Neighbour Discovery primitives in Low Power Wireless Area Networks

Michele Grisafi, Mat. number 203297
michele.grisafi@studenti.unitn.it

I. Introduction

This paper, and the work it proposes, aims at implementing two basic neighbour discovery protocols for a Wireless Sensor Network (WSN). The goal of the two protocols is the discovery of the highest number of neighbours in a WSN, while respecting the principles of low powered WSNs, thus introducing a trade-off between discovery rate and power consumption, both equally important. The implementation, although it is driven by some requirements and guidelines, follows some design choices based on principles common to WSNs and it introduces some possible strategies to optimise the discovery reducing the number of collisions, one of the main challenges in this type of network communication.

The rest of the paper is structured as follows: section II gives an overview on the two protocols and the instruments used to implement them, section III describes some of the implementation choices and, finally, section IV evaluates the two approaches.

II. Fundamentals

The following subsections will tackle some of the fundamentals behind the design choices taken by the work proposed by this paper. All of the protocols and theory hereby proposed have been taken as guidelines for the development of the neighbour discovery primitives.

A. Project requirements and guidelines

Neighbour discovery is the process during which nodes in range of each other, through radio communication, signal each other presence thus becoming aware of the neighbours around them. Among the various types of neighbour discovery (ND) the work of this paper pursues the design of two asynchronous unidirectional protocols. Asynchronicity is defined by the lack of any type of coordination between the nodes, which act solely on the base of some static intervals[1] scheduling their reception and transmission windows accordingly. The unidirectionality of the discovery is defined by the following relation: given any two nodes *A* and *B* in a network, A discovers B does not imply that B discovers A. These characteristics greatly simplify the design principles behind the two ND primitives proposed by this paper. The two primitives are called *Burst* and *Scatter* and are both organised in epochs, which are the interval of time during which a node can discover its surrounding nodes. Epochs are divided in time slots which are used exclusively either for the transmission or for the reception of beacons. Beacons are short transmissions of single packets containing only the recipient identifier, used by the nodes to discover each other. During the reception, if one such beacon is received and correctly decoded then the neighbour encoded in its payload will be added to the list of discovered nodes of the receiving node.

Alternating receiving slots to transmission slots allows nodes to create a discovery protocol that does not require any preamble or more sophisticated transmission schema. However, beacons are a broadcast transmission that is likely to collide with other beacons. The protocol lacks synchronicity, thus increasing the scalability of the protocol while negatively impacting the collisions caused by the overlapping of transmission slots. Finally, the basic primitives lack of any collision avoidance technique.

1) Burst ND: The first primitive is called Burst and owes its name to way it handles the transmission (TX) slots. During the entire epoch, each node uses only one of the available slots for the transmission of beacons which the node keeps on broadcasting with an arbitrary frequency for the duration of the whole slot. The remaining slots are then used for the reception of other nodes' beacons, by listening to the channel, although only for an arbitrary fraction of the slot.

2) Scatter ND: The second primitive is called Scatter and it inverts the ratio of TX and RX slots with nodes that use exclusively one slot for the reception throughout the entire epoch. Contrarily to Burst mode, the single reception slot is used intensively by the radio that listen for incoming beacons for its entire duration. As for the transmission, each remaining slot is used to send a single beacon, right at the beginning of the slot.

The basic design of the two discovery primitives are depicted in Figure 1 in the Appendices.

B. Contiki

The proposed work is implemented using Contiki, an Operating System (OS) for Internet of Things (IoT). Contiki offers the programmer a plethora of different functions to easily interact with the hardware of the device on which it is installed. The presence of an OS allows the programmer to easily use functionalities such as timers and callbacks without needing to operate with the devices registers and hardware controllers. Most importantly, Contiki offers some low-level primitives to interact with the radio and other hardware mounted on the Micro Controller Unit (MCU), without leaving to the

[1] Due to clock drift the intervals are bound to shift in time and cannot be considered perfectly equals between different nodes.

programmer the burden of setting registers and pins, or more generally dealing with the low level details of the MCU. Although this feature greatly simplifies the programmer's work, it also introduces some overhead and prevents a fine grained optimisation of the code[2]. On the other hand, the Contiki OS introduces the concept of interoperability of the code, which can be adapted to different MCUs without altering the programmer's code.

1) RTimers: Among the various tools that Contiki provides to the programmer, one of the most important ones are the RTimers which allow the execution of real-time tasks. This type of timers offers an higher clock resolution than the other available timers, allowing more precision in the scheduling of time events. Moreover, RTimers are pre-emptive, thus allowing the execution of tasks in precise time instances, powered by the interrupts. The real-time execution of tasks requires the programmer to structure the code in such a way that pre-emption does not interfere with the correct flow of the application logic. Interrupts might, for instance, be disabled throughout the code to guarantee the atomicity of a function, although potentially defeating the purpose of the RTimer.

RTimers, however, do not allow the possibility to schedule multiple events: maximum one timer at a time might be set so that, if a series of scheduled events is required, it must be programmed by scheduling each consecutive event whenever the previous event is triggered.

C. Application Scenario

The two ND protocols are to be used in a Low Powered Wireless Networks (LPWNs) with a variable number of nodes organised in an arbitrary yet static topology. The topology and the range of reception of these wireless nodes is such that all of the nodes are to be considered in range of every other node, thus creating a very dense communication channel. This scenario is particularly exposed to collisions which are bound to grow with the density of the network and might be influenced by the different implementation choices for the two discovery primitives. Albeit the application scenario is formed by static nodes, the two protocols support continuous neighbour discovery by using the concept of epochs: at the end of a certain epoch, the node will start the discovery again with a new epoch.

III. System Design

A. Implementation Style

The Contiki OS offers different programming style that can be used to implement the primitives at hand. Among them we can find processes, protothreads and callbacks. The implementation presented in this paper is based entirely on callbacks which are managed via the use of RTimers, hereafter referred to as simply timers. As mentioned in Section II-B1, RTimers offer a pre-emptive solution that allow the execution of real-time tasks with a good time-precision. If, for instance, a callback is scheduled at a certain time *x* in the future, we can be sure that the associated function will be executed at that time with good precision[3]. This allows for the ND primitives to be completely implemented through the use of callbacks to which the different node functions are associated. Specifically, different functions are created for the different time events, such as the end and beginning of each slot. The functions are then either called directly by other time-sequential events - for instance the beginning of a slot will be called directly, with no delay, by the end of the previous one - or are associated to a timer with a callback to be triggered at a specific point in time - for instance the beginning of a slot will schedule it's end. Due to the impossibility of scheduling multiple timers, as discussed in Section II-B1, the scheduling of events whose occurrence timing is well-known but preceded by other events is taken care by keeping some counters or track of the passed time. For instance, the end of an epoch is triggered whenever a certain number of slots has occurred, even if its duration is well known. Moreover, the end of a slot in some TX or RX modes is scheduled keeping track of the difference between the passed time and the time the slot should end.

The different functions used to implement the primitives are designed in such a way that the different timers are scheduled as early as possible in the function code flow. Since the scheduling of the timers is mainly based on the tracking of the passed time, the more time passed we don't account for and the less precision the callbacks will have. If, for instance, a certain function *B* should be called after x ms from another function *A*, if we call *B* at the end of *A* then *B* will be executed exactly $x + z$ ms after *A*, where *z* is the amount of time required by the MCU for the execution of the code of *A*. Calling *B* in the first line of *A* will instead reduce drastically the amount of time *z*, thus achieving more precision. Of course it's not always possible to schedule the call of *B* at the beginning of *A*, but the design should try to move the scheduling as early as possible[4].

B. Parameters

The guidelines presented in Section II-A do not specify any detail about the different timing and duration of the various events; the only parameter which is given is the duration of an epoch which is set to 1 s. The epoch should be seen as the end of a discovery session, after which the performance evaluation for that epoch can be performed. At the beginning of each new epoch all of the gathered data is to be reset. The lack of details in the guidelines for the design of both primitives requires us to choose several crucial parameters that heavily impact the performance of the discovery. Performances, in a neighbour discovery protocol, can be defined in different ways but, ultimately, they are driven by some requirements. In a neighbour discovery context we can measure different factors that indicate how the protocol at hand performs. Three of the most common ones are the following:

[2]Contiki, as most OS is still optimised, thus grating acceptable performance.

[3]Assuming no manual disabling of interrupts was performed.

[4]An analysis of the assembly code generated by the assembler and the CPU cycle required for their execution could be used to take into account the execution time of the function, thus incrementing the overall precision.

- *Discovery rate*: the number of nodes, out of the ones in range, that are discovered in an epoch. In our application scenario it is the percentage of discovered nodes.
- *Discovery latency*: the latency with which new neighbours are discovered. In a dynamic topology it is the time passed between the instant the node enters in range and the instant the node is detected.
- *Power consumption*: the amount of power used by the nodes, also referred to as the *duty cycle*.

The application scenario at hand, briefly discussed in Section II-C, makes the overall discovery latency of the protocols a rather meaningless measure. Since all of the nodes are to be considered static, the latency with which a node is discovered by its neighbour does not impact sensibly the final result, as long as the neighbour is discovered. If, instead, the nodes were mobile then the latency would play a crucial role in the ability of a node to be discovered. The other two measurements, instead, are considerably relevant to the performance evaluation of the scenario: discovery rate gives a measure of how reliable our neighbour discovery is and power consumption is a measure that, as in most of the LPWN scenarios, gives an insight on the potential lifespan of the network in case of real-world deployment.

As it is typical of this type of scenarios, these two different metrics are quite in contrast with each other, presenting a trade-off to the protocol designer who is forced to make some decision based on whether power consumption is more important than discovery rate. Generally speaking, the design choices usually follow some clear requirements on at least one of the measurements, trying to respect a certain threshold while optimising at best the other measurement. For instance, given the requirement of a 50% discovery rate, the protocol is tweaked to reach such threshold while minimising the power consumption as much as possible. For this reason, it is not uncommon to resort to a complex Requirements Analysis to yield the best parameters for the protocol. An example of such approach can be found in the work of Julien et. al [1] which proposes BLEnd, a continuous neighbour discovery protocol that relies on an optimiser module capable of analysing some given requirements to output the best parameters for the protocol itself.

The lack of specific requirements or goals makes the design of the two neighbour discovery primitives quite arbitrary. The different parameters play indeed a role in the aforementioned trade-off and should be adjusted accordingly. For this reason, the work proposed in this paper makes an important assumption: *the discovery rate and the power consumption of the protocol have equal importance*. All of the following choices should therefore be reconsidered in case this assumption was to be considered not valid.

Another important factor that is crucial in the implementation of any protocol is the technology for which it is designed. An example can be made of BLEnd, which was designed to be used with Bluethooth low energy. In the case of the two primitives discussed in this paper, no technical details are given about the hardware. As a consequence, the only physical requirements and information will be given by the simulation scenario, whose value and characteristics will determine some of the following choices.

1) Slot duration: The *slot duration* is a parameter that has an important role in the two neighbour discovery protocols at hand. Such parameter influences both the number of transmissions and the total time spent in reception by the single nodes. Since the epoch duration is a static parameter, the best way to determine the slot duration is to derive it from a certain *number of slots* in a certain epoch. A higher number of slots equals to shorter slots and vice-versa.

In case of the Burst primitive, a lower number of slots would result in sparser receptions but in longer sequences of beacons, which have to be transmitted during the whole duration of the TX slot. From the power consumption point of view, the effect of this parameter depends on the comparison between the cost of TX and RX. It is to be expected that longer slots would be beneficial to the power consumption in case the cost of reception, thus the power spent by the radio for listening on the channel, exceeds the cost of transmission. For what concerns the discovery ratio, the impact on the slot duration on such metric is hard to determine without extensive simulation. Nonetheless, a longer slot implies a much higher chance of collisions since it is much more likely that two neighbours' TX slots overlap. This is true especially in the given application scenario, having a topology where all of the nodes are in range of every other node.

In case of the Scatter primitive, a lower number of slots would result in sparser transmissions and longer receptions, thus favouring scenarios in which RX is less expensive than TX. However, the reception slots are user much more intensively than the transmission slots, thus heavily impacting the power consumption on longer slots.

As for the Burst primitive, also in the Scatter version no simple analysis of the impact of the slot duration on the discovery ratio can be made. However, the collision issue raised in the Burst mode does not appear in Scatter since the reception is a passive action that does not provoke any collision if overlapping with other nodes'.

2) Beacon frequency: The guidelines of the Burst primitive offer quite some room when dealing with the transmission slots. Although they could as well suggest an uncontrolled beacon transmission, with nodes transmitting the beacons as frequently as possible, this solution would be quite counter-productive. An excessively high frequency would in fact result in a considerably high number of collisions, especially considering the application scenario at hand with all of the nodes in range of each others. Additionally, the higher the frequency and the higher the power consumption required during the entire transmission slot. On the other hand, a too infrequent transmission would impact on the RX slot as discussed in the next section.

Analysing the single beacon transmission, the length of such event depends strictly on the radio technology used and on the content of the transmitted data. Given the content set as the node identifier, the length measured during the simulation of a

single beacon transmission is approximately $258\,\mu s$. However, the length of the radio activity, which is measured from the moment the radio is on to the moment it is turned off, is approximately $576\,\mu s$. Considering the Cooja primitive that is in charge of transmitting data, it is reasonable to assume that it would not be possible to achieve more than 1 transmission every $576\,\mu s$, which would correspond to the the maximum frequency of the beacons. This value is only theoretical for it doesn't take into consideration the overhead of the instructions calling the primitives or any other code in charge of managing such transmission frequency.

Even considering a rather low beacon frequency to avoid collisions, in the event that one such collision occur then many more are expected to happen. A collision might indeed happen if two neighbours transmission slots overlap and, more specifically, two different beacons overlap as well. However, since the interval between two beacons is fixed across nodes, if two beacons overlap then it is most likely[5] that also the subsequent beacons will collide as well. To account for this problem, a collision avoidance (CA) technique can be engineered to reduce such possibility.

3) Reception time: Reception time is a parameter that only concerns the Burst primitive since in the Scatter ND protocol nodes listen throughout the whole duration of the slot. Such parameter determines the amount of time the nodes will be listening on the channel for incoming beacons, at the beginning of each reception slot. The reception time yields a tangible effect on the power consumption of the reception slot while also impacting the expected performance of the protocol: the longer the radio is used for reception and the more power will be depleted, while also being able to receive more beacons. From the point of view of the primitive correctness, the reception time must be at least as long as the maximum interval between two beacons. Only that way the neighbours are guaranteed to be discovered, assuming optimal conditions with no collision. Given the periodicity of the reception slots of a certain node, the neighbours' transmission slots are bound to overlap with them; only if during such overlap a single beacon is ensured to be listened then the discovery is guaranteed. As a consequence, the most logical choice for the reception time in Burst is indeed the maximum interval between two beacons plus the beacon transmission time - to ensure that it is correctly received. However, a longer reception time might smooth the negative impact of collisions on the protocol, allowing a node to receive more than one beacon in case the first one was lost. It's worth noting that, due to the close relation between the beacon frequency and the reception time, a longer beacon frequency will lead to a greater power consumption for the reception phase, thus introducing another trade-off.

C. Collision Avoidance

The collision avoidance technique adopted by the work proposed by this paper is derived and similar to the one designed by the work of Christine Juliene et al. [1] in BLEnd, a continuous neighbour discovery protocol with a lot of similarities with the Burst primitive. Among the various design choices, BLEnd uses a periodic beacon transmission that takes into account the aforementioned collision problem and addresses it by introducing a random time slack before the transmission of any beacon. Such slack, whose entity is smaller than the frequency interval, allows nodes to unsynchronise their beacons transmission during two overlapping transmission slots. The interval between beacons will indeed be afflicted by such slack, thus varying, thanks to its randomicity, the intervals across different nodes.

An alternative technique that could address the collision problem is the introduction of static but random beacon intervals. Rather than randomising each interval during a single transmission slot, the nodes might initialise the primitive by setting a random yet persistent interval, thus avoiding the synchronisation problem. However, in case of a consistent number of nodes, the amount of variance required in the intervals would negatively impact the power consumption by influencing the reception time.

No collision avoidance technique is engineered for the Scatter primitive since the number of collisions is to be expected far inferior. Scatter is a reception-focused protocol that, by sending very few beacons, is not particularly affected by the collision problem.

D. Scatter Revision

The proposed implementation of the Scatter ND differs from the guidelines presented in section II-A and depicted in Figure 1. If the original guidelines suggested the broadcast of the beacon at the beginning of the TX slots, the proposed implementation performs the transmission approximately in the middle of the slot. This change does not compromise the effectiveness, nor the nature, of the primitive but it solves a bug during its simulation. Figure 2 shows how the simulation encounters a bug for node 9. Specifically, it can be seen how the latter remains stuck in reception even after the RX slot has terminated. This bug, which afflicts several nodes during the simulation making them crash, can be traced back to the immediate transmission of the beacons right after the reception of something, thus requiring a delay between end of reception and the beginning of a transmission.

E. Optimiser

In order to account for the lack of stricter requirements, this paper also proposes a basic optimiser module that, leveraging extensive simulation, can determine an *optimal* set of parameters to be used with the two discovery primitives. Table I shows the possible inputs that can be given to the optimiser. The first three inputs are mandatory parameters and represent the weights to be given to the different performance indicators: average discovery rate, standard deviation of discovery rate and average of power consumption (duty cycle). Additionally, the *preferred number of nodes* can be used to indicate a preference in the number of nodes for which the primitive should be optimised for. The protocols are tested on different topologies with different numbers of nodes, thus potentially

[5] A clock drift might prevent this event from happening.

Parameter	Type	Symbol
Discovery Rate (Avg) - Weight	float [0,inf)	μ
Discovery Rate (Std Dev) - Weight	float [0,inf)	θ
Power consumption (Avg) - Weight	float [0,inf)	γ
Preferred Number of nodes	int [2,5,10,20,50]	$\#$
Preferred Number of nodes - Weight	float (1,inf)	d

Table I: List of input parameters

Burst output	
Output	**Type**
Slots per epoch	int[4,8,12,16]
Beacon frequency	ms [3,5,10,15,20]

Table II: List of optimiser outputs for the Burst ND

yielding different performance. Using the preferred number of nodes parameter and giving a certain weight to such choice will allow the optimiser to give more importance to the performance of the primitives with a specific number of nodes. This allows a further optimisation in case the magnitude of the topology is well known. The output of the optimiser can be seen in table II and III. For the Burst ND primitive it consists in two of the parameters discussed in section III-B, slots per epoch (slot duration) and beacon frequency, while Scatter only support slots per epoch. Such outputs are based on the estimated performance ρ which is computed based on the total average μ of the Discovery Rate, the total standard deviation σ of the Discovery Rate and the total average γ of the Power Consumption, as follows:

$$\mu = \frac{1}{6} \sum_{n \in Sim} e * n\_Avg\_DiscoveryRate \tag{1}$$

$$\sigma = \frac{1}{6} \sum_{n \in Sim} e * n\_StdDev\_DiscoveryRate \tag{2}$$

$$\gamma = \frac{1}{6} \sum_{n \in Sim} e * n\_Avg\_PowerConsumption \tag{3}$$

$$Sim = [2, 5, 10, 20, 30, 50] \tag{4}$$

$$e = \begin{cases} d, & \text{if } n = \#. \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

$$\rho = \frac{a * \mu}{b * \theta + c * \gamma} \tag{6}$$

The performance ρ are thus directly dependent on the input of the optimiser. The optimiser is, however, a simple python script that can be extended and modified to take into consideration different criterias or to compute a different performance measurements. ρ is an arbitrary formula that scales with the input parameters in a logical yet arbitrary way. The formula, as it can be seen, is directly proportional to μ and indirectly proportional to both θ and γ. The current version of the optimiser is limited in both the input and the output; future work could extend both the type and number of input and output.

Scatter Output	
Output	**Type**
Slots per epoch	int[4,8,12,16,20,24,28]

Table III: List of optimiser outputs for the Scatter ND

IV. Evaluation

In order to evaluate the work proposed by this paper, the two neighbour discovery protocols are experimented over the application scenario discussed in Section II-C. The scenario is virtually simulated with the help of Cooja, a java network simulator designed for Contiki OS. The goal of this evaluation is to compare the performance of the two protocols using the parameters from *optimiser*. Furthermore, we give an overview on how the various parameters affects the various performance indicator mentioned in the previous sections. Finally, an evaluation on the collision avoidance technique is proposed.

Each ND primitive set up has been tested with 6 different topologies, having [2,5,10,20,30,50] nodes to measure the effectiveness of the protocol with various network sizes. Furthermore, each single topology has been tested 5 different times, each one with a different random seed and node startup time. Each topology has been tested for $\sim$170 epochs, discarding the results of the first 5 and last 5 to discard potential incomplete data.

A. Burst Analysis

ND Burst is the most complex protocol out of the two proposed. The TX slots' complexity increases due to the need of handling the beacons' transmission. Figure 3 shows how the various performance metrics, explored in section III-A, vary depending on the various parameters of the ND. Precisely, it can be seen how the discovery rate improves, in both average and standard deviation, with shorter slots (more slots per epoch). On the contrary, the graph shows how the power consumption increases alongside the discovery rate, thus remarking the trade-off between the power efficiency and discovery ratio. We can justify the duty cycle increase with the fact that more slots require more listening activities. We can therefore conclude that increasing the number of slots per epoch will be beneficial if discovery rate is the main goal.

The graph also shows how decreasing the frequency of beacons, thus increasing the interval between the beacons, results in better discovery ratios. This can be justified by the reduction in collisions due to the fewer transmissions. Since, however, sparser transmission require also longer receptions, the lower frequency sensibly increase the power consumption of nodes - especially when number of slots per epoch is higher. Figure 3 shows indeed how the most power expensive version of the Burst ND is the one using the shortest slots and the lowest frequencies which implies the highest number of reception slots with the most time spend in reception.

1) Collision avoidance: The collision avoidance technique implemented in the Burst ND primitive has the effect of reducing the number of collision, thus improving the discovery rate of the protocol. Figure 4 and 5 show a direct comparison

between the discovery ratio and the duty cycle, respectively, of the Burst protocol with and without the proposed Collision Avoidance (CA) technique. As expected, Figure 4 shows that CA improves the discovery ratio in almost every scenario, although slightly. Figure 5 shows, instead, how the duty cycle sensibly increase with CA. This is due to the longer reception needed to account for the increased maximum interval between two beacons (due to the added random slack). Using the Optimiser script on Burst will thus yields unexpected results. Figure 6 shows the computed performance with the following input parameters: $\mu = 2$, $\theta = 1$, $\gamma = 2$ and without preferred nodes, thus giving equal importance to discovery rate and power consumption. The graph shows how the performance are indeed lower with CA since the increase in duty cycle is higher than the increase in discovery ratio. We can therefore state that CA is to be adopted exclusively if discovery ratio is the main goal, and be forsaken in more power-conservative scenarios.

B. Scatter Analysis

The Scatter protocol is a simpler ND primitive that, rather than focusing on transmission, it emphasize the reception, using a whole slot to listen for incoming beacons. Contrarily to Burst, the transmission of the beacons happens once in every TX slot, thus simplifying the protocol. As a consequence, the Optimiser only returns the *optimal* number of slots per epoch. Figure 7 shows the various performance metrics for the Scatter protocol; it can be seen how the discovery rate slightly decreases with a higher number of slots per epoch. On the other hand, the graph clearly show how the increase in the slots reduces the average duty cycle, which is to be expected given the big difference between the TX and RX slots in terms of power consumption: shorter slots are paid only with single beacon transmissions. Surprisingly, the power consumption stabilises with increasing slots. Looking at Figure 8, which shows the computed performance using the optimiser, the optimal number of slots can be identified.

C. Burst and Scatter comparison

Having analysed the various primitives singularly, a general comparison can be made to understand which version can be considered the best. Figure 9 shows how Burst is generally a better primitive which can be used to obtain both better discovery rates and better power consumption. If, for instance, the focus is on discovery rate than *br_ca_16_20*[6] protocol is the best one. If, on the other hand, power consumption is the focus then Burst offers several solutions yielding better discovery rates while retaining lower power consumption than any of the Scatter options. Looking at the computed performance graph generated by the proposed optimiser in Figure 10, it can be seen how the previous conclusion is supported: the Scatter performances are generally worse.

[6]Burst CA with 16 slots per epoch and beacon interval of 20 ms.

V. Conclusion

Neighbour Discovery in IoT is a task that can be driven by different requirements which, ultimately, might define the protocol to be used. This paper proposes and analyses several options for two ND protocols: Scatter and Burst. It can be seen how Burst is, overall, a better protocol that can obtain great versatility by tweaking its parameters. Parameters have indeed a huge impact on the performance of the various ND primitives, and should therefore be carefully defined.

References

[1] Christine Julien, Chenguang Liu, Amy L Murphy, and Gian Pietro Picco. Blend: practical continuous neighbor discovery for bluetooth low energy. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 105–116. IEEE, 2017.

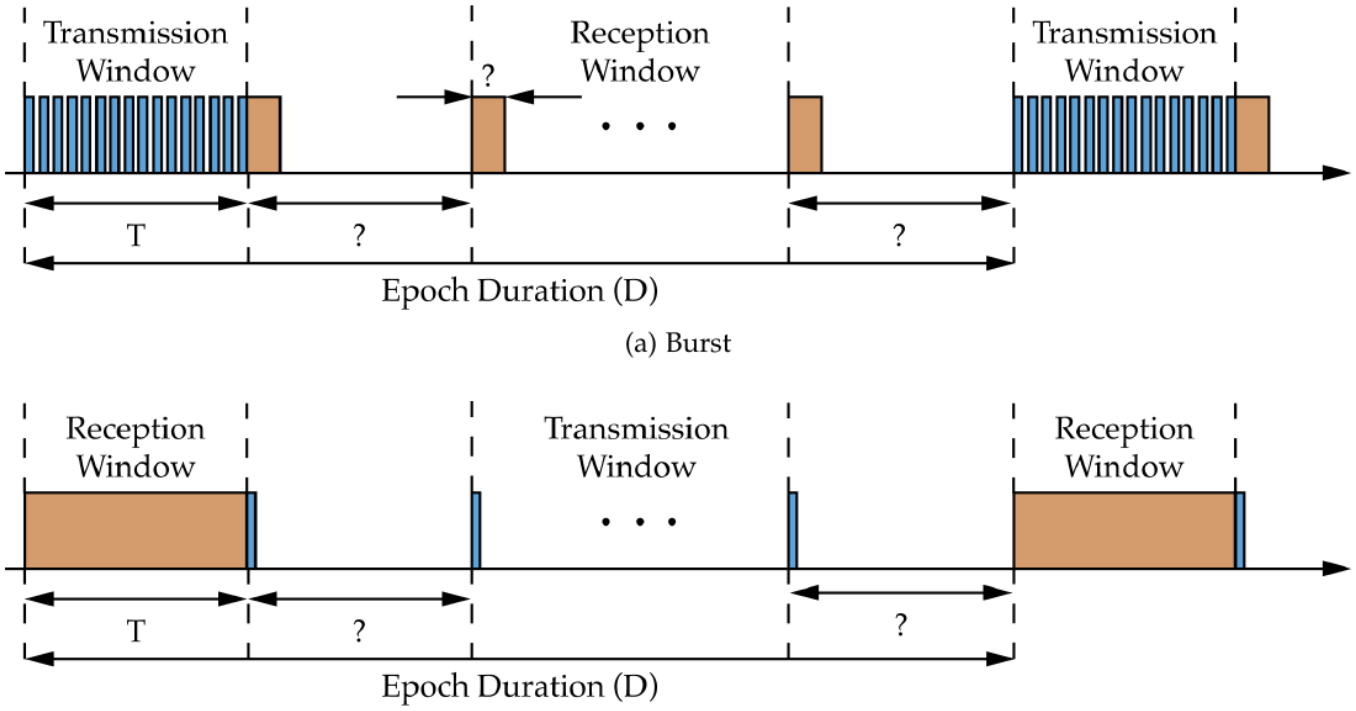

Figure 1: The two neighbour discovery primitives guidelines

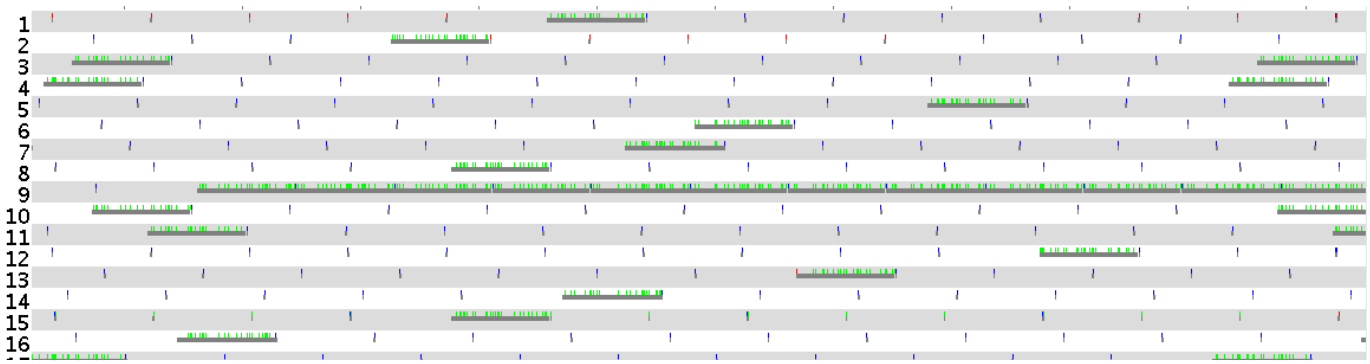

Figure 2: Graphical representation of the Contiki simulation. A bug afflicting the protocol can be seen in the anomalous behaviour of the node 9. Short blue lines indicate the transmission of a beacon while long gray lines indicate the listening activity of the nodes.

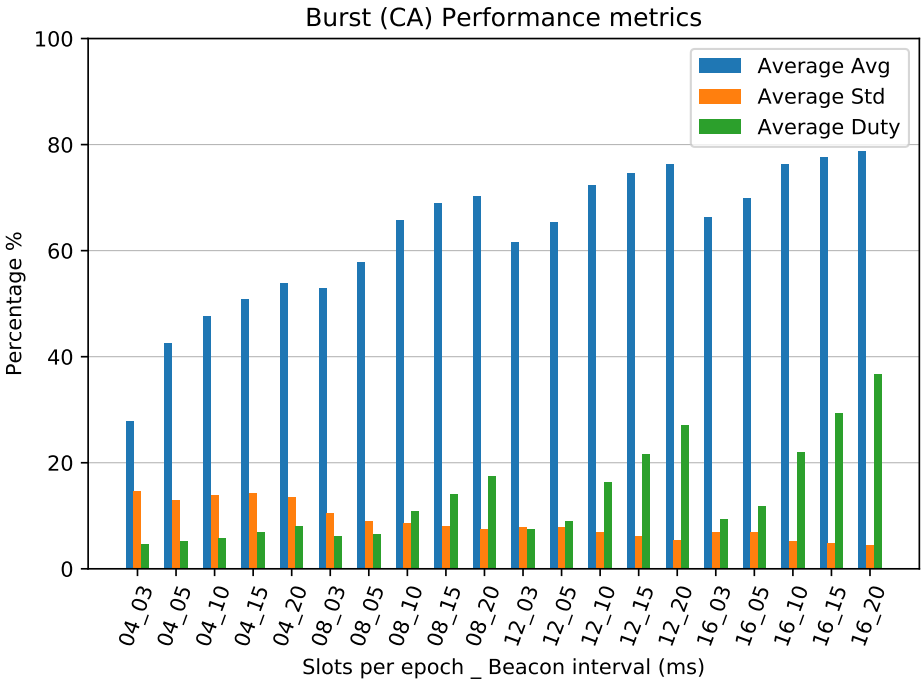

Figure 3: Performance metrics for the Burst protocol with collision avoidance.

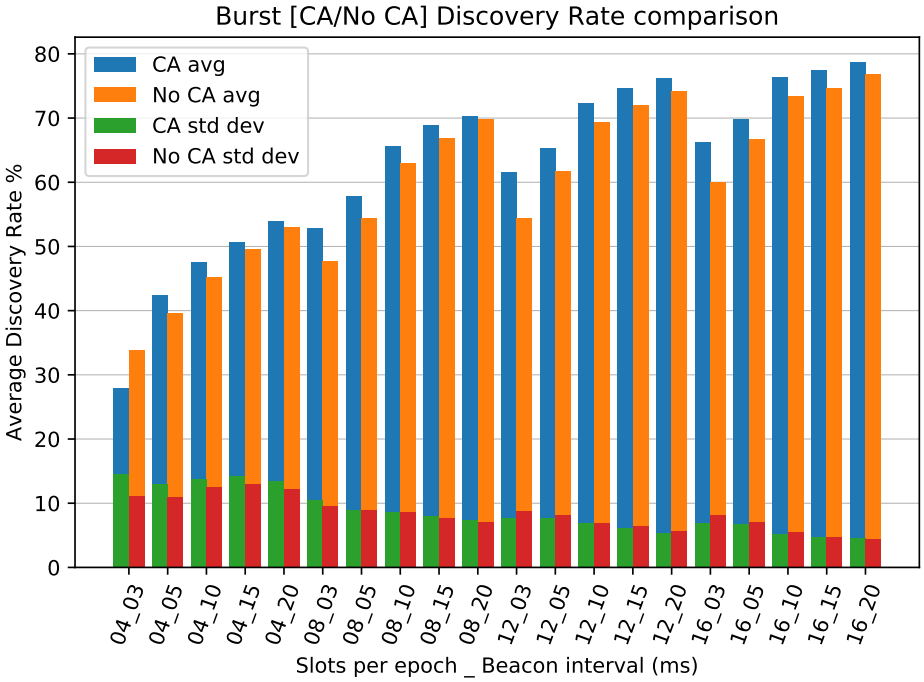

Figure 4: Discovery rate comparison between the Burst protocol with and without collision avoidance.

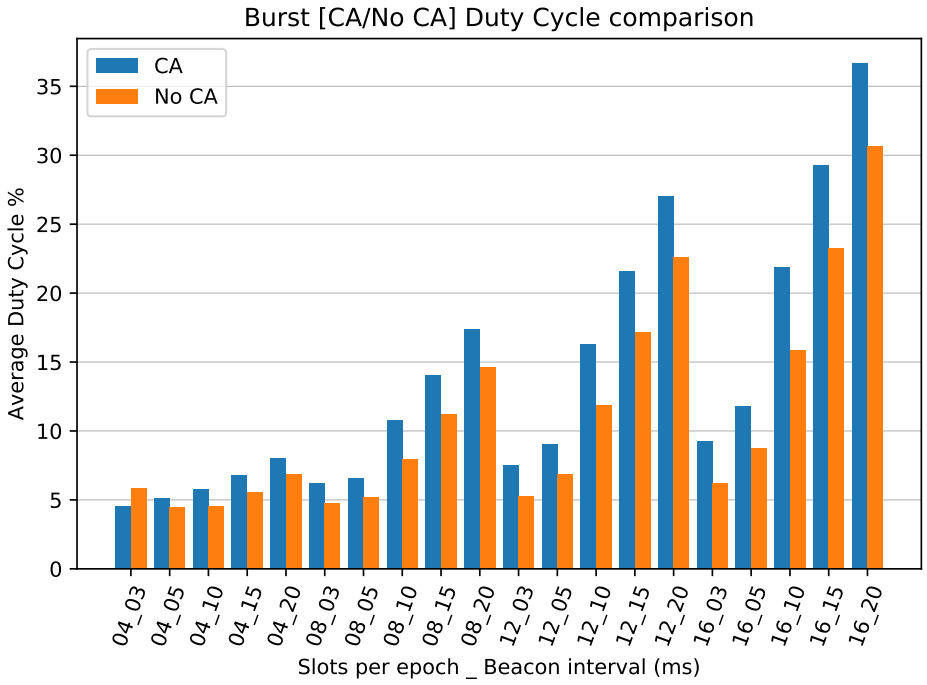

Figure 5: Duty cycle comparison between the Burst protocol with and without collision avoidance.

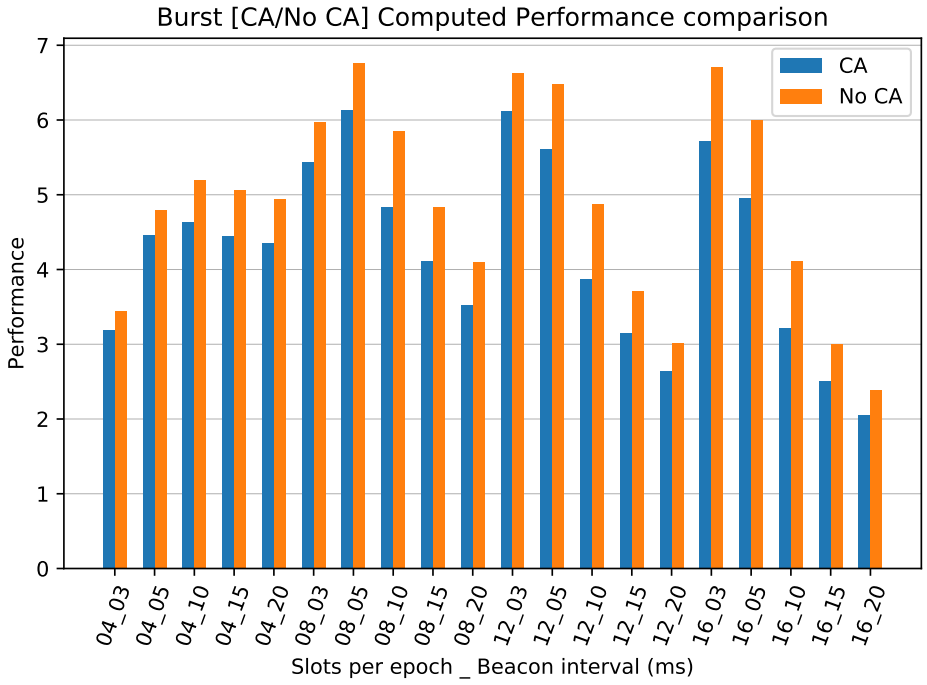

Figure 6: Computed performance for the Burst protocol with and without collision avoidance. Optimiser inputs: $\mu = 2$, $\theta = 1$, $\gamma = 2$ and without preferred nodes.

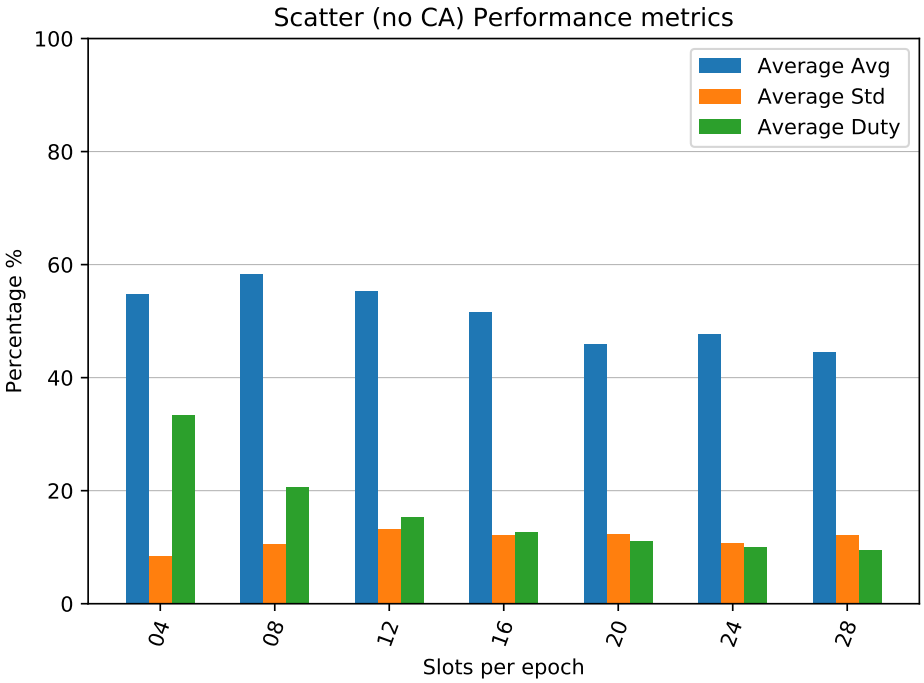

Figure 7: Performance metrics for the Scatter primitive.

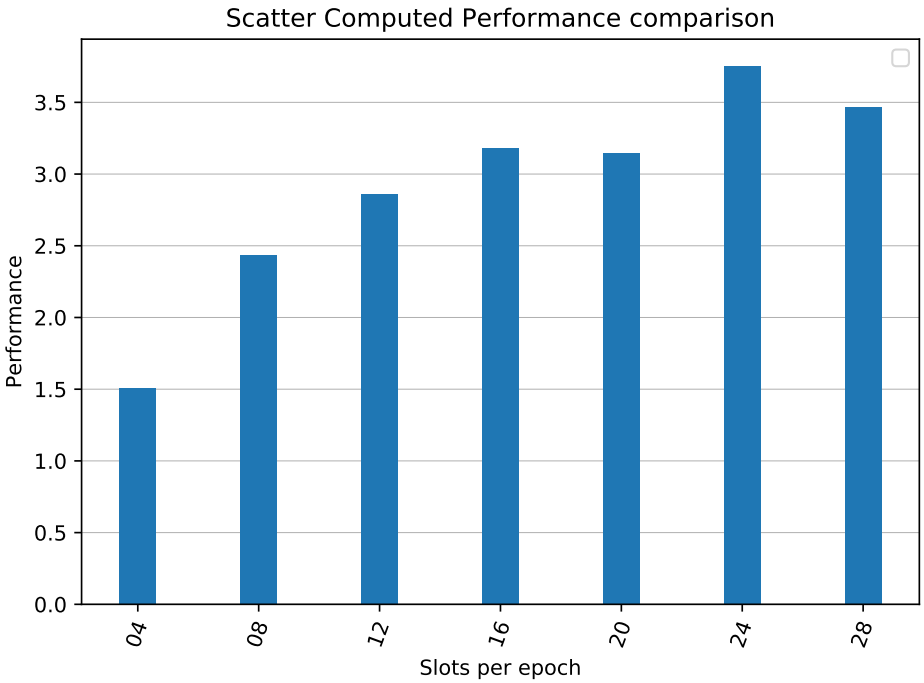

Figure 8: Computed performance for the Scatter protocol. Optimiser inputs: $\mu = 2$, $\theta = 1$, $\gamma = 2$ and without preferred nodes.

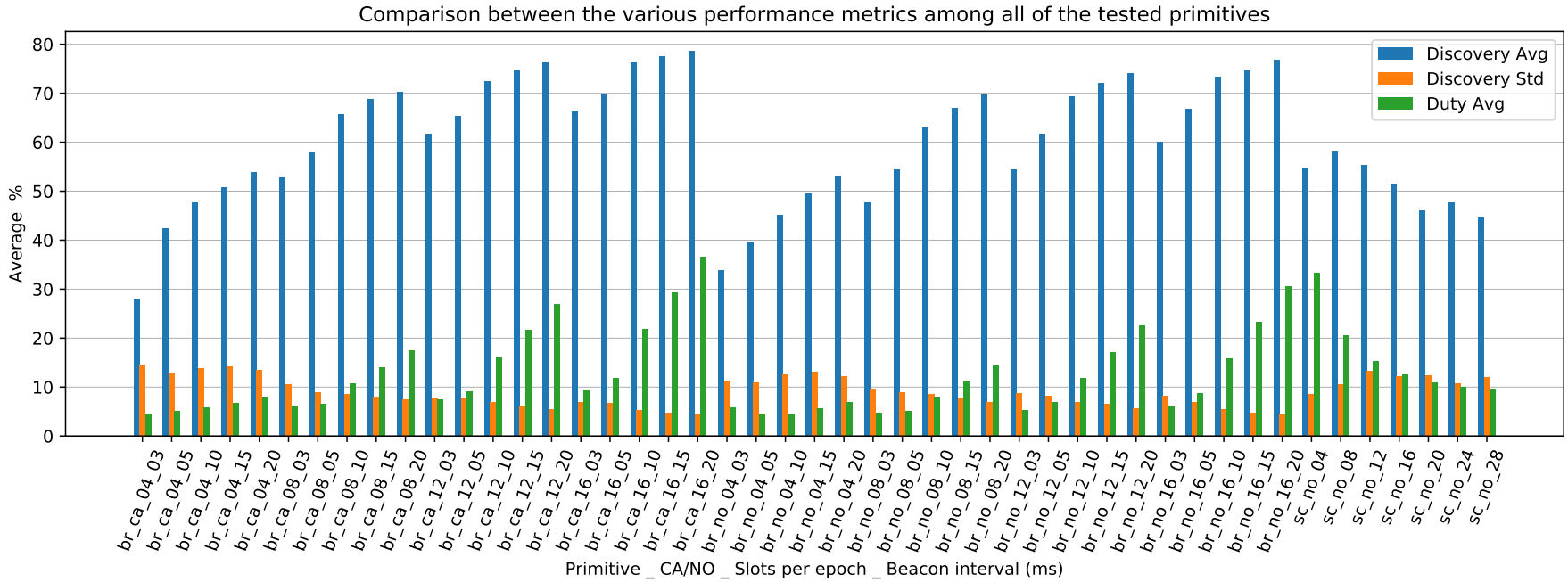

Figure 9: Performance metrics for all primitives.

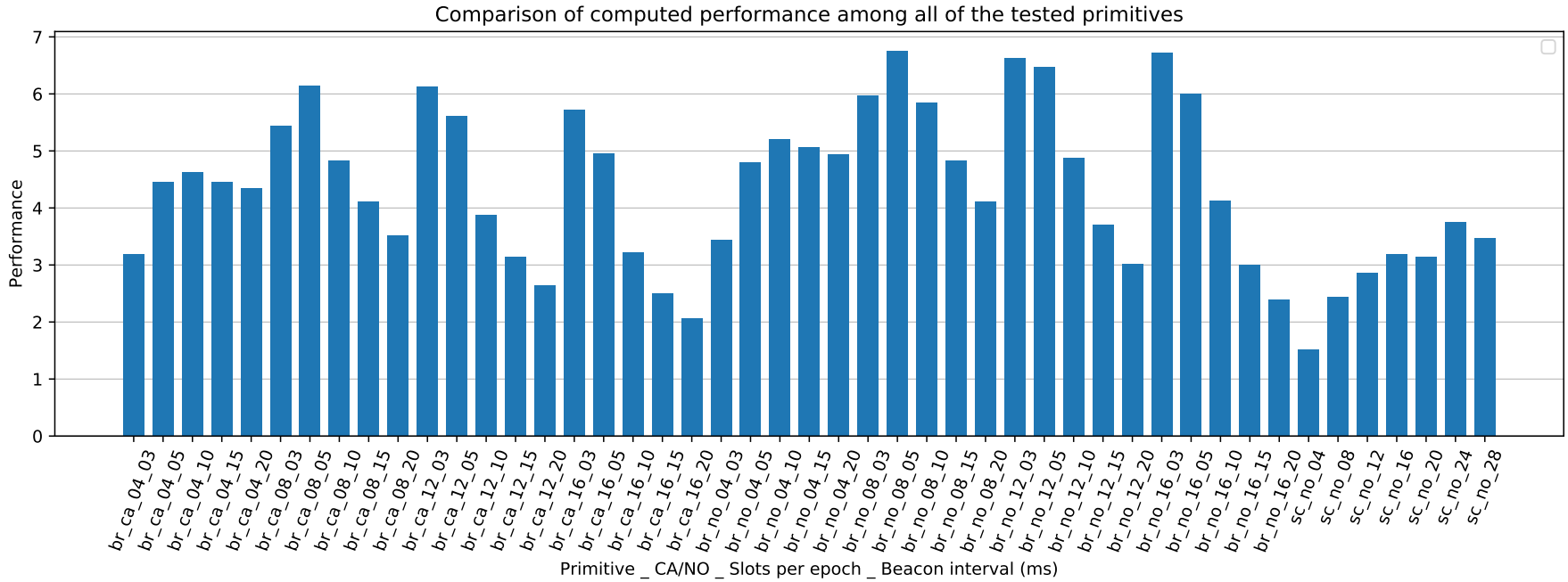

Figure 10: Computed performance for all the primitives. Optimiser inputs: $\mu = 2$, $\theta = 1$, $\gamma = 2$ and without preferred nodes.