Implementing a Graph Neuron Array for Pattern Recognition Within Unstructured Wireless Sensor Networks

M. Baqer, A.I. Khan, and Z.A. Baig

Monash University, Australia

{mohamed.baqer, asad.khan, zubair.baig}@infotech.monash.edu.au

Abstract. Graph Neuron (GN) is a network-centric algorithm which envisages a stable and structured network of tiny devices as the platform for parallel distributed pattern recognition. However, the unstructured and often dynamic topology of a wireless sensor network (WSN) does not allow deployment of such applications. In this paper, using GN as a test-bed application, we show that a simple virtual topology overlay would enable distributed applications requiring stable structured networks to be deployed over dynamic unstructured networks without any alteration.

1 Introduction

WSNs are deployed in critical environments for event sensing and reporting purposes. Due to the computation constraints of sensors, most contemporary WSNs are designed to react to immediate real-time events by relying on a high-performance base station or a server for centralised event processing. Such WSNs have been deployed in areas such as Health Monitoring, Traffic Control, and Industrial Sensing. More recent applications are in Infrastructure Security and other detection/tracking areas [1]. The Graph Neuron (GN) proposed in [2][3], as a real-time parallel pattern recognition algorithm for tiny devices uses a simple distributed algorithm to intelligently recognise patterns in the monitored environment. We assert that for real-time systems, particularly those which make use of in-network processing, the underlying network must have a structured topology with established mechanisms for ensuring bounded time delays for all of its operations. Chord [4] as distributed lookup scheme can be conjectured to provide a structure to a WSN for seamlessly handling the dynamic aspects of the WSN and for providing the necessary framework for supporting distributed applications such as the GN. In this paper we would investigate a Chord based scheme for establishing a self-organising and self-adapting structure for the WSN. Additionally, we would use the Chord overlay to meet the application's scalability requirements.

2 Background

Graph Neuron (GN) is a pattern recognition algorithm which can form an associative memory network by interconnecting tiny devices in a graph-like structure

called the GN array; shown in Figure 1. The information presented to each of the tiny devices (GN nodes) is in the form of a value, position pair. Each of these pairs represents a data point in the reference pattern space. The GN array hence forth converts the spatial/temporal patterns into a graph-like representation and then compares the edges of the graphs for memorisation and recall operations. The advantage of having a graph-like representation is that it provides a mechanism for placing the spatial/temporal information in a context. Hence not only can we compare the individual data points but we may also compare the order in which these occur. The GN algorithm utilises the real parallelism present within the network to speed up the comparisons. The number of comparisons required for matching a stored pattern with an incoming sequence for an n vertex graph could be of the order of $O(2^{n/3})$ [2][5], however the GN circumvents this very high computational cost through fine-grained parallelism [5].

The data representation for a GN may be summarised as follows: input pairs $p(\text{value}, \text{position})$ are mapped on to a virtual array of processors by using the adjacency characteristic of the input e.g. alphabets and numbers would have their inherent adjacency characteristics. Similarly images would have the frequency bands, intensity, and spatial coordinates as the adjacency characteristics per pixel etc. For a reference pattern domain R , the GN array represents all possible combinations of P in R . Each GN node is initialised with a distinct pair p from the input domain R . Also, each GN node executes an instance of the full GN code. A GN instance keeps a record of the number of times it encounters a matching input pair within its bias array. Each row of the bias array comprises a list of the adjacent GNs relating to a matched input pair. The bias row counter is incremented for every new pair encountered by the instance. A new pair is defined as the one which has a different set of adjacent GN nodes to the existing rows of the bias. The GN algorithm may be categorised into the following three stages:

Fig. 1. A simplified representation of the GN array's store (memorise) and recall operations

Fig. 2. Structuring WSN with a Chord overlay

1. Mapping of the input pattern to the appropriate GN nodes.
2. Marking the end of the incoming pattern.
3. Bias update and lookup operations for pattern recall or memorisation.

Most of the steps within these stages may be executed in parallel. Stage 3 operation will result in one of the two possible outputs, namely memorisation or recall. A memorisation process is initiated if an input pattern is not matched with stored patterns. On the other hand, a recall is the result of a match being found amongst the stored patterns [2][5]. The GN algorithm requires that the nodes are setup as an ordered array where each node is aware of its adjacent nodes. In our proposed scheme, the Chord lookup protocol will be used to structure the WSN, as can be seen in Figure 2.

3 A Structured In-Network Processing Scheme for WSNs

Implementation of the GN code on an unstructured and dynamic WSN is not feasible since the GN algorithm assumes that the network be deterministic to support its real-time processing requirements. Also, the GN algorithm requires a structured network for gathering node adjacency information [2]. Our proposed scheme ensures that a suitable structure is appropriated to the WSN (using the Chord protocol) to meet these requirements.

Traditional WSN applications may constantly get interrupted owing to nodes' join, leave, and failure. The Chord protocol in our proposal defines a distributed lookup mechanism in which the position fields of the GN pairs and WSN nodes are used to provide a robust self-organising overlay (*virtual topology*) capable of handling the WSN dynamics. In addition, the overlay provides a reliable mechanism [4] for locating the WSN node designated for hosting a particular GN pair. The overlay uses an in-network distributed hash table (DHT)-based mechanism to avoid relying on a centralised entity for content discovery and structure maintenance.

In our scheme, every WSN node is assigned a key called the *WSN_key* generated from its position in the network in our scheme. WSN node's location discovery is beyond the scope of this paper, however, [6] [7] propose effective distributed techniques for WSN node position discovery. An input pattern P of size N arriving at a node say X of the GN array may be visualised as a collection of $N \times p(\text{value, position})$ pairs. On receiving the input pattern P , node X decomposes P into N input pairs. A new *GN_key* is generated for each input pair p using its position field's value. Node X next looks up its local memory for any existing *GN_keys* generated from previous inputs that can be reused for the current input pattern. All additional *GN_keys*, as required, are generated. Subsequently, all generated *GN_keys* are mapped by node X to corresponding WSN nodes based on the individual *WSN_key* ranges of each WSN node. Both the *GN_keys* as well as the *WSN_keys* are generated using one of two hash functions supported by the Chord protocol, namely, SHA-1 or MD-5. All WSN nodes including node X which will be involved in the pattern recognition operation would

forward the incoming GN pairs from node X to their respective GN instances in the array. Finally, the GN instances running on the participating WSN nodes will collaboratively complete the pattern recognition process.

The structured WSN topology in our scheme ensures that all search operations for finding adjacent GN pairs are completed in $O(\log n)$ time [4]. The network layer handles the data communication and routing aspects. The separation between the network layer and the Chord layer allows for flexibility and efficiency in selecting the optimal routes based on routing techniques described in [8]. We assume in our design that all active WSN nodes, regardless of their contribution to the pattern recognition process, participate in routing.

All WSN nodes participating in the pattern recognition process are arranged, by the Chord protocol, in a circular structured called the identifier circle - based on their respective *WSN_keys*. The identifier circle is divided into value ranges based on *WSN_key* values. Each WSN node in the identifier circle is responsible for a range of *WSN_keys* starting from its own *WSN_key*. GN pairs are distributed among the WSN nodes based on the individual *GN_keys* of the pairs. The WSN node responsible for hosting a GN pair is located on the identifier circle by checking for the existence of the *GN_key* (for the GN pair) within the *WSN_key* range of the WSN node. These steps are shown as a pseudo code in Figure 3.

The total number of available nodes in the WSN must be equal to or greater than the size of the input pattern to ensure that every GN pair is mapped to its designated WSN node. If the required number of WSN nodes is less than the input pattern size, then the existing WSN nodes cater for missing nodes by hosting multiple GN pairs derived from the input pattern. The consistent hashing mechanism of Chord would to some extent balance the load resulting from the handling of multiple GN pairs per node in this case.

```

1  for i = 1 to N
2    for j = 1 to N
3      if  $GN\_Key_i \in \text{range}(WSN\_Key_j)$ 
4        GN pairi is mapped to WSN nodej
5  for available WSN nodes  $\geq N$ , each GN pair will be mapped to a
6  unique WSN node.
7  Where N is the size of the arriving pattern P
8   $\text{range}(WSN\_Key_j) = (\text{predecessor's } WSN\_key, WSN\_key_j]$ 

```

Fig. 3. Pseudo code for mapping GN pairs to WSN nodes

4 Maintaining the Virtual Topology

In [9] [10], Zou et al refer to the relationship between nodes in the identifier circle as links and classify Chord's links into short links and long links. Nodes adjacent to any given node within the identifier circle are called its successor and predecessor nodes respectively. Long links are introduced to provide short cuts within

the identifier circle. Each node participating in Chord creates a routing table called the *finger-table* containing these long links as its entries. All *finger-tables* contain a maximum of k unique long links, where $k \leq m$, where m is the number of entries in the *finger-table* [4]. Short and long links are prone to disruptions owing to the dynamic nature of WSNs. Therefore, the process of maintaining and verifying the correctness of these links is essential for maintaining the *virtual topology*. The Chord protocol implements three different types of periodic updates for maintaining the *virtual topology* - *stabilize*, *notify* and *fix-fingers*. The *stabilize* and *notify* updates are used for learning about newly joined WSN nodes and node failures in short links. Whereas the *fix-fingers* update is used to adjust the long links by updating nodes' *finger-tables*.

A new WSN node discovers its location in the *virtual topology* after communicating with an existing WSN node in the Chord's identifier circle. Using its *WSN_key* the node successfully finds its location in the identifier circle and updates its adjacent nodes with its position. Subsequently, the newly joined node contacts its successor, i.e. the first node with a higher ID than this node, to acquire all *GN_keys* of the GN pairs that belong to it along with the GN pairs falling in its *WSN_key* range on the identifier circle. When leaving the *virtual topology*, a node migrates its GN pairs and its *GN_keys* to its successor in the identifier circle. The consistency of the identifier circle is preserved by ensuring that a leaving node informs its successor prior to leaving and requests a *stabilize* update routine to be executed by it.

5 Performance Evaluation and Results

The GN algorithm using a Chord overlay for the WSN was simulated using a Java DHT simulator [11]. We assume that all WSN nodes are placed within one hop from each other. Consequently, the network layer routing schemes do not affect data collected from the Chord layer. The simulations implement the application design described in Section 3. The GN pairs of the decomposed input pattern are relayed to their designated WSN nodes by matching their individual *GN_keys* with *WSN_key* ranges of the WSN nodes.

5.1 Simulation Parameters

The *finger-table_entry* parameter depicts the number of entries in the *finger-table* of a WSN node that are actually in use. The variations in the average path length owing to the changes made to the following parameters were studied:

- *finger-table_entries* in actual use,
- network size, and
- the percentage of network nodes with updated *finger-tables*.

Chord suggests using all m entries of the *finger-table* to achieve optimal lookup, where m is the number of bits used in the hashing process. This means that m entries of the table need to be updated regularly to reflect the current

network status and the sensor network needs to dedicate a significant portion of its scarce memory and processing resources to verify the correctness of the *finger-table*. A smaller number of the *finger-table-entries* may lead to sub-optimal lookups, but would make better use of the scarce resources. The effects of varying *finger-table-entries* on the average path length were studied as part of the simulation. Both the *finger-table-entry* parameter as well as the network size were varied in our study. Several simulations were performed with different percentages of WSN nodes getting their respective *finger-tables* updated. Nodes were randomly selected for *finger-table* updates in this case. The average path length, for finding a match, was studied by varying the number of WSN nodes randomly selected for updates.

5.2 Analysis

The result of the simulation for evaluating three most commonly used lookup strategies are shown in Figure 4. The Figure shows the average path length for random, iterative, and Chord key lookups being performed over network size ranging from 64 to 16384 nodes. The iterative lookup sequentially traverses a random set of WSN nodes and hence achieves approximately the same average path length as the random lookup. When compared with iterative and random lookups, Chord markedly outperforms both of these since it takes advantage of the routing information present in its *finger-table*. Average path lengths for Chord stay sub-linear with increasing network sizes and hence provide much better scalability than random and iterative lookups.

As can be seen from Figure 5, the average path length shows a consistent trend over all three simulated network sizes. In each case, the average path length significantly drops after 10% of the network has been updated, thus indicating that the lookup performance can tolerate sub-optimal network updates. This result suggests that *finger-table* updates can be done more selectively and less frequently in the network without significantly increasing the average path length value. Our 32-bit Chord identifiers were derived from the hash of the WSN node locations. Hence, each WSN node was assigned a *finger-table* with 32 entries initially. The *finger-table-entry* parameter was varied between 1 to 32 for network sizes of 1024, 4096 and 16384 to study its effect on the average path lengths. The lookup cost for all the networks stayed consistently high for all *finger-table-entry* values less than 20. The average path length showed a sudden drop for *finger-table-entry* values greater than 20. These results are shown in Figure 6.

Smaller values of the *finger-table-entry* parameter lead to more number of hops for the WSN node search operation. As the value of the *finger-table-entry* parameter is increased, the number of hops required reduces sharply - from size 20 onwards. Smaller *finger-table-entry* values imply that all WSN nodes will simply forward the incoming GN pair keys to their respective successor nodes if the key doesn't belong to their individual ranges. Thus, a sequential traversal of the identifier circle results leads to higher average path lengths. In light of these findings, *finger-table-entry* values up to 20 showed limited or no response to the updates. For greater *finger-table-entry* values, the average path length was sig-

Fig. 4. Average path lengths for random, iterative, and Chord lookup strategies over network sizes ranging from 64 to 16,384 nodes

Fig. 5. Average path lengths for percentage nodes selectively updated over 1024, 4096, and 16,384 node networks

nificantly reduced for node update proportions greater than 10%. The proposed overlay design therefore needs to strike an optimal trade-off between the lookup speed and the corresponding maintenance overhead for the Chord layer. The selective update method was repeated over various *finger_table_entry* values.

The changes in network sizes, as applied in Figure 5 and Figure 6, did not show any significant change to average path length values. We therefore took a constant network size of 4096 for studying the impact of varying the percentage of selective *finger_table* updates on the average path length. Figure 7 shows the optimal trade-off occurring whilst using a sub-optimal Chord's configuration of *finger_table_entry* value of 25. It may be seen from this Figure that, the average path length for *finger_table_entry* value of 25 remains almost constant as the total numbers of WSN nodes with updated *finger_tables* is increased. This implies that having more nodes updated does not significantly improve the average lookup path length. Also, the average path length stays significantly lower than those for *finger_table_entry* values < 20 . It may be pointed out that the above results apply to cases with the *finger_table* set to 32 ($m=32$).

Fig. 6. The effect of varying the *finger_table_entry* on average path length for 1024, 4096, and 16384 node networks

Fig. 7. *finger_table_entry* variations with selective update versus average path length for the 4096 node network

6 Memory and Processing Requirements for Structured WSNs

WSN nodes are usually inexpensive battery-operated devices with low computation and communication capabilities. A typical smartdust sensor node has the following specifications [12]:

```
CPU 8-bit, 4MHz
Storage 8K Instruction flash
512 byte RAM 512 byte EEPROM
Operating System: TinyOS
OS code space 3500 bytes
Available code space 4500 bytes
```

The TinyOS operating system itself occupies 4Kbytes of the flash memory leaving 4Kbytes of storage for the GN and the Chord codes. The *finger_table* has a depth of m rows with each row containing two entries of length m - hash value of the successor node and index of the node. The GN instance on each WSN node was implemented with a maximum of three *bias_arrays*. The *bias_array* size was set to 10 for these simulations. The *key_list* entries were the hashed values of the value-position pairs of the input pattern comprising a character string.

The cost of storage for the GN instance and the Chord protocol on a single WSN node may thus be calculated as follows.

$$Cost_{storage} = \text{finger_table storage} + \text{key_list storage} + \text{bias_array storage} \quad (1)$$

where,

$$\text{finger_table storage} = m * (m + m) \quad (2)$$

$$\text{key_list} = 2 * P * m \text{ bits}, \quad (3)$$

P is the number of pairs in the input pattern

$$\text{bias_array} = 3B \text{ bits}, \quad (4)$$

B is the length of the *bias_ array* = 10

$$Cost_{storage} = 2m^2 + 2Pm + 3B \text{ bits} \quad (5)$$

$$Cost_{storage} \cong 2m^2 \text{ bits} \quad (6)$$

Assuming that the 2Kbytes of memory available for the executables is used in its entirety, the remaining 2Kbytes of memory can be used for storing the *finger_table*, *key_list*, and *bias_array* of the GN instance running over a WSN node.

$$2m^2 \leq 2 * 8 \text{ Kbits} \quad (7)$$

$$m \leq 89 \text{ bits} \quad (8)$$

It may be seen from inequality 8 that the maximum hash key length used for generating both *WSN_keys* and *GN_keys* cannot be greater than 89 bits. Assuming unique identifiers (keys) for each WSN node, it can be deduced that the maximum possible combination of unique keys for the WSN nodes that may participate in the GN pattern recognition process is equal to 2^{89} . This upper bound also determines the maximum length of input patterns that may be stored by the GN array, in the case where each input value, position pair is being mapped to a unique WSN node.

The total cost of computation on each WSN node comprises a summation of the key generation, *finger_table* lookup/update, and the bias lookup/update costs. All three operations defined above are hash calculations. Hence the total cost of computation, in the worst case, based on the hash computation metrics from [13] may be estimated as follows.

$$\begin{aligned}
 Cost_{computation} &\cong key_generate + finger_table & (9) \\
 &lookup/update + bias\ lookup/update \\
 &\cong 1080\ \mu sec + m * 1\ \mu sec + 1\ \mu sec * B \\
 &\cong 1179\ \mu sec\ for\ an\ 89-bit\ hash\ key\ and\ a\ 10\ entry \\
 &bias\ table\ hash\ computation.
 \end{aligned}$$

The pattern analysing experiment for purposes of this paper assumes non-overlapping input patterns for the GN nodes and therefore, at any given time, a sensor node will be performing a single hash operation. In our experiment, the *finger_table* size, m , is taken as 32. Therefore, from equations 6 and 9, the costs for storage and computation for the experiment are 256 bytes and $1122\ \mu sec$ s, respectively. The storage cost for the experiment is well below the maximum storage capacity of 2 Kbytes of a smartdust sensor node, and the cost of computation per node is fairly small and it could thus be safely neglected.

7 Conclusions

We have presented a self-organising scheme, called the *virtual topology*, for imparting a structure to an otherwise unstructured WSN. Chord's adaptive circular structure was proposed to manage the dynamics of the network. The Chord overlay decouples the application (GN) layer from the physical network uncertainties and provides a deterministic virtual environment for supporting the real-time requirements of the application. The GN array and our *virtual topology* utilise the sensor network in a decentralised and balanced manner for performing in-network computations. The simulation results clearly indicate that the deployment of the *virtual topology* for the pattern recognition application on the WSN is feasible both in terms of the memory usage and the computational requirements of a WSN node.

References

1. Chee-Yee, C., Kumar, S.: Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE* **91** (2003) 1247–1256
2. Khan, A.I.: A peer-to-peer associative memory network for intelligent information systems. In: *The Thirteenth Australasian Conference on Information Systems*. Volume 1. (2002)
3. Khan, A.I., Mihailescu, P.: Parallel pattern recognition computations within a wireless sensor network. In: *ICPR*. Volume 1. (2004) 777–780
4. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking* **11** (2003) 17–32
5. Khan, A.I., Isreb, M., Spindler, R.: A parallel distributed application of the wireless sensor network. In: *Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region*. (2004)
6. Iyengar, R., Sikdar, B.: Scalable and distributed gps free positioning for sensor networks. In: *ICC '03*. Volume 1., IEEE (2003) 338–342
7. Savarese, C., Rabaey, J.M., Langendoen, K.: Robust positioning algorithms for distributed ad-hoc wireless sensor networks source. In: *the General Track: 2002 USENIX Annual*. (2002) 317–327
8. Qiangfeng, J., Manivannan, D.: Routing protocols for sensor networks. In: *CCNC 2004*. (2004) 93–98
9. Zou, F., Chen, S., Ma, F., Zhang, L., Tang, J.: Using the linking model to understand the performance of dht routing algorithms. *ISPA 2004* (2004) 544–549
10. Zou, F., Zhang, L., Li, Y., Ma, F.: Effect of links on dht routing algorithms. In: *GCC*. Volume 3032/2004., Springer-Verlag GmbH (2004) 308–315
11. Mao, Y.: Dht java simulator. <http://www.cis.upenn.edu/~maoy/research/> (2004)
12. Perrig, A., Tygar, J.: *Secure Broadcast Communication in Wired and Wireless Networks*. Kluwer Academic Publishers (2002)
13. Ganesan, P., Venugopalan, R., Peddabachagari, P., Dean, A., Mueller, F., Sichitiu, M.: Analyzing and modeling encryption overhead for sensor network nodes. In: *the 2nd ACM international conference on Wireless sensor networks and applications*. (2003)