Using a Prioritized MAC Protocol to Execute the Database Operation Join in Networked Embedded Computer Systems

Björn Andersson

Nuno Pereira

Eduardo Tovar

Filipe Pacheco

HURRAY-TR-100506

Version:

Date: 05-17-2010

Using a Prioritized MAC Protocol to Execute the Database Operation Join in Networked Embedded Computer Systems

Bjorn Andersson, Nuno Pereira, Eduardo Tovar, Filipe Pacheco

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Abstract—Database query languages on relations (for exampleSQL) make it possible to join two relations. This operation is verycommon in desktop/server database systems but unfortunatelyquery processing systems in networked embedded computersystems currently do not support this operation; specifically,the query processing systems TAG, TinyDB, Cougar do notsupport this. We show how a prioritized medium access control(MAC) protocol can be used to efficiently execute the databaseoperation join for networked embedded computer systems whereall computer nodes are in a single broadcast domain.

Using a Prioritized MAC Protocol to Execute the Database Operation Join in Networked Embedded Computer Systems

Björn Andersson, Nuno Pereira, Eduardo Tovar, and Filipe Pacheco
CISTER/IPP-Hurray Research Unit at the Polytechnic Institute of Porto
Email: bandersson@dei.isep.ipp.pt, nap@isep.ipp.pt, emt@dei.isep.ipp.pt, ffp@isep.ipp.pt

***Abstract*—Database query languages on relations (for example SQL) make it possible to join two relations. This operation is very common in desktop/server database systems but unfortunately query processing systems in networked embedded computer systems currently do not support this operation; specifically, the query processing systems TAG, TinyDB, Cougar do not support this. We show how a prioritized medium access control (MAC) protocol can be used to efficiently execute the database operation join for networked embedded computer systems where all computer nodes are in a single broadcast domain.**

I. Introduction

Embedded computers are increasingly networked (through wired or wireless medium) in order to let the sensor reading acquired by one computer node be known to other computer nodes. This increases the number of sensor readings that a computer node knows; hence it improves the perception a computer node has of the physical world which consequently improves the way a computer node can control the physical world.

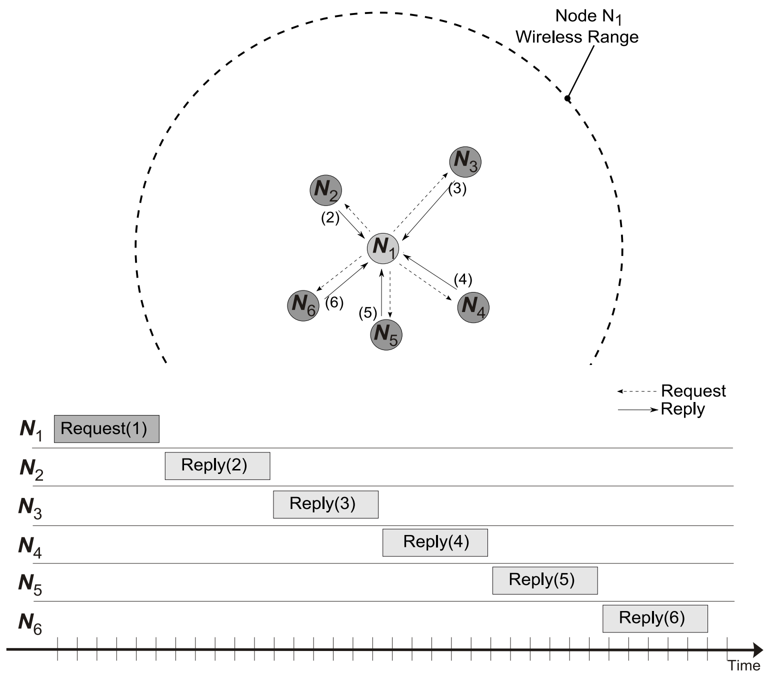
It is possible to share raw sensor readings between computer nodes. Users (human users or application software) are typically not interested in raw sensor readings however; they are instead interested in aggregate quantities of sensor readings, for example MIN or MAX or AVERAGE of sensor readings or a selection of sensor readings. The research community of wireless sensor networks (WSN) has embraced the view of a networked embedded computer system as a query processing system of sensor readings. The community has developed a family of query processing systems (TAG [1], TinyDB [2], Cougar [3] are the most well-known) which allows users to pose queries in an SQL-like language. The use of such a query system makes it possible to reduce the number of transmitted messages thanks to the use of in-network processing (a computer node does not simply forward a received message; the node performs a computation based on the data payload of many received messages and then transmits the result of the computation in a single message). This reduction in the number of transmitted messages leads to lower energy consumption (and hence improved longevity for battery-operated computer nodes).

We believe that a query processing system is potentially of value, not only for WSNs but also for networked embedded computer systems in general. For example, an automotive embedded computer system may be interested in the query `select wheel_id, rotation_speed, breaking_force from wheel where wheel.skidding=true`. It is currently not obvious which query language a query processing system for networked embedded computer systems should use. A good starting point however is to give networked embedded systems the capability to execute queries in the same query language as most desktops/servers use, and this is SQL [4]. Unfortunately, SQL processing systems for WSN (TAG [1], TinyDB [2], Cougar [3]) suffers from two drawbacks:

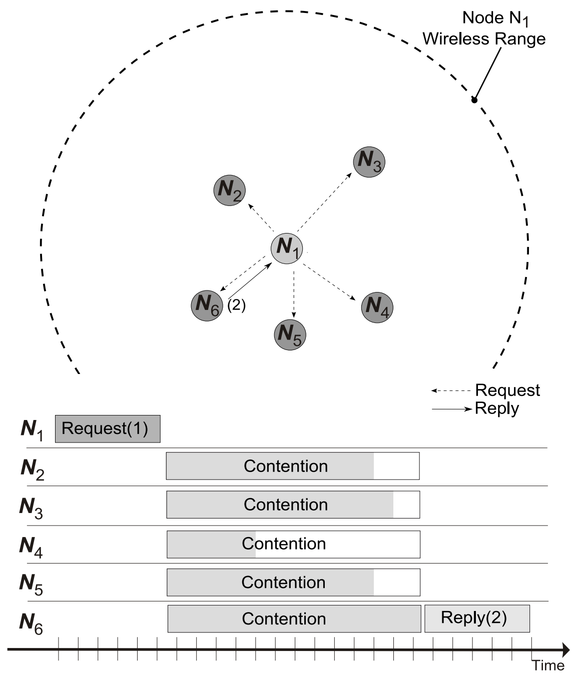
D1. They do not support join operations something that normal SQL engines do. (A join query ”links” two relations together and hence creates a single relation.)
D2. The execution of queries is slow in networks where all sensor nodes are in a single broadcast domain, that is where a single broadcast transmission reaches all computer nodes but no two broadcast transmissions can be made simultaneously (this is the case for a densely deployed sensor network or a set of computer nodes sharing a bus, for example a CAN bus [5]).

Recent research has shown that dominance protocols, a class of prioritized medium access control (MAC) protocol with a very large number of priorities, can be used to efficiently execute *some* queries. MIN [6] of sensor readings can be computed because each sensor node can request to transmit with the priority of the request being the value of the sensor reading and since the MAC protocol elects the computer node with the smallest priority number and makes this number available to all computer nodes, it is possible for all computer nodes to know the minimum sensor reading. Other quantities can be computed as well; MAX [6] of sensor readings, COUNT [7] and also obtaining an interpolation of sensor readings [6]. Common for these algorithms is that the time-complexity of these algorithms does not depend on the number of sensor nodes yet every sensor node impacts the result. Unfortunately, this type of operation for query processing using a prioritized MAC protocol is currently not yet able to execute join queries.

Therefore, in this paper, we present a simple algorithm for executing join queries in networked embedded computer systems. The algorithm uses a prioritized MAC protocol in order to execute efficiently. Our algorithm for joining has the



(a) Obtaining Aggregate Quantities in a SBD (no Parallel Transmissions).



(b) Obtaining Aggregate Quantities in SBD by Exploiting a Prioritized MAC

Fig. 1. Motivating Example: How to obtain aggregate quantities when all nodes are in a single broadcast domain (SBD). (a) Possible solution for the example application using a TDMA-like MAC; (b) Possible solution for the example application exploiting a prioritized MAC protocol.

prominent features that (i) it allows tuples of a relation to be stored on different computer nodes and (ii) it does not require that tuples are pre-sorted and it does not require any indices.

The remainder of this paper is organized as follows. Section II gives preliminaries, that is, the main idea of how a prioritized MAC protocol can be used for computations and also the system model we will use. Section III gives a background on relational databases and join operations. Section IV presents our new algorithm. Section V gives conclusions and future work.

II. Preliminaries

The basic premise for this work is the use of a prioritized MAC protocol. This implies that the MAC protocol assures that out of all nodes contending for the medium at a given moment, the one(s) with the highest priority gain access to it. This is inspired by Dominance/Binary-Countdown protocols [8]. In such protocols, messages are assigned unique priorities, and before nodes try to transmit they perform a contention resolution phase named arbitration such that the node requesting to transmit the highest-priority message succeeds. As a result of the contention for the medium, all participating nodes will have knowledge of the winner's priority.

The CAN bus [5] is an example of a technology that offers such a MAC behavior. It is used in a wide range of applications, ranging from vehicles to factory automation (the reader is referred to [9] for more examples of application fields and figures about the use of CAN technologies). In [6], the authors illustrate that CAN-enabled platforms can be used to compute various aggregate quantities.

The WiDom protocol extends the Dominance protocols to wireless networks consisting of single broadcast domain [10]. Given the growing importance of wireless sensor networks, this extension is significant. Later, that work was generalized to multiple broadcast domains [11]. It is important to note, however, that the implementations of WiDom presented in [10], [11] introduce a significant overhead. To a large extent, this overhead is due to large switching time of transceiver's transmission and reception modes and to the time needed to perform carrier sensing. This problem was addressed successfully in [12] by designing an hardware platform that implements WiDom with a low overhead.

WiDom has also been applied to compute aggregate values of sensor data in multi-hop wireless sensor networks [13]. In this case, the algorithm exhibits a time complexity that depends on the network diameter and on the range of sensor reading values.

The focus of this paper is on exploiting a prioritized MAC protocol for efficiently computing join of two relations. A key idea in the design of such an algorithm is the use of a prioritized MAC protocol for performing computations; this is explained next.

A. The Main Idea

The problem of performing computations in a single broadcast domain can be solved with a naïve algorithm: every node broadcasts its sensor reading sequentially. Hence, all nodes know all sensor readings and then they can obtain the aggregated quantity. This has the drawback that in a broadcast domain with m nodes, at least m broadcasts are required to be performed. Considering a network designed for $m \geq 100$, the naïve approach can be inefficient; it causes a large delay.

Let us consider the simple application scenario as depicted in Figure 1a, where a node (node N_1) needs to know the minimum (MIN) temperature reading among its neighbors. Let us assume that no other node attempts to access the medium before this node. A naïve approach would imply that N_1 broadcasts a request to all its neighbors and then N_1 would wait for the corresponding replies from all of them. As a simplification, assume that nodes orderly access the medium in a time division multiple access (TDMA) fashion, and that the initiator node knows the number of neighbor nodes. Then, N_1 can derive a waiting timeout for replies based on this knowledge. Clearly, with this approach, the execution time depends on the number of neighbor nodes (m).

Consider now that instead of using their priorities to access the medium, nodes use the value of its sensor reading as priority. Assume that the range of the analog to digital converters (ADC) on the nodes is known, and that the MAC protocol can, at least, represent as many priority levels. This assumption typically holds since ADC tend to have a data width of 8, 10, 12 or 16-bit while the CAN bus offers up to 29 priority bits. This alternative would allow an approach as depicted in Figure 1b. With such an approach, to obtain the minimum temperature among its neighbors, node N_1 needs to perform a broadcast request that will trigger all its neighbors to contend for the medium using the prioritized MAC protocol. If neighbors access the medium using the value of their temperature reading as the priority, the priority winning the contention for the medium will be the minimum temperature reading. With this scheme, more than one node can win the contention for the medium. But, considering that at the end of the arbitration the priority of the winner is known to all nodes, no more information needs to be transmitted by the winning node. In this scenario, the time to obtain the minimum temperature reading only depends on the time to perform the contention for the medium, not on m. If, for example, one wishes that the winning node transmits information (such as its location) in the data packet, then one can code the priority of the nodes by adding a unique number (for example, the node ID) in the least significant bits, such that priorities will be unique.

A similar approach can be used to obtain the maximum (MAX) temperature reading. In that case, instead of directly coding the priority with the temperature reading, nodes will use the bitwise negation of the temperature reading as the priority. Upon completion of the medium access contention, given the winning priority, nodes perform bitwise negation again to know the maximum temperature value.

MIN and MAX are just two simple and rather obvious examples of how aggregate quantities can be obtained with a minimum message complexity (and therefore time complexity) if message priorities are dynamically assigned at runtime upon the values of the sensed quantity.

In Section III we will show how this technique of using a prioritized MAC protocol for computations can be used for performing a join of two relations.

B. System Model

The network consists of m nodes that take sensor readings where a node is given a unique identifier in the range $1..m$. MAXNNODES denotes an upper bound on m and we assume that MAXNNODES is known by the designer of the system before run-time. Nodes do not have a shared memory and all data variables are local to each node.

Each node has a transceiver and is able to transmit to or receive from a single channel. Every node has an implementation of a prioritized MAC protocol with the characteristics as described earlier. Nodes perform requests to transmit, and each transmission request has an associated priority. Priorities are integers in the range [0, $MAXP$], where lower numbers correspond to higher priorities. Let $N_{PRIOBITS}$ denote the number of priority bits. This parameter has the same value for all nodes. Since N_{PRIOBITS} is used to denote the number of bits used to represent the priority, the priority is a number in the range of 0 to $2^{NPRIOBITS} - 1$. Clearly, $MAXP = 2^{NPRIOBITS} - 1$.

A node can request to transmit an *empty packet*; that is, a node can request to the MAC protocol to perform the contention for the medium, but not send any data. This is clarified later in this section. All nodes share a single reliable broadcast domain.

A program on a node can access the communication system via the following interface. The `send` system call takes two parameters, one describing the priority of the packet and another one describing the data to be transmitted. If a node calling `send` wins the contention, then it transmits its packet and the program making the call unblocks. If a node calling `send` loses the contention, then it waits until the contention resolution phase has finished and the winner has transmitted its packet (assuming that the winner did not send an empty packet). Then, the node contends for the channel again. The system call `send` blocks until it has won the contention and transmitted a packet. The function `send_empty` takes only one parameter, which is a priority and causes the node only to perform the contention but not to send any data after the contention. In addition, when the contention is over (regardless of whether the node wins or loses), the function `send_empty` gives the control back to the application and returns the priority of the winner.

The system call `send_and_rcv` takes two parameters, priority and data to be transmitted. The contention is performed with the given priority and then the data is transmitted if the node wins. Regardless of whether the node wins or loses, the system call returns the priority and data transmitted by the winner and then unblocks the application.

We consider a set of relations $\{R_1, R_2, \ldots, R_n\}$. A relation is comprised of a set of components (which describe the type of data in the relation) and tuples which describe the actual data. Figure 2a, Figure 2b and Figure 2c show an example; a set of three relations. X1 is a component in the relation Areas and <1,100,100,110,110> is a tuple in the relation Areas.

Relation name: Areas				
AreaId	X1	Y1	X2	Y2
1	100	100	110	110
2	100	110	110	120

(a) The relation: Areas

Relation name: TemperatureSensorReadings		
AreaId	Temperature	Time
1	28	July 7, 2010, 14h21
1	30	July 7, 2010, 14h22
2	19	July 7, 2010, 14h20

(b) The relation: TemperatureSensorReadings

Relation name: HumiditySensorReadings		
AreaId	Humidity	Time
1	55	July 7, 2010, 14h21
1	53	July 7, 2010, 14h22
2	40	July 7, 2010, 14h20

(c) The relation: HumiditySensorReadings

The relation generated by joining Areas and TemperatureSensorReadings on the component AreaId					
X1	Y1	X2	Y2	Temperature	Time
100	100	110	110	28	July 7, 2010, 14h21
100	100	110	110	30	July 7, 2010, 14h22
100	110	110	120	19	July 7, 2010, 14h20

(d) The relation generated by joining Areas and TemperatureSensorReadings on the component AreaId

Fig. 2. Figure (a)-(c) shows examples of relations. Figure (d) shows the result of joining Areas and TemperatureSensorReadings on the component AreaId.

```
1.  Sort $R$ with $Y$ as the sort key.
2.  Sort $S$ similarly.
3.  Merge the sorted $R$ and $S$. We use only two buffers: one for the current block of $R$ and the other for the current block of $S$.
    The following steps are done repeatedly:
      (a) Find the least value $y$ of the join attributes $Y$ that is currently at the front of the blocks for $R$ and $S$.
      (b) If $y$ does not appear at the front of other relations, then remove the tuple(s) with sort key $y$.
      (c) Otherwise, identify all the tuples from both relations having sort key $y$. If necessary, read blocks from the sorted $R$ and/or $S$,
          until we are sure there are no more $y$s in either relation. As many as $M$ buffers are available for this purpose.
      (d) Output all the tuples that can be formed by joining tuples from $R$ and $S$ that have a common $Y$-value $y$.
      (e) If either relation has no more unconsidered tuples in main memory, reload the buffer for that relation.
```

Fig. 3. A simple sort-based join algorithm for joining $R(X,Y)$ and $S(Y,Z)$. This pseudo-code is taken from page 728 in [14]). M denotes the number of tuples per (disk) block.

We use the notation $R(X,Y)$ to mean that the relation R is comprised of the component X and the component Y. We also use $R(X,Y)$ to mean that the relation R is comprised of a set of components X and a set of components Y such that $X \cap Y = \emptyset$.

We are interested in obtaining the result of joining two relations on a certain component. The joining of two relations perform the Cartesian product of all tuples of one relation with all tuples in another relation and then eliminates those resulting tuples where the components on which the join was made on does not match. Figure 2d shows an example of a join operation based on the relations in Figure 2a and Figure 2b.

We assume that all computer nodes know the components of the relations. We also assume that a tuple is stored in exactly one computer node (that is no duplicates on different computer nodes) and it is not the case that different pieces of a tuple are stored on different computer nodes. We make no assumptions on where the tuples are stored however. As an illustration, it is possible that the two tuples in the relation Areas are stored on different computer nodes.

III. Execution of Joins in Traditional Database Systems

Traditional database systems rely heavily on join operations. A straightforward way to join relation $R(X,Y)$ with $S(Y,Z)$ is as follows:

```
FOR each tuple s in S DO
   FOR each tuple r in R DO
      IF r and s join to make a tuple t THEN
         output t
```

```
1.  currenty := min(y: y $\in$ R.Y)
2.  **if** no value is obtained in step 1 **then** terminate the algorithm **end if**
3.  currenty := min(y: y $\in$ S.Y and y $\geq$ currenty)
4.  **if** no value is obtained in step 3 **then** terminate the algorithm **end if**
5.  Each node $N_i$ creates the set of tuples in $R$ that are stored on $N_i$ and which have the value of $Y$ being currenty.
    Let $\mathrm{currentTuplesofR_i}$ denote this set on node $N_i$.
6.  Each node $N_i$ creates the set of tuples in $S$ that are stored on $N_i$ and which have the value of $Y$ being currenty.
    Let $\mathrm{currentTuplesofS_i}$ denote this set on node $N_i$.
7.  for each node $N_i$: $\mathrm{remainingcurrentTuplesofR_i}$ := $\mathrm{current\_TuplesofR_i}$
8.  **while** there exists a node $N_i$ such that $\mathrm{remainingcurrentTuplesofR_i} \neq \emptyset$ **do**
9.      select an arbitrary node $N_i$ with $\mathrm{remainingcurrentTuplesofR_i} \neq \emptyset$ and in this node $N_i$,
            select an arbitrary tuple in $\mathrm{remainingcurrentTuplesofR_i}$. Let currenttupleR denote this tuple.
10.     for each node $N_i$: $\mathrm{remainingcurrentTuplesofS_i}$ := $\mathrm{currentTuplesofS_i}$
11.     **while** there exists a node $N_i$ such that $\mathrm{remainingcurrentTuplesofS_i} \neq \emptyset$ **do**
12.         select an arbitrary node $N_i$ with $\mathrm{remainingcurrentTuplesofS_i} \neq \emptyset$ and in this node $N_i$,
                select an arbitrary tuple in $\mathrm{remainingcurrentTuplesofS_i}$. Let currenttupleS denote this tuple.
13.         the node that stores currenttupleR should broadcast this tuple to all computer nodes receive it.
14.         the node that stores currenttupleS should broadcast this tuple to all computer nodes receive it.
            – note that the broadcasts on line 13 and line 14 form a single tuple; this single tuple is one of the tuples in the result of the join.
15.         for the node that stores currenttupleS, $\mathrm{remainingcurrentTuplesofS_i}$ := $\mathrm{remainingcurrentTuplesofS_i} \setminus \{$ currenttupleS $\}$
16.     **end while**
17.     for the node that stores currenttupleR, $\mathrm{remainingcurrentTuplesofR_i}$ := $\mathrm{remainingcurrentTuplesofR_i} \setminus \{$ currenttupleR $\}$
18. **end while**
19. currenty := min(y: y $\in$ R.Y and y $>$ currenty)
20. **if** no value is obtained in step 19 **then** terminate the algorithm **end if**
21. currenty := min(y: y $\in$ S.Y and y $\geq$ currenty)
22. **if** no value is obtained in step 21 **then** terminate the algorithm **end if**
23. go to step 5.
```

Fig. 4. The new algorithm for joining.

The research literature in traditional database systems have focused on executing join operations faster however; in particular by reducing the number of disk operations. Approaches that are typically used involve (i) clustering (storing those tuples that will be accessed consecutively on the same disk block), (ii) indexing data structures and (iii) sorting before executing the join. The two former do not work in our setting because we assume that only the computer node that stores a tuple knows its location. The latter could be used in our setting but sorting data that is spread out on different computer nodes is costly. In order to design a good scheme for joining relations however it is valuable to understand the simple sort-based join algorithm. Figure 3 shows this.

IV. New Algorithm

The algorithm in Figure 3 assumes that the relations are sorted and this makes it possible to quickly execute the step 3(a), finding the least value y of the join attributes Y that is currently at the front of the blocks for R and S. But sorting is not easy to achieve efficiently in networked embedded computer systems where the tuples may be located on different computer nodes. Note that the find operation can however be performed by finding the minimum value of y which has not yet been seen and for this purpose, our MIN calculation from Section **??** comes in handy. We end up with the new algorithm as shown in Figure 4.

Although this algorithm is much longer than the algorithm in Figure 3, it has the benefit that it can be executed quickly in networked embedded systems. In particular, we note the following about the algorithm in Figure 4:

1) Step 1 can be executed with a prioritized MAC protocol as we discussed in Section **??**. Specifically, for executing step 1, for each node N_i do the following:

```
a) find the minimum value of R.Y stored on node $N_i$.
   This obtained value should be assigned to the variable
   my_prio.
b) currenty := send_empty( my_prio).
```

2) Step 3 can be executed with a prioritized MAC protocol as we discussed in Section **??**. Specifically, for executing step 3, for each node N_i do the following:

```
a) Let currenty be the value of currenty when step 1 was
   executed most recently.
b) Find the minimum value of S.Y stored on node $N_i$ such
   that this value is $\geq$ currenty.
c) **if** this obtained value exists **then**
d)   this obtained value should be assigned to the
e)     variable my_prio.
f) **else**
g)   my_prio should be assigned the highest priority
h)     number (which corresponds to the
i)     lowest priority).
j) **end if**
k) currenty := send_empty( my_prio).
```

3) Step 5,6 and 7 are local operations which need no communications at all.

4) Step 8 and 9 can be implemented with a prioritized MAC protocol. For each node N_i do the following:

```
a) if $\mathrm{remainingcurrentTuplesofR_i} = \emptyset$ then
b)   my_packet := "I have no data"
c)   my_prio :=
d)     highest_possible_priority_number_allowed
e) else
f)   my_packet := pick an arbitrary tuple from
g)     $\mathrm{remainingcurrentTuplesofR_i}$
h)   my_prio := $|\mathrm{remainingcurrentTuplesofR_i}|$
i)     concatenated with the identifier of $N_i$.
j) end if
k) <rcv_prio, rcv_packet> :=
l)   send_and_receive( my_prio, my_packet);
m) if rcv_packet="I have no data" then
n)   the condition of the while loop of line 8 is false.
o)   continue execution on step 19 in Figure 4.
p) else
q)   currenttupleR := rcv_packet
r)   continue execution on step 10 in Figure 4.
s) end if
```

5) Step 10 is a local operation.

6) Step 11 and 12 can be implemented similarly to step 8 and 9.

7) Step 13 and step 14 are normal broadcasts; they do not need the prioritized MAC protocol but the broadcasts must be collision-free.

8) Step 15 is a local operation.

9) Step 17 is a local operation.

10) Step 19 and step 21 can be implemented similarly to step 1 and step 3. Note that step 19 is different from step 1 though.

V. Conclusions

We have shown how a prioritized MAC protocol can be used to efficiently join relations in networked embedded systems. This is important because join is a fundamental building block in database systems which can be used to form arbitrarily complex structures, for example when joining the relations R and S, the relation R may in turn be the result of a join of the relations U and V. And the relation S might be the result of a selection operation on a relation T.

It appears that the join algorithm presented here can be extended for cases where the same data is duplicated on many computer nodes and the case where a component of a single tuple is stored on different computer nodes. But we did not elaborate on these details.

We left open the question on how to create a fully working query processing system (with query parsing, optimization, etc) for networked embedded computer systems and do so based on the prioritized MAC protocol. Also, we did not explore whether a prioritized MAC protocol can be used to reason about predicates in different contexts (so-called multi-context systems [15]–[17]).

Acknowledgment

This work was partially funded by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053, the ARTISTDesign Network of Excellence on Embedded Systems Design ICT-NoE- 214373 and by the Portuguese Science and Technology Foundation (Fundao para Cincia e Tecnologia - FCT).

References

[1] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," in *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI'02)*, 2002.

[2] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems (TODS*, vol. 30, pp. 122–173, 2005.

[3] Y. Yao and J. E. Gehrke, "TinyDB: an acquisitional query processing system for sensor networks," *Sigmod Record*, vol. 31, 2002.

[4] C. D. Donald and R. F. Boyce, "Sequel: A structured english query language," in *1974 ACM SIGFIDET Workshop on Data Description, Access and Control*, 1974.

[5] *CAN Specification, ver. 2.0*, Bosch GmbH, Stuttgart, Germany, 1991.

[6] B. Andersson, N. Pereira, W. Elmenreich, E. Tovar, F. Pacheco, and N. Cruz, "A scalable and efficient approach to obtain measurements in CAN-based control systems," *IEEE Transactions on Industrial Informatics*, vol. 4, no. 2, pp. 80–91, May 2008.

[7] B. Andersson, N. Pereira, and E. Tovar, "Estimating the number of nodes in wireless sensor networks," in *IPP-HURRAY Technical Report - TR-060702*, 2006, http://www.hurray.isep.ipp.pt/widom/hurray-tr-060702.pdf.

[8] A. K. Mok and S. Ward, "Distributed broadcast channel access," *Computer Networks*, vol. 3, pp. 327–335, 1979.

[9] (CiA), "CAN in Automation Website." [Online]. Available: http://www.can-cia.org

[10] N. Pereira, B. Andersson, and E. Tovar, "Widom: A dominance protocol for wireless medium access," *IEEE Transactions on Industrial Informatics*, vol. 3, no. 2, pp. 120–130, May 2007.

[11] N. Pereira, B. Andersson, E. Tovar, and A. Rowe, "Static-priority scheduling over wireless networks with multiple broadcast domains," in *Proceedings of the 28th Real Time Systems Symposium (RTSS'07)*, Tucson, U.S.A., December 2007.

[12] N. Pereira, R. Gomes, B. Andersson, and E. Tovar, "Efficient aggregate computations in large-scale dense WSN," in *15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)*, San Francisco, California, USA, 2009, pp. 317–326.

[13] B. Andersson, N. Pereira, and E. Tovar, "Exploiting a prioritized MAC protocol to efficiently compute min and max in multihop networks," in *5th Workshop on Intelligent Solutions in Embedded Systems (WISES'07)*, Madrid, Spain, 2007.

[14] H. Garcia-Molina, J. D. Ullman, and J. WiDom, *Pearson International Edition*. Database Systems: The Complete Book, 2009.

[15] J. McCarthy, "Generality in artificial intelligence," *Communications of the ACM*, vol. 30, 1987.

[16] ——, "Notes on formalizing context," *IJCAI*, pp. 555–562, 1993.

[17] F. Roelofsen and L. Serafini, "Minimal and absent information in context," *Proceedings of the 19th international joint conference on Artificial intelligence*, pp. 558–563, 2005.