copies of $V^+_{\mathsf{sync}}(p+1)$ begin to fire.

Claim 7. *For every $u \in V$ its in-copy u_{in} has a round $\tau_u \in [\tau^* + c_1 \cdot L^2, \tau^* + c_2 \cdot L^3 + L]$ in which all its incoming neighbors in $V^+_{\mathsf{sync}}(p)$ fire, and the remaining neighbors do not fire.*

Proof. The output neuron of R_1 fires in some round $\tau' \in [\tau^* + L + 1, \tau^* + L^2 + L]$, and therefore all neurons v_{out} are inhibited by round $\tau^* + L^2 + 2L$. Recall that the delay module D is implemented by $\mathsf{DetTimer}_{\mathsf{async}}$ with time parameter $2L^2$. Therefore the output neuron of D fires in round $\tau_D \in [\tau^* + 2L^2 + 1, \tau^* + 10L^2]$, ensuring D fires after v_{out} was inhibited by R_1. Recall that the reset module R_2 counts L rounds after receiving a signal from D. This ensures that the inhibitory output of R_2 starts inhibiting v_{delay} after v_{out} received the signal from D. By Claim 15 we conclude that when neuron v_{out} receives the signal from the delay module D in some round $\tau_{\mathsf{out}} \in [\tau^* + c_1 \cdot L^2, \tau^* + c_2 L^3]$, it fires iff $v \in V^+_{\mathsf{sync}}(p)$. As a result, due to the self loops of the

out-copies, u_{in} has a round $\tau_u \in [\tau^* + c_1 \cdot L^2 + 1, \tau^* + c_2 \cdot L^3 + L]$ in which all its incoming neighbors in $V^+_{\mathsf{sync}}(p)$ fire. □

Therefore u_{in} starts firing in round τ_u only if $u \in V^+_{\mathsf{sync}}(p+1)$ and it continues firing from round τ_u ahead in that phase due to the self loops of the out-copies of its neighbors. Since the pulse generator fires to signal the end of phase $p+1$ in round $\tau^* + c_3 L^3 > \tau^* + c_2 \cdot L^3 + 2L$, every neuron $v \in V^+_{\mathsf{sync}}(p+1)$ fires in round $t(v, p+1)$ since both g and v_{in} fired previously (and other neurons will idle). □

Acknowledgment: We are grateful to Cameron Musco, Renan Gross and Eylon Yogev for various useful discussions.

References

[Adr26] Edgar D Adrian. The impulses produced by sensory nerve endings. *The Journal of physiology*, 61(1):49–72, 1926.

[AFM69] 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.

[AP90] 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.

[ATGM14] Melissa J Allman, Sundeep Teki, Timothy D Griffiths, and Warren H Meck. Properties of the internal clock: first-and second-order principles of subjective time. *Annual review of psychology*, 65:743–771, 2014.

[BM06] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1):1, 2006.

[CCL19] 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.

[DP08] 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.

[Fla85] Philippe Flajolet. Approximate counting: A detailed analysis. *BIT*, 25(1):113–134, 1985.

[FSJ+15] Gerald T Finnerty, Michael N Shadlen, Mehrdad Jazayeri, Anna C Nobre, and Dean V Buonomano. Time in cortical circuits. *Journal of Neuroscience*, 35(41):13912–13916, 2015.

[FWW+18] Huawei Fan, Yafeng Wang, Hengtong Wang, Ying-Cheng Lai, and Xingang Wang. Autapses promote synchronization in neuronal networks. *Scientific reports*, 8(1):580, 2018.

[GKMH97] Wulfram Gerstner, Andreas K Kreiter, Henry Markram, and Andreas VM Herz. Neural codes: firing rates and beyond. *Proceedings of the National Academy of Sciences*, 94(24):12740–12741, 1997.

[Hau95] Scott Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, 1995.

[HLMP19] Yael Hitron, Nancy Lynch, Cameron Musco, and Merav Parter. The neural renaming problem. In *Submitted*, 2019.

[IB06] Kaori Ikeda and John M Bekkers. Autapses. *Current Biology*, 16(9):R308, 2006.

[KSPS10] 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.

[Lin09] Benjamin Lindner. Some unsolved problems relating to noise in biological systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(01):P01008, 2009.

[LM18] Nancy Lynch and Cameron Musco. A basic compositional model for spiking neural networks. *arXiv preprint arXiv:1808.03884*, 2018.

[LMP17a] 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.

[LMP17b] Nancy Lynch, Cameron Musco, and Merav Parter. Spiking neural networks: An algorithmic perspective. In *5th Workshop on Biological Distributed Algorithms (BDA 2017)*, July 2017.

[LMP17c] 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.

[LMPV18] 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.

[Maa94] Wolfgang Maass. Lower bounds for the computational power of networks of spiking neurons. *Electronic Colloquium on Computational Complexity (ECCC)*, 1(19), 1994.

[Maa96] Wolfgang Maass. On the computational power of noisy spiking neurons. In *Advances in Neural Information Processing Systems 8 (NIPS)*, 1996.

[Maa97] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[MHM13] Hugo Merchant, Deborah L Harrington, and Warren H Meck. Neural basis of the perception and estimation of time. *Annual review of neuroscience*, 36:313–336, 2013.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[MSJW15] Jun Ma, Xinlin Song, Wuyin Jin, and Chuni Wang. Autapse-induced synchronization in a coupled neuronal network. *Chaos, Solitons & Fractals*, 80:31–38, 2015.

[PV19] 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.

[Spa01] Jens Sparsø. Asynchronous circuit design-a tutorial. In *Chapters 1-8 in" Principles of asynchronous circuit design-A systems Perspective"*. Kluwer Academic Publishers, 2001.

[TM97] Misha V Tsodyks and Henry Markram. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the national academy of sciences*, 94(2):719–723, 1997.

[WL19] Barbeeba Wang and Nancy Lynch. Integrating temporal information to spatial information in a neural circuit. *arXiv preprint arXiv:1903.01217*, 2019.

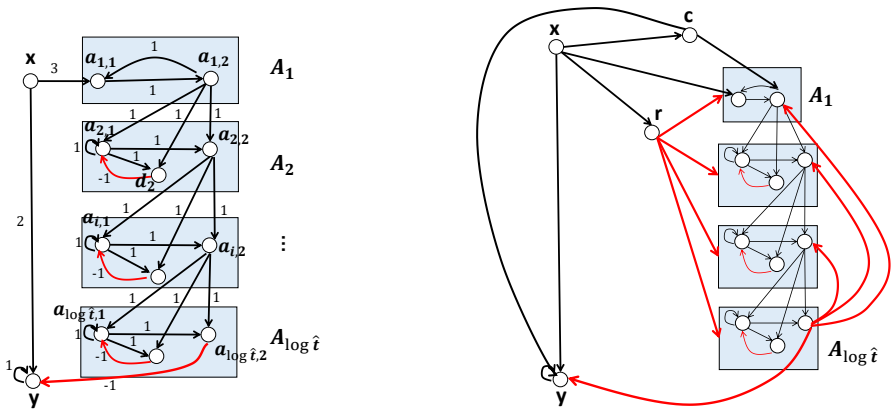

Figure 5: Illustration of the DetTimer network. Left: The simplified network for the case that x fired once. The neurons y, $a_{1,2}$ and the set of neurons $\{a_{1,1}, \ldots, a_{\log \hat{t},1}\}$ have threshold 1. For $i \geq 2$ the threshold of $a_{i,2}$ and d_i is 2. Right: A complete network description for the general case, where the input can fire several times during the execution. The reset neuron r resets the timer in case x fires several times. The control neuron c takes care of the special extreme case where x fires again one round *before* the last counting neuron $a_{\log \hat{t},2}$ fires.

A Missing Proofs for Det. Neural Timer

A.1 Complete Description of The DetTimer Network

Handling the General Case: We begin by extending the DetTimer(t) network to handle the case where x fired more than once within the execution.

- **Case 1: x fires several times within a span of t rounds.** We introduce an additional reset (inhibitory) neuron r that receives input from x with weight $w(x, r) = 1$, has outgoing edges to all neurons except $a_{1,2}$ and y with negative weight of -2, and threshold value $b(r) = 1$.

- **Case 2: x fires again just one round before $a_{\log \hat{t},2}$ fires.** To process this new spike, we introduce a control neuron c that receives input from x with weight $w(c, x) = 1$ and threshold $b(c) = 1$ and fires one round after x. The control neuron c has outgoing edges to y and $a_{1,2}$ with weight $w(c, y) = w(c, a_{1,2}) = 3$. Therefore even if $a_{\log \hat{t},2}$ fires one round after x, the control neuron will cancel the inhibition on the output y and on $a_{1,2}$ and the timer will continue to fire.

Figure 5 illustrates the structure of the network.

We next use Claim 1 in order to prove the the first part of Thm. 1.

A.2 Complete Proof of Thm. 1(1)

Proof. We start by considering the case where x fires once in round t'. If x fired in round t', due to the self loop of y, starting from round $t' + 1$, the output keeps firing as long as $a_{\log \hat{t},2}$ did not fire. By Claim 1, $a_{\log \hat{t},2}$ fires in round $t' + \hat{t} + \log \hat{t} - 1 = t' + t - 1$, and therefore y will not be inhibited in round $t' + t$. Note that $a_{\log \hat{t},2}$ also inhibits all other auxiliary neurons, and therefore as long as x will not fire again, y will also not fire. Next we consider the case where x also fired in round $t'' \geq t' + 1$.

- **Case 1:** $t'' \geq t' + t$. Because in round $t' + t - 1$ the neuron $a_{\log \hat{t},2}$ inhibits all counting neurons in the network, starting round $t' + t$ no counting neuron fires until x fires again in round t'' and this is the same as the previous case of the first firing.

- **Case 2:** $t'' \leq t' + t - 3$. In round $t'' + 1 \leq t' + t - 2$ the reset neuron r inhibits all counting neurons except for $a_{1,2}$. Hence, in round $t'' + 2$ only y and $a_{1,2}$ fire, and the neural timer continues to count for additional $t - 2$ rounds.

- **Case 3:** $t'' = t' + t - 1$. The neuron $a_{\log \hat{t},2}$ fires on the same round as x. Since the weight of the edge from x to y and $a_{1,1}$ is greater than the weight of the inhibition from $a_{\log \hat{t},2}$, the timer continues to fire based on the last firing event of x.

- **Case 4:** $t'' = t' + t - 2$. In this case x fires in round t'' and in the next round, $a_{\log \hat{t},2}$ fires and inhibits the output y (at the same round that the reset neuron r fires). Note that in round $t'' + 1$ the control neuron c also fires, and therefore in round $t'' + 2$ it excites y and $a_{1,2}$ canceling the inhibition of $a_{\log \hat{t},2}$ and the counting continues properly.

□

A.3 Useful Modifications of Deterministic Timers

We show a slightly modified variant of neural timer denoted by $\mathsf{DetTimer}^*$ which receives as input an additional set of $\log t$ neurons that encode the desired duration of the timer.

(1) Time Parameter as a Soft-Wired Input. The $\mathsf{DetTimer}$ construction is modified to receiving a time parameter $t' \leq t$ as a (soft) input to the network. That is, we assume that t is the upper limit on the time parameter. The same network can be used as a timer for any $t' \leq t$ rounds, and this t' can be given as an input to the network. In such a case, once the input neuron x fires, the output neuron y will fire for the t' consecutive rounds. The time parameter t' is given in its binary form using $\log t$ input neurons denoted as $z_1 \dots z_{\log t}$. We denote this network as $\mathsf{DetTimer}^*(t)$. The idea is that given time parameter t', we want to use only $\log(t'')$ layers out of the $\log t$, where $t'' = t' + \log(t')$ (we use t'' due to the $\log(t'')$ delay in the update of the timer). The modifications are as follows.

1. The time input neurons are set to be inhibitors.

2. The intermediate layer of neurons $c_1 \dots c_{\log t''}$ determine how many layers we should use. Each c_i has negative edges from $z_1, \dots, z_{\log t''}$ with weights $w(c_i, z_j) = -2^{j-1}$, and threshold $b(c_i) = -i - 1 - 2^{i-1}$. Hence c_i fires iff $i - 1 + 2^{i-1} \geq \mathsf{dec}(\bar{z}) = t'$.

3. We introduce $\log t''$ inhibitors $r_1, \dots r_{\log t''}$ in order to inhibit the output y after we count to t' and reached layer t''. Each r_i has incoming edges from c_i and $a_{i,1}$, and fires as an AND gate. Hence, each r_i fires only when the timer count reach $2^{i-1} + i - 1$ and $i - 1 + 2^{i-1} \geq t'$.

4. The output neuron y receives negative incoming edges from the neurons $r_1 \dots r_{\log t''}$ with weight $w(r_i, y) = -1$, and stops firing if at least one r_i fired in the previous round.

5. Every r_i also has negative outgoing edges to all counting neurons $a_{j,k} \quad k \in \{1, 2\}, j = 1 \dots \log t$ with weight $w(r_i, a_{j,k}) = -2$ in order to reset the timer when we finish counting to t'.

See Figure 6 for an illustration of $\mathsf{DetTimer}^*(t)$ network.

(2) Extension to Neural Counting. We next show a modification of the timer into a neural counter network $\mathsf{DetCounter}$ that instead of counting the number of rounds, counts the number of input spikes in a time interval of t rounds. This network also has $O(\log t)$ neurons. To improve upon this bound, we resort to approximation and in Appendix B, we combine the $\mathsf{DetCounter}$ network with the streaming algorithm of [Fla85] to provide an approximate counting network with $O(\log \log t + \log(1/\delta))$ neurons where δ is the error parameter. We next describe the required adaptation for constructing the network described in Lemma 1.

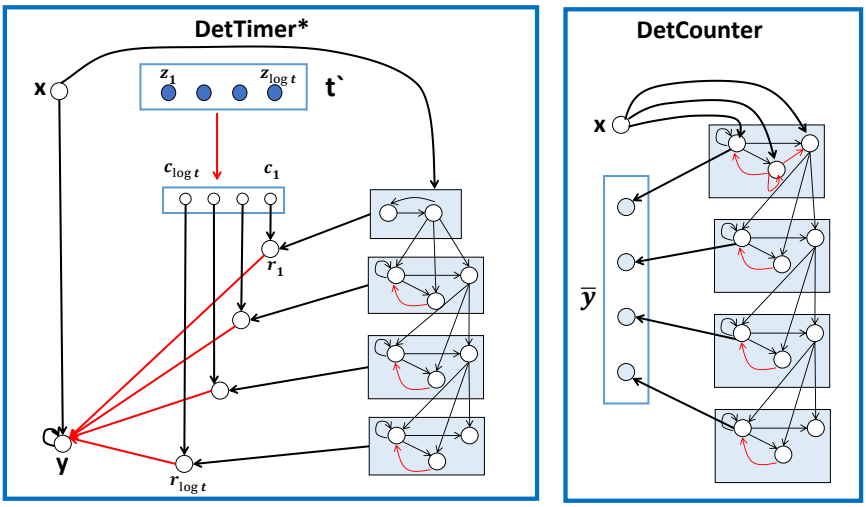

Figure 6: Left: Det. neural timer DetTimer* with a soft-wired time parameter. The input neurons $z_1, \ldots, z_{\log t}$ encode the time parameter t'. The intermediate neurons $c_1, \ldots, c_{\log t}$ control how many layers are used depending on the time parameter t'. Once the timer reaches layer $i = \Theta(\log(t'))$ for which c_i fires, the inhibitor r_i inhibits the output y and the counting terminates. Right: neural counter DetCounter, the output neurons $\bar{y}$ encode the number of times x fired in a time window of t rounds.

The DetCounter with parameter t contains $\log t$ layers, all layers $i \geq 2$ are the same as in DetTimer and only the first layer is slightly modified. The first counting neuron $a_{1,1}$ has a positive incoming edge from x with weight $w(x, a_{1,1}) = 4$, and a self loop with weight $w(a_{1,1}, a_{1,1}) = 1$. In addition $a_{1,1}$ has a negative edge from the inhibitor d_1 with weight $w(d_1, a_{1,1}) = -1$, and threshold $b(a_{1,1}) = 1$. The second counting neuron $a_{1,2}$ has positive edges from x and $a_{1,1}$ with weight $w(x, a_{1,2}) = w(a_{1,1}, a_{1,2}) = 1$, a negative edge from d_1 with weight $w(d_1, a_{1,2}) = -2$ and threshold $b(a_{1,2}) = 2$. The reset neuron d_1 is an inhibitor copy of $a_{1,2}$ and therefore also has positive edges from x and $a_{1,1}$ with weight $w(x, d_1) = w(a_{1,1}, d_1) = 1$, a negative self loop with weight $w(d_1, d_1) = -2$ and threshold $b(d_1) = 2$. We then connect the counting neurons $a_{1,1}, \cdots a_{\log t,1}$ to the output vector directly, where y_i has an incoming edge from $a_{i,1}$ with weight $w(a_{i,1}, y_i) = 1$ and threshold $b(y_i) = 1$. Figure 6 demonstrate the DetCounter(t) network.

We will next show that once the counter is updated, the number of times that x fired is represented as a binary number where the counting neuron $a_{i,1}$ represents the i^{th} bit in the binary representation (where $a_{1,1}$ is the least significant bit). We note that if the last firing of x is in round τ then after at most $\log c + 1$ rounds the counter will be updated with the new value, where c is the value of the counter before round τ. We start by showing the following claim concerning the first layer.

Claim 8. *If x fired in round τ, neurons d_1 and $a_{1,2}$ fire in round $\tau+1$ iff x fired an even number of times by round τ.*

Proof. By induction on the number of times x fired, denoted as n. Since d_1 and $a_{1,2}$ have identical potential functions it is sufficient to prove the claim for the neuron d_1. Base case: for $n = 1$, if x fired once in round τ, then $a_{1,1}$ fires for the first time in round $\tau + 1$, and since d_1 fires only if $a_{1,1}$ fired in the previous round, in round $\tau + 1$ both neuron d_1 and $a_{1,2}$ are idle. For $n = 2$, since x fired for the first time in some round $\tau' \leq \tau - 1$, starting round $\tau' + 1$ neuron $a_{1,1}$ fires on every round until d_1 fires. Hence, in round $\tau + 1$ the neuron d_1 receives spikes from both x and $a_{1,1}$ and therefore fires. Assume the claim holds for every $k \leq n - 1$ and we will show correctness for n. Denote the round in which x fired for the $(n-1)^{th}$ time by $\tau' \leq \tau - 1$.

- (Case 1: n is even.) Since $n-1$ is odd, by the induction assumption d_1 did not fire in round $\tau'+1$. Hence $a_{1,1}$ is not inhibited until round $\tau+1$, and due to the self loop $a_{1,1}$ also fires in round τ. Therefore d_1 and $a_{1,2}$ fire in round $\tau+1$.
- (Case 2: n is odd.) If $\tau' = \tau - 1$, by the induction assumption d_1 fires in round $\tau'+1 = \tau$, and due to the negative edges from d_1, both d_1 and $a_{1,2}$ will idle in round $\tau+1$. Otherwise, $\tau' \leq \tau - 2$. By the induction assumption, d_1 fires in round $\tau'+1$. Since x did not fire in round $\tau'+1$ (as it will fire again only in round τ), in round $\tau'+2 \leq \tau$ the neuron $a_{1,1}$ is inhibited by d_1 and therefore in round τ the neurons d_1 and $a_{1,2}$ receives a signal only from x and will not fire.

□

Next, we will show that if x fired in round τ for the last time, for each layer $i \in [1, \log n]$, neuron $a_{i,2}$ fires in round $\tau+i$ only if x fired $\ell \cdot 2^{i-1}$ times by round τ for some integer $\ell \geq 1$.

Claim 9. *For every layer $i \in [2, \log t]$ if $a_{i-1,2}$ fired in round τ for the n^{th} time, the neurons d_i and $a_{i,2}$ fire in round $\tau+1$ iff n is even.*

Proof. By induction on n. For $n=1$, one round after the first time neuron $a_{i-1,2}$ fires, the neuron $a_{i,1}$ fires for the first time, and therefore $a_{i,2}$, d_i do not fire. For $n=2$, the second time $a_{i-1,2}$ fires, due to the self loop on $a_{i,1}$, it will fire as well and therefore after one round $a_{i,2}$, d_i fire. Assume that $a_{i-1,2}$ fired in round τ' for the $(n-1)^{th}$ time. If n is even, then by the induction assumption d_i does not fire in round $\tau'+1 \leq \tau$. Hence, due to the self loop of $a_{i,1}$, in round τ also $a_{i,1}$ fires and therefore d_i and $a_{i,2}$ fire in round $\tau+1$. If n is odd, by the induction assumption d_i fires in round $\tau'+1$. By Claim 8 there is at least one round distance between every two firing events of $a_{1,2}$. Thus, there is at least one round distance between every two firing events of $a_{i-1,2}$, and therefore $\tau \geq \tau'+2$. Hence, because $a_{i,1}$ was inhibited by d_i in round $\tau'+1 < \tau$, it is idle in round τ and the neurons d_i and $a_{i,2}$ do not fire in round $\tau+1$. □

Corollary 1. *If x fired for the n^{th} time in round τ, for every layer $i \in [1, \log t]$ the neurons d_i and $a_{i,2}$ fire in round $\tau+i$ iff $(n \bmod 2^i) = 0$.*

Proof. By induction on i. The base cases for $i=1$ follows from Claim 8. Assume that the claim holds for layer i and we will show it also holds for layer $i+1$. If $(n \bmod 2^i) = 0$, then $n = q \cdot 2 \cdot 2^{i-1}$ for some integer q. Therefore by the induction assumption, $a_{i,2}$ fires in round $\tau+i$, and moreover it fired an even number of times by that round. Hence, by Claim 9 the neurons d_{i+1} and $a_{i+1,2}$ fire in round $\tau+i+1$. Otherwise, if $(n \bmod 2^i) \neq 0$, by the induction assumption $a_{i,2}$ does not fire in round $\tau+i$ and therefore d_{i+1} and $a_{i+1,2}$ will not fire in round $\tau+i+1$. If $(n \bmod 2^i) = 0$ but $(n \bmod 2^{i+1}) \neq 0$, then by the induction assumption $a_{i,2}$ fired an odd number of times by round $\tau+i$ and by Claim 9 neurons d_i and $a_{i,2}$ will not fire in round $\tau+i+1$. □

The first counting neuron $a_{i,1}$ fires one round after $a_{i-1,2}$ fires, and as long as d_i and $a_{i,2}$ did not fire. Hence we can conclude that if x fired for the last time in round τ, by round $\tau+\log r_\tau+1$, the neurons $a_{1,1}, \ldots, a_{\log t,1}$ hold a binary representation of the number of times r_τ that x fired by round τ.

B Approximate Counting

In this section, we provide improved constructions for neural counters by allowing approximation and randomness. Our construction is inspired by the *approximate counting* algorithm of Morris as presented in [Mor78, Fla85] for the setting of dynamic streaming. The idea is to implement a counter which holds the logarithm of the number of spikes with respect to base $\alpha = 1 + \Theta(\delta)$. The approximate neural counter problem is defined as follows.

Definition 5 ((Approximate) Neural Counter)**.** *Given a time parameter t and an error probability δ, an approximate neural counter has an input neuron x, a collection of $\log t$ output neurons represented by a vector $\bar{y}$, and additional auxiliary neurons. The network satisfies that in a time window of t rounds, in every given round, the output $\bar{y}$ encodes a constant approximation for the number of times x fired up to that round, with probability at least $1-\delta$.*

Throughout, we assume that $1/\delta < t$. For smaller values of δ, it is preferable to use the deterministic network construction of DetCounter with $O(\log t)$ neurons described in Lemma 1. For the sake of simplicity, we first describe the construction under the following promises:

- (S1) The firing events of x are sufficiently spaced in time, that is there are $\Omega(\log t)$ rounds between two consecutive firing events.
- (S2) The state of $\bar{y}$ encodes the right approximation in every round τ such that the last firing of x occurred before round $\tau - \log r_\tau$ where r_τ is the number of x's spikes up to round τ.

High Level Description. The network ApproxCounter(t, δ) consists of two parts, one for handling small number of spikes by the input x and one for handling the large counts. The first part that handle the small number of spikes is deterministic. Specifically, as long as the number of spikes by x is smaller than $s = \Theta(1/\delta^2)$, we count them using the exact neural counter network (presented in Appendix A.3), using $O(\log 1/\delta)$ neurons. We call this module *Small Counter SC* and it is implemented by the DetCounter network with time parameter $\Theta(1/\delta^2)$.

To handle the large number of spikes, we introduce the *Approximate Counter AC* implemented by DetCounter with time parameter $\log_\alpha t$ which approximates the logarithm of the number of rounds x fired with respect to base $\alpha = 1 + \Theta(\delta)$. This module is randomized, and will provide a good estimate for the count provided that it is sufficiently large. The idea is to update the AC module (by adding a $+1$) upon every firing event of x with probability of $\frac{1}{1+\alpha^c}$ where c is the current value stored in the counter and $\alpha = 1 + \Theta(\delta)$. To do so, we have a spiking neuron a^* that has incoming edges from the output of AC module, and fires with the desired probability. The reason we use probability $\frac{1}{1+\alpha^c}$ instead of $\frac{1}{\alpha^c}$ as suggested in Morris algorithm, is due to the sigmoid probability function of spiking neurons (see Eq. (1)). Once the count is large enough (more than s), the network starts using AC. This is done by introducing an indicator neuron v_I, indicating that the small-counter is full. This neuron starts firing after SC is full, and keeps on firing due to its self loop.

The input neuron of AC, denoted as x_{ac} computes an AND of the input x, the spiking neuron a^* and the indicator neuron v_I. In addition, v_I initiates a reset of the small counter SC to make sure that the output $\bar{y}$ receives only information from the large-count module AC. Figure 7 provides a schematic description of the construction.

Detailed Description. Let r_n be the number of times x fired in the first n rounds, and let $\alpha = 1 + \Theta(\delta)$ be the base of the counting in the approximate counting module.

- **Handling Small Counts.** The module *Small-Counter SC* is implemented by the DetCounter module with time parameter s and the input coming from x, where $s = \frac{1}{\delta(\alpha-1)}$. Since $\alpha = 1 + \Theta(\delta)$, it holds that $s = \Theta(1/\delta^2)$. In addition, we introduce an excitatory *indicator* neuron v_I that has an incoming edge from the last layer of SC (i.e. neuron $a_{\log s,2}$) as well as a self loop, each with weight 1 and threshold $b(v_I) = 1$. The indicator neuron v_I has an outgoing edge to an inhibitory *reset* neuron v_r with weight $w(v_I, v_r) = 1$, which is connected to all neurons in SC with negative weight -5. The reset neuron v_r also has an incoming edge from $a_{\log s,2}$ with weight 1 and threshold $b(v_r) = 1$. As a result, one round after SC reaches value s, it is inhibited.
- **Handling Large Counts.** The *Approximate-Counter AC* is implemented by a DetCounter module with time parameter $\log_\alpha t$, and its input neuron is denoted by x_{ac}. Denote by

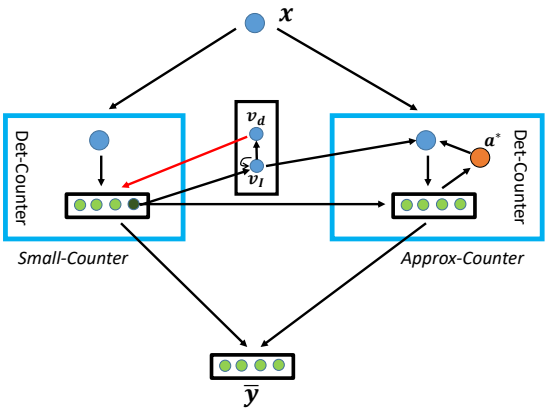

Figure 7: Schematic description of network ApproxCounter. In each module only the input and output layers are shown. The Small-Counter module SC is responsible for counting up to $\Theta(\log 1/\delta)$ spikes, and is implemented by the DetCounter module with time parameter $(\Theta(1/\delta))$. For handling large counts, we use the Approx-Counter AC module implemented by the DetCounter module with time parameter $\Theta(\log t/\delta)$. The Approx-Counter module simulates Morris' algorithm and maintain an estimate for the logarithm of the spike counts. The neurons v_I and v_r switch between the two stages (small and large counts) of the execution.

$\ell = \log_2 \log_\alpha t$ the number of layers in the AC module, and for every $1 \leq i \leq \ell$, denote the counting neuron $a_{i,1}$ by c_i. To initialize the counter we connect the last output neuron of SC to the counting neurons $c_{i_1} \dots c_{i_k}$ in AC which correspond to the binary representation of $\log_\alpha(1/\delta + 1)$ with weights 5. We introduce a probabilistic spiking neuron a^* that will be used to increase the counter with the desired probability. In order for a^* to receive negative weights from AC, we connect each counting neuron c_i to an inhibitor copy $c_{i,2}$ with weight $w(c_i, c_{i,2}) = 1$ and threshold 1. We then connect the inhibitors $c_{1,2}, \dots, c_{\ell,2}$ to a^* with weights $w(c_{i,2}, a^*) = -2^{i-1} \cdot \ln \alpha$, and set $b(a^*) = 0$. Hence, a^* fires in round τ with probability $\frac{1}{1+\alpha^c}$, where c is the value of AC in round $\tau - 1$. Finally, the input neuron x_{ac} has incoming edges from a^*, x and v_I each with weight 1 and threshold $b(x_{ac}) = 3$. As a result, x_{ac} fires only if v_I, x and a^* fired in the previous round.

- **The Output Neurons.** The counter modules SC and AC are connected to the output vector $\bar{y}$ as follows. Each y_i has incoming edges from neurons $c_1, \dots, c_\ell$ with weight $w(c_i, y) = \log \alpha \cdot 2^{i-1}$, and threshold $b(y_i) = i + \log(\alpha - 1)$. In addition, each output neuron y_i has an incoming edge from the i^{th} output of SC with weight $b(y_i)$. Hence, y_i fires in round τ if either $\log \alpha \cdot (\sum_{j=1}^{\ell} c_j \cdot 2^{j-1}) - \log(\alpha - 1) \leq i$, or the i^{th} output of SC fired in the previous round.

Size Bound. All neurons except the spiking neuron a^* are threshold gates. Recall that $\alpha = 1 + \Theta(\delta)$. Hence the size of the counter AC is $O(\log_2 \log_\alpha t) = O(\log \log t + \log(1/\delta))$. Since the size of the counter SC is $O(\log 1/\delta)$, overall we have $O(\log \log t + \log(1/\delta))$ neurons.

Correctness Analysis (under the simplifying assumptions). We first show the correctness of the ApproxCounter construction under the two promises. At the end of the section we will show correctness for the general case as well. Let r_τ be the number of times x fired up to round τ. If $r_\tau \leq s$ the correctness of ApproxCounter follows from the correctness of the DetCounter construction (see Lemma 1). From now on, we assume $r_\tau \geq s + 1$. Let z_n be the random variable for the value of AC after x fired n times (i.e when $r_\tau = n$). We start by bounding the expectation of α^{z_n}.

Claim 10. $\mathbb{E}[\alpha^{z_n}] \in [n(\alpha - 1)(1 - \delta) + 1, n(\alpha - 1) + 1]$.

Proof. The *AC* counter starts with $n = s = \frac{1}{\delta(\alpha-1)}$ spikes, and we initiate the counter with value $c = \log_\alpha(1/\delta+1)$. Hence for $n = s$, we get $\alpha^{z_n} = n(\alpha-1)+1$ and the claim holds. For $n \geq s+1$ we get

$$\begin{aligned}
\mathbb{E}[\alpha^{z_n}] &= \sum_{j=c}^{n-1} \mathbb{E}[\alpha^{z_n} \mid z_{n-1} = j] \cdot \Pr[z_{n-1} = j] \\
&= \sum_{j=c}^{n-1} \Pr[z_{n-1} = j] \cdot (\alpha^{j+1} \cdot \frac{1}{\alpha^j + 1} + \alpha^j \cdot (1 - \frac{1}{\alpha^j + 1})) \\
&= \mathbb{E}[\alpha^{z_{n-1}}] + (\alpha - 1) \cdot \sum_{j=c}^{n-1} \Pr[z_{n-1} = j] \cdot (\frac{\alpha^j}{1+\alpha^j}) \ . \qquad (3)
\end{aligned}$$

Note that for $j \geq c$, it holds that $1 > \frac{\alpha^j}{1+\alpha^j} > 1 - \delta$. Therefore

$$\sum_{j=c}^{n-1} \Pr[z_{n-1} = j] \cdot (\frac{\alpha^j}{1+\alpha^j}) \in [1-\delta, 1] \ .$$

By combining this with Eq. (3) we get that $\mathbb{E}[\alpha^{z_n}] \in [n(\alpha-1)(1-\delta), n(\alpha-1)+1]$. □

Claim 11. $\Pr[|\alpha^{z_n} - \mu| > 1/2 \cdot \mu] \leq \delta$, *where* $\mu = \mathbb{E}[\alpha^{z_n}]$.

Proof. We will use Chebyshev's inequality, and start by computing $\mathbb{E}[\alpha^{2z_n}]$ in order to bound the variance of α^{z_n}.

$$\begin{aligned}
\mathbb{E}[\alpha^{2z_n}] &= \sum_{j=c}^{n-1} \mathbb{E}[\alpha^{2z_n} \mid z_{n-1} = j] \cdot \Pr[z_{n-1} = j] \\
&= \sum_{j=c}^{n-1} \Pr[z_{n-1} = j] \cdot \left(\alpha^{2j+2} \cdot \frac{1}{\alpha^j + 1} + \alpha^{2j} \cdot (1 - \frac{1}{\alpha^j + 1})\right) \\
&= \mathbb{E}[\alpha^{2z_{n-1}}] + \sum_{j=c}^{n-1} \Pr[z_{n-1} = j] \cdot (\frac{\alpha^{2j}(\alpha^2-1)}{\alpha^j + 1}) \leq \mathbb{E}[\alpha^{2z_{n-1}}] + (\alpha^2-1)\mathbb{E}[\alpha^{z_{n-1}}] \\
&\leq \mathbb{E}[\alpha^{2z_{n-1}}] + (\alpha^2-1) \cdot ((n-1)(\alpha-1)+1) \ , \qquad (4)
\end{aligned}$$

where Ineq. (4) is due to Claim 10. For $n = s$, it holds that

$$\mathbb{E}[\alpha^{2z_s}] = s^2(\alpha-1)^2 + 2s(\alpha-1) + 1 \leq (\alpha+1)(\alpha-1)\sum_{i=1}^{s} i + (\alpha-1)(\alpha+1)s \ ,$$

and combined with Eq. (4) we get

$$\mathbb{E}[\alpha^{2z_n}] \leq \frac{1}{2}\left(n(3\alpha^2 - \alpha^3 + \alpha - 3) + n^2(\alpha+1)(\alpha-1)^2\right) \ .$$

Therefore the variance is bounded by

$$\begin{aligned}
Var[\alpha^{z_n}] &= \mathbb{E}[\alpha^{2z_n}] - (\mathbb{E}[\alpha^{z_n}])^2 \\
&\leq \frac{1}{2}n^2(\alpha-1)^2\left((\alpha-1)+\delta^2\right) + n\left((\alpha-1)(2\alpha+1-a^2)+2\alpha\delta\right) \ .
\end{aligned}$$

Using Chebyshev's inequality and Claim 10 we can now conclude the following:

$$\begin{aligned}
\Pr[|\alpha^{z_n} - \mu| \geq 1/2 \cdot \mu] &\leq \frac{Var[\alpha^{z_n}]}{((1/2)\cdot\mu)^2} \leq \frac{4Var[\alpha^{z_n}]}{n^2(\alpha-1)^2(1-\delta)^2} \\
&\leq 4\left((\alpha-1)+\delta^2\right) + \frac{8n\,(\alpha-1+2\delta)}{n^2(\alpha-1)^2(1-\delta)^2} \ , \qquad (5)
\end{aligned}$$

since we assume $n \geq s = 1/\delta(\alpha-1)$ it holds that $n \leq n^2(\alpha-1)\delta$. As a result, by Eq. (5) we get:

$$\Pr[|\alpha^{z_n} - \mu| \geq 1/2 \cdot \mu] \quad \leq \quad 4\left((\alpha-1)+\delta^2\right) + 10\delta\left(1 + 2\delta/(\alpha-1)\right) \ .$$

Since $\alpha = 1 + \Theta(\delta)$, we have that $Var[\alpha^{z_n}] \leq \Theta(\delta)$. We can use $\delta' = \Theta(\delta)$ in our construction and set parameter α accordingly in order to achieve

$$\Pr[|\alpha^{z_n} - \mu| \geq 1/2 \cdot \mu] \leq \delta \ .$$

□

Combining Claim 10 and Claim 11 we conclude that $\alpha^{z_n} \in [n(\alpha-1)/4, 2n(\alpha-1)]$ with probability at least $1-\delta$. Let $S = \log\alpha \cdot z_n - \log(\alpha-1)$. Thus, $S \in [\log(n/4), \log(2n)]$. Recall that after SC gets reset, each y_i fires only if $\log\alpha \cdot z_n - \log(\alpha-1) \leq i$. As a result, the value of the output $\bar{y}$ is given by

$$\mathsf{dec}(\bar{y}) = \sum_{i=1}^{S} 2^i = 2^{S+1} - 2 \in [n/2-2, 4n-1] \ ,$$

which is a constant approximation of n as desired.

Adaptation to the General Case. We now explain the modifications needed to handle the general case without the two simplifying assumptions. In order to fire with the correct probability without the spacing guarantee, every time we increase AC, we wait until its value gets updated before we attempt to increase it again. In order for the output $\bar{y}$ to output the correct value also during the update of the counter AC, we introduce an intermediate layer of neurons $c''_1, \ldots, c''_\ell$ that will hold the previous state of AC during the update.

- **Removing Assumption (S1):** In the DetCounter construction, we say that there are k *active layers* in round τ if the value of the counter in round τ is at most 2^k and no neuron in layer $j \geq k+1$ fired. Once we increase the counter, after at most $k+1$ rounds the value is updated. During this update operation, the network waits and ignores spikes from x that might occur during this time window. To implement this waiting, we introduce a *Wait-Timer* (WT) module which uses the DetTimer* module[17]. This DetTimer* gets an input from x_{ac} and the time parameter input $\bar{q}$ with $\log\ell$ neurons where $\ell = \log_2\log_\alpha t$ is the number of layers in the module AC. The counting neurons $c_1, \ldots, c_\ell$ of AC are connected to $\bar{q}$ as follows. Each q_i has an incoming edge from $c_{2^{i-1}}$ with weight $w(c_{2^{i-1}}, q_i) = 1$ and threshold $b(q_i) = 1$. Hence, the value of $\bar{q}$ is at least $k+1$ and at most $4k$ where k is the number of active layers in AC. In order for the time parameter to stay stable throughout the update, for each q_i we add a self loop with weight $w(q_i, q_i) = 1$. The WT module has two outputs, an inhibitor g_r which fires as long as the timer did not finish the count, and an excitatory g which fires after the count is over. We connect r_r to x_{ac} with weight $w(g_r, x_{ac}) = -5$, preventing it from firing while the counter is not updated. We connect g to an additional inhibitor neuron q_r which inhibits the time parameter neurons $q_1, \ldots, q_\ell$ one round after we finished the count. The size of WT is $O(\log\ell) = O(\log\log 1/\delta + \log\log\log t)$.

- **Removing Assumption (S2):** Two copies of the counting neurons $c_1, \ldots, c_\ell$ are introduced. The first copy $c'_1, \ldots c'_\ell$ allows us to copy the state of the counter AC once its update proceess is complete. Each c'_i has incoming edges from c_i and the excitatory output of the WT module, each with weight $w(g, c'_i) = w(c_i, c'_i) = 1$ and threshold $b(c'_i) = 2$. Thus, c'_i fires iff in the previous round both c_i and g fired (implying that neuron c_i was active when the counter finished the update). The second copy $c''_1, \ldots, c''_\ell$ hold the previous state

[17] Recall that DetTimer* is a variant of the neural timer in which the time parameter is given as a soft-wired input and the upper bound on this input time is hard coded in the network.

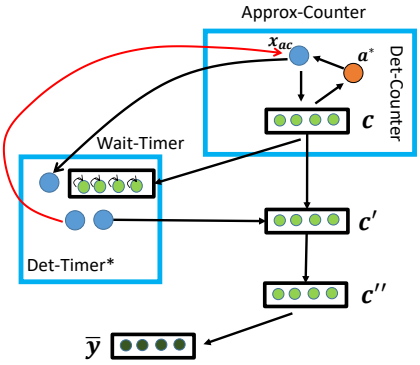

Figure 8: A description of the modifications in ApproxCounter (to handle the general case). The *Wait-Timer* (WT) module is implemented as a DetTimer* with input from x_{ac} and time input from the counting neurons of AC. The WT module has two output neurons, one inhibitor and one excitatory. The inhibitor output of WT inhibits the input neuron x_{ac}, preventing it from firing during the update process of the AC counter. In addition, we have two copies of the counting neurons of AC denoted as c' and c''. These copies are used for the output vector $\bar{y}$ to receive a correct input from AC at all times, even during the update process of the AC counter. Once the WT module finishes its count, in order to copy the information from c to c'', we use c' as that OR gates between c and the excitatory output of the module WT.

of AC during the update of the AC module. Each c''_i has an incoming edge from c'_i with weight 2, a self loop with weight 1, a negative edge from the inhibitor g_r with the weight (-1) and threshold 1. Note that the inhibition of c''_i occurs on the same round it receives the updated state from neuron c'_i. Finally, the output layer $\bar{y}$ has incoming edges from neurons $c''_1, \ldots, c''_\ell$ instead of $c_1, \ldots, c_\ell$ with the same weights.

Figure 8 illustrates the modifications made to handle the general case.

Proof of Thm. 4 (for the general case). Assume x fired n times up to round τ. If $n \leq s$ we count the number of times x fired explicitly via the SC module. We note that in round τ the counter might be still updating the last $O(\log n)$ spikes of x. Hence, by the DetCounter construction, the value of the counter is at least $\frac{n-\log n}{2} = \Theta(n)$, and therefore we indeed output a constant approximation of n with probability 1.

Otherwise, $n \geq s$. First note that when we switch from the SC to the AC counter, we might omit at most $\Theta(\log 1/\delta)$ spikes of x due to the delay in the DetCounter module. Since $n \geq s = \Theta(1/\delta^2)$ this is negligible, as we want a constant approximation. Next, we bound the number of times x might have fired during the rounds in which the wait module WT was active. As we only omit attempts to increase the counter, by Claim 11 with probability at least $1 - \delta$, the value of counter has been increased for at most $\log_\alpha(2n(\alpha - 1))$ times.

Each time that the counter value is increased, the waiting module WT is active for at most $4 \log_2 \log_\alpha 2n(\alpha - 1) \leq 4 \log n$ rounds. Thus, in total we omit at most $4 \log n \cdot \log_\alpha(2n(\alpha - 1)) < 4\sqrt{n} \log^2 n$ spike events. In addition, since the copy neurons $c''_1, \ldots, c''_\ell$ might hold the previous value of the counter in round τ, we might lose another factor of two in the output layer. All together, in round τ the output $\bar{y}$ holds a constant approximation of n and Theorem 4 holds for the general case as well.

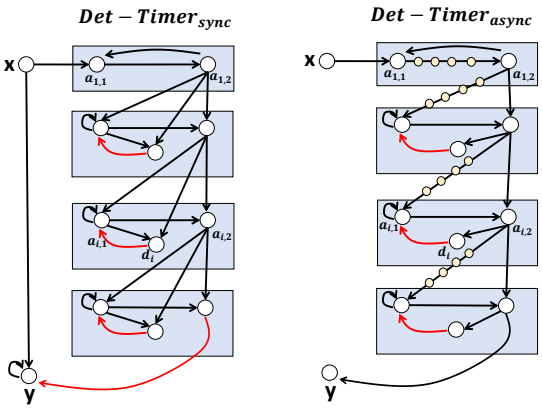

Figure 9: $\mathsf{DetTimer}_{\mathsf{async}}$ versus $\mathsf{DetTimer}_{\mathsf{sync}}$. Left: The deterministic timer $\mathsf{DetTimer}_{\mathsf{sync}}$ network. Right: The modified $\mathsf{DetTimer}_{\mathsf{async}}$ network which works in the asynchronous setting. We add a chain of L neurons in the first layer and between neurons $a_{i-1,2}$ and $a_{i,1}$, where L is the upper bound on the response latency of a single edge in the asynchronous setting.

C Missing Details for Synchronizers

C.1 Missing Details for the Asynchronous Analog of DetTimer

Proof of Lemma 3. The construction starts with $t' = t/2L$ layers of the DetTimer network. These layers are modified as follows (see Figure 9 for comparison with the standard construction).

- Neurons $a_{1,1}$ and $a_{1,2}$ are connected by a chain of length $4L$. all neurons in the chain as well as $a_{1,2}$ have an incoming edge from the previous neuron in the chain with weight 1 and threshold 1.
- For every $i \geq 2$, the inhibitor neuron d_i has an incoming edge only from $a_{i,2}$ with weight $w(a_{i,2}, d_i) = 1$ and threshold $b(d_i) = 1$.
- For every $i \geq 1$, the neurons $a_{i-1,2}$ and $a_{i,1}$ are connected by a chain of length L, instead of a direct edge, where the weight of the edge from the end of the chain to $a_{i,1}$, is 1.
- The neuron $a_{\log t',2}$ is an excitatory (rather than an inhibitory) neuron, and the output neuron y has one incoming edge from $a_{\log t',2}$ with weight $w(a_{\log t',2}, y) = 1$ and threshold $b(y) = 1$.
- A newly introduced inhibitor neuron r that has an incoming edge from $a_{\log t',2}$ with weight $w(a_{\log t',2}, r) = 1$, threshold $b(r) = 1$, and negative outgoing edges to all neurons in the timer with weight -2 for clean-up purpose.

The correctness is based on the following auxiliary claim.

Claim 12. *Fix a layer $i \geq 2$. Assume that (1) $a_{i-1,2}$ fired for the first time in round f_{i-1}, and that (2) it fires every τ_{i-1} rounds. It then holds that (1a) $a_{i,2}$ fires for the first time in round f_i for $f_i \in [f_{i-1} + \tau_{i-1} + 1, f_{i-1} + \tau_{i-1} + L^2 + L]$, and that (1b) $a_{i,2}$ fires from that point on for every $\tau_i \in [2 \cdot \tau_{i-1}, 2 \cdot \tau_{i-1} + (L^2 + L)]$ rounds.*

Proof. Assume that neuron $a_{i-1,2}$ fires every τ_{i-1} rounds starting at round f_{i-1}. It then holds that $a_{i,2}$ gets the spike from $a_{i-1,2}$ strictly *before* the spike of $a_{i,1}$. Specifically, it gets the spike from $a_{i-1,2}$ by round $\tau \leq f_{i-1} + L$, and it gets the spike from $a_{i,1}$ in some round $\tau' \geq f_{i-1} + L + 1$. Note that it is crucial that the spike from $a_{i-1,2}$ will arrive earlier to $a_{i,2}$, as otherwise $a_{i,2}$ will fire in round τ. As a result, the first time $a_{i,2}$ fires is after round $f_{i-1} + \tau_{i-1} + 1$ and therefore

$f_i \geq f_{i-1} + \tau_{i-1} + 1$. Due to the self loop on $a_{i,1}$, neuron $a_{i,2}$ gets a spike from $a_{i,1}$ in every round $\tau'' \geq \tau'$. Because we have fixed latencies, $a_{i,2}$ gets a signal from $a_{i-1,2}$ every τ_{i-1} rounds, and therefore $a_{i,2}$ will fire by round $\tau' + \tau_{i-1}$. Since each edge has latency of at most L, it holds that $\tau' \leq f_{i-1} + L^2 + L$, hence $f_i \leq f_{i-1} + L^2 + L + \tau_{i-1}$ and (1a) follows.

We now show (1b). We first observe that $a_{i,1}$ stops firing at least L rounds *before* the next firing of $a_{i-1,2}$. This holds since once $a_{i,2}$ fires in round f_i, after at most L rounds the inhibitor d_i fires, and after at most $2L$ rounds neuron $a_{i,1}$ is inhibited. Since $\tau_j \geq 4L$ (due to the chain in the first layer), it indeed holds that in the next round when $a_{i-1,2}$ fires, no neuron in layer i will fire. Since the latency of each edge is fixed and $a_{i-1,2}$ fires every τ_{i-1} rounds by our assumption, we get that $a_{i,2}$ fires every τ_i rounds where $\tau_i \in [2\tau_{i-1}, 2\tau_{i-1} + L^2 + L]$. □

Claim 13. *Assume that x fired in round τ_0. Then for every $i \geq 1$ it holds that: (1) the neuron $a_{i,2}$ fires for the first time during the interval $[\tau_0 + 2^i \cdot 2L, \tau_0 + 2^i \cdot 8L^2]$ and (2) it fires every τ_i rounds for $\tau_i \in [2^i \cdot 2L, 2^i \cdot (4L^2)]$.*

Proof. Once the input neuron x fired in round τ_0, the neuron $a_{1,2}$ fires for the first time in round $f_1 \in [\tau_0 + 4L, \tau_0 + 4L^2 + L]$ and continue to fire every τ_1 rounds for $\tau_1 \in [4L, 4L^2 + L]$. This is due to the chain between $a_{1,1}$ and $a_{1,2}$ and the fact that the latency $\ell(e)$ is fixed for every e. Using Claim 12 in an inductive manner, we conclude that for every $i \geq 1$: (1) $a_{i,2}$ fires every $\tau_i \in [2^i \cdot 2L, 2^i \cdot 4L^2]$ rounds, (2) $a_{i,2}$ fires for the first time in round $f_i \in [\tau_0 + 2^i \cdot 2L, \tau_0 + 2^i \cdot 8L^2]$. □

Since the edge between neuron $a_{\log t',2}$ and the output neuron has latency of at most L, we conclude that if the input neuron x fires in round τ_0, the output neuron fires in round $\tau \in [\tau_0 + 2Lt', \tau_0 + 9L^2t']$. Because $t' = t/2L$, given that the input x fired in round τ, the output neuron fires between round $\tau + t$ and round $\tau + 5Lt$ and Lemma 3 follows.