

# Virtual Full Replication for Scalable and Adaptive Real-Time Communication in Wireless Sensor Networks

Gunnar Mathiason, Sten F. Andler  
University of Skövde  
{gunnar.mathiason, sten.f.andler}@his.se

Sang H. Son  
University of Virginia  
son@cs.virginia.edu

## Abstract

*Sensor networks have limited resources and often support large-scale applications that need scalable propagation of sensor data to users. We propose a white-board style of communication in sensor networks using a distributed real-time database supporting Virtual Full Replication with Adaptive Segmentation. This allows mobile client nodes to access, transparently and efficiently, any sensor data at any node in the network. We present a two-tiered wireless architecture, and an adaptation protocol, for scalable and adaptive white-board communication in large-scale sensor networks. Sensor value readings at nodes of the sensor tier are published at nodes of the database tier as database updates to objects in a distributed real-time database. The search space of client nodes for sensor data is thus limited to the number of database nodes. With this scheme, we can show scalable resource usage and short adaptation times for several hundreds of database nodes and up to 50 moving clients.*

## 1 Introduction

A wireless sensor network (WSN) consists of a large number of communicating nodes that each has a limited supply of energy. Sensor nodes (*source nodes*) are often deployed once, at fixed locations, but sensor locations may also change. Client nodes (*sink nodes*) use sensor data, as mobile nodes inside the network or as nodes at the network edge. A number of specific challenges need to be met for WSNs [18]: 1) Usually the sensors are battery operated, thus efficient use of the available energy supply is critical. Once the energy supply is depleted, the network must be recharged or abandoned. Transmissions require a relatively high energy cost, which means that delayed, coordinated and aggregated transmissions may save much of the energy and increase the lifetime of WSN nodes. 2) In comparison with local area networks, communication links in WSNs are several orders of magnitude slower. This gives rise to high latency, in particular for multi-hop transfers. 3) The connectivity is often unreliable, as communication paths may be lost or obstructed during operation, and the communica-

tion range of sensor nodes is irregular and often dependent on battery level. 4) Further, due to power, size and cost limitations, the memory at each node is usually very limited.

WSNs are typically deployed as large-scale sensor applications. There are typically a few edge nodes that use high level sensor information, while mobile in-network nodes may use any of the information from sensors in the network. Typically, there are only a few critical sensor events that must reach the network edge in bounded time, while less time-critical and possibly aggregated data may reach the network edges with best effort. A sensor node is usually the single updater of a specific sensor value, and the value is propagated through the network to be used by one or multiple clients. Many propagation and routing protocols have been proposed to address the challenges within WSNs, for reporting sensor readings and critical timely events to the edges of the network. To reduce resource usage for sensor data streams, data is filtered into application specific events. Typically, sensors are explicitly programmed or use queries [4]. Queries may be declared at network edges and then distributed to sensors [10]. Many communication approaches assume known sensor locations and that localization schemes are available. To the best of our knowledge, no available data-centric scheme uses a distributed real-time database to meet several challenges in WSNs.

Consider a large surveillance or rescue mission, where a large number of sensors are deployed. Sensor nodes may form an infrastructure, but finding and communicating their data, in particular between locations far apart, involves many intermediate sensor nodes. Further, if many users of sensor data move through the area while using the sensor data, routing of sensor data updates quickly becomes a very complex task. We propose a scheme that uses a real-time database with virtual full replication, as an approach for timely access of sensor data and for scalable propagation of updates across the network. This scheme is called *virtual full replication* with adaptive segmentation for sensor networks (ViFuR-ASN). ViFuR-ASN enables large scale sensor networks, it simplifies addressing of data and nodes, it provides in-network storage, data fusion and aggregation by active rules, and it supports fault tolerance by data replication. We believe ViFuR-ASN can meet the challenges of resource usage and scalability, of real-time requirements for local execution, and of the complexity of propagation

from sensors to multiple mobile clients for networks with unreliable connectivity. A distributed real-time database with virtual full replication can be used as a communication medium that allows a *whiteboard* approach for publishing and reading sensor data. Sensor values can be made available to all clients at any location, by publishing sensor values as database objects that can be used at any database node in the system. We have previously developed virtual full replication with adaptive segmentation (ViFuR-A) as a generic scheme to achieve scalability by allocating database replicas only to nodes where the data is currently used [14]. This paper extends ViFuR-A by utilizing properties of wireless sensor networks, to find and allocate database object replicas where clients need them.

## 2 A scalable database approach

To improve predictability, the distributed real-time database system prototype DeeDS [2] stores its database entirely in main memory, avoiding disk I/O delays caused by unpredictable access times for hard drives. To avoid transaction delays due to unpredictable network delays, the database is virtually fully replicated to all nodes, such that there are always local database object replicas for transactions that execute. This makes transaction execution timely and independent of network delays and network partitioning, since there is no need for remote data access during the execution of transactions. A (virtually) fully replicated database with *detached replication*, [19], where replication is done after transaction commit, allows *independent updates*, that is, concurrent and unsynchronized updates to replicas of the same data object. Such independent updates may cause database replicas to become inconsistent, and inconsistencies need to be resolved in the replication process by a conflict detection and resolution mechanism.

In DeeDS, update replication is detached from transaction execution, by *propagation* after transaction commit, *integration* of replicated updates performed at all the other nodes. Conflicting updates are resolved at integration time. Temporary inconsistencies are thus allowed and also guaranteed to be eventually resolved, giving the database the property of *eventual consistency* [16]. Applications that use eventually consistent databases need to be tolerant of the temporarily inconsistent replicas, and this can be achieved for many distributed and embedded applications. In a fully replicated database using detached replication, a number of predictability problems that are associated with synchronization of concurrent updates at different nodes can be avoided, such as agreement protocols or distributed locking of replicas of objects. Further, if the network of database nodes becomes partitioned, the users of the database can continue to execute transactions, since replicas of all used data are available locally. Conflicts that may be introduced during such partitioning are ensured to be resolved at reconnection, by conflict detection and conflict resolution in DeeDS [19].

The concept of Virtual Full Replication (ViFuR) [2, 12]

has been introduced in DeeDS to ensure that all used data objects are available at the local node, and to reduce the resource usage compared to full replication. ViFuR has the advantages of full replication, such as transaction timeliness, simplified addressing of communication, built-in storage and data aggregation, and support for fault tolerance and partitioning. Virtual full replication creates an *image of full replication* to the database user, such that a database client cannot distinguish a virtually fully replicated database from a fully replicated one. By replicating only the objects that are currently in use at a database node, the database will scale with the actual required degree of data object replication, rather than with the number of nodes.

We have previously presented two schemes for ViFuR: ViFuR-S [13] and ViFuR-A [14]. In ViFuR-S, the allocation of segment replicas to nodes are fixed (static). ViFuR-A, however, is a scheme that adaptively changes the allocation to meet access patterns that cannot be pre-specified. The scheme is suitable for soft transactions, where the first access may be delayed, but where subsequent transactions execute timely due to local availability of data. ViFuR-A makes the database scalable under the assumptions that the number of actively used segments (or data objects) is bounded, and the rate of changes to the allocation of replicas is bounded. ViFuR-A includes distributed segment management as a mean of scalable management of replica allocations, and a protocol for allocation (*establishment*) and de-allocation (*removal*) of replicas. Segmentation allows scalable management of replicas by grouping objects that share properties, such as the nodes of residence [14].

## 3 ViFuR-ASN: A database approach for communication in WSN

In this section we present the ViFuR-ASN scheme that tailors our generic ViFuR-A scheme for usage in WSNs. Sensor readings from sensor nodes are published as sensor data objects in a distributed real-time database, and sensor readings are accessed by database operations in transactions. We consider a wireless sensor network system of ten to hundreds of database nodes, possibly supporting hundreds to several thousands of wireless sensor nodes.

### 3.1 A two-tier sensor-database approach

We use a two-tier approach with a *sensor tier* and a *database tier* of nodes, where the database tier has more powerful nodes and greater energy supplies. We assume that both sensor tier nodes and database tier nodes know their own locations, e.g. by utilizing localization schemes. *Clients* use sensor data as sensor data objects accessed at database nodes, and a client can select which database node it currently uses. *Switching* of database nodes is typically done when a client moves to another location in the network (Figure 1). Each database node has a virtually fully replicated database that stores the data objects for the sensor nodes

connected to it, which is typically only a single or a few data objects for each sensor node.

Sensor nodes connect to a suitable (not always the physically closest) database node and updates the corresponding sensor objects in the database. The set of sensor nodes that connect to the same database node makes up a sub-network of sensor nodes, and sensor nodes may connect to a database node over multiple hops in the sensor sub-network. The number of hops that a sensor node uses for its communication with the database node is bounded to the number of sensors in the sub-network. The actual number of hops from a sensor to a database node is typically lower, since sensor nodes are located at all sides of a database node.

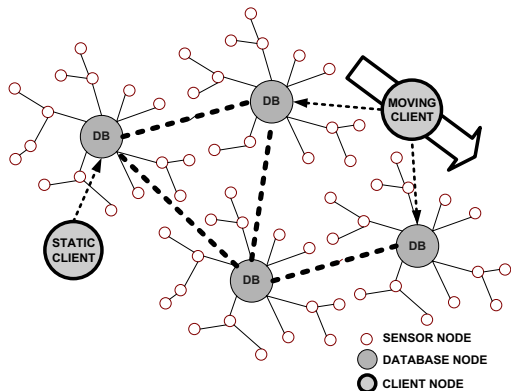


Figure 1. Infrastructure

In this solution, we adopt independent updates, virtual full replication, conflict detection and conflict resolution approaches of DeeDS [1], for timely execution of transactions and for network latency independence. We assume that 1) The network used by the database tier guarantees delivery and hides multi-hop connections, so the database nodes only perceive such connections having a higher propagation delay. 2) Sensor sub-networks use a timely communication protocol. 3) The database tier may use either best effort or bounded time propagation communication. 4) Unique identifiers are assigned when new database nodes are added.

Using a distributed real-time database with virtual full replication in a wireless sensor network will make a local replica available where: 1) A sensor node updates a sensor object at the database node close to the sensor node (this replica is called the *primary replica* of the sensor object) 2) A database node have a client connected that use a sensor object. A client that moves towards a new area where it can connect to another database node, declares its data needs by executing a transaction with operations accessing data needed at the node. If a data object replica is missing there, the database node extends the set of available data replicas by *establishing* a new local replica. When the client moves out of reach of a previously accessed database node, the lack of accesses is detected by a deallocation algorithm, the data object replica is removed.

The primary replica receives updates received from the sensor node. This is typically the only updater for the sen-

sor data object. The distributed database allows concurrent updates at different replicas, and we plan to make use of that in future work, for data fusion or data aggregation. With a single writer for each sensor data object, there are typically no write-write conflicts between replicas. However, clients may suffer from read-write conflicts between replicas, due to usage of stale non-primary replicas that are accessed and used for computation stored back into the database. Further, as an active database, DeeDS can use active rules to select how a database object is updated, which may reduce the amount of database updates locally as well as the amount of propagation of updates.

### 3.2 Database tier

Database nodes are peer nodes that replicate updates in the same way. The Segmentation Manager of a database node is a module that manages allocations and deallocations of replicas in the Data Object Store. Other important modules are the Transaction Manager, which executes transactions, and the Updater Logger, which holds local updates to be sent to other nodes with replicas of updates data objects. Database nodes give unique identifiers to sensor objects. The location of each sensor is stored as a separate database object, and gives spatial context for sensor objects. Updates coming from sensor nodes will update sensor data objects, but necessarily not update the associated location object. Location objects are used by database nodes when searching for sensor information in a particular area of interest. Since all sensor locations are known at the database tier, a search for a sensor for an area of interest is done by searching only at the database nodes, and there is no need to (partially) flood sensor nodes at all.

In a distributed system, a remote user of a sensor value will always use a delayed value, which is caused by the network latency. This is true also with detached replication, as used in DeeDS. With a locally stored value, however, there will always be a value available, which can be used without any delay. This value may be stale, but conflicting usage of it will be detected and resolved by consistency management in DeeDS [19]. Such consistency management may also find and compensate for lost messages or corrupted message ordering on the network, but we leave that as future work. With ViFuR-ASN (and ViFuR-A), there is a setup delay at the first access to an object at a database node, and this is a low price for scalability in non hard-real time applications. For ViFuR-ASN this delay will usually be short since another replica is typically located at a close database node.

### 3.3 Sensor tier

Sensor sub-networks are organized to propagate updates towards a database node. There exist already many approaches in the literature for ad-hoc sensor node organization, message routing, and sensor node localization, to organize networks of sensor nodes. For our approach, we assume that the sensor tier can organize sub-networks around

database nodes, and that such sub-networks can timely deliver sensor updates to database nodes for bounded sized sub-networks. Several approaches exist for timely communication in WSN when there is a bound on the maximum number of communication hops. RT-Link [15] is an approach that allows end-to-end real-time guarantees for WSN. This is a TDMA-based approach that uses global clock synchronization to divide time between sensor nodes. The 802.15.4 communication standard, as a part of the Zigbee initiative, offers real-time communication when the number of sensor nodes in the WSN is known. Koubaa et al. [6] presents an approach for time division communication that schedules beacon frames in a beacon enabled Zigbee communication setting. This approach allows more stringent timeliness requirements to be fulfilled for small sensor networks. Approaches such as SPEED [7] offers soft real-time communication by considering velocities of communication packages propagated through large WSNs.

### 3.4 Establishment and removal of replicas

Any data object in the database may be requested at any database node, by executing a transaction that accesses the object. If a local replica is missing, the transaction is blocked until a local replica has been established. The adaptation protocol of ViFuR-ASN consists of three components: 1) Find the data objects that need to be established 2) Create a consistent replica of a set of database objects from a node where the database is in use 3) Remove unused replicas that have no benefit.

#### 3.4.1 Finding replicas

To establish a new replica, the nodes of existing replicas need to be known. For WSN applications, the interest for a certain area is a more important search type, than a search for a certain data object with a known identifier. The ViFuR-A scheme [14] is identifier-based and uses a replicated directory to list all current replicas and their identifiers. However, a directory is a critical resource that may be a single point of failure, or may suffer from congestions. In ViFuR-ASN, a directory is not needed, since data will always be available within a bounded search: 1) For every sensor object in the database there is a known sensor location, and finding an area of interest is a matter of searching a bounded number of database nodes, instead of flooding an entire sensor network. Further, only those database nodes in the direction of the requested location are searched. 2) For moving clients that continue to use a data set, the location based search has been done once, and the data object identifiers are known. At a switch of the database, the new connected node need only a simple lookup by identifier at neighboring nodes. Such lookups will typically find the data one or a few hops away.

A moving client can choose different strategies for switching nodes. One strategy is to continue the accesses at a certain database node as long as that node can be connected to, and to switch to another database node once the

connection is lost. Another strategy is to evaluate connectivity to connect to the database node with the strongest radio signal. This causes more frequent adaptations and may cause switching to oscillate due to radio irregularity. The strategy to use is a decision of the client, and the different strategies affect resource usage differently. To avoid oscillation, we choose to switch database node only at a lost connection.

If no replica can be found by a location-based search, there is a hole with no coverage in the WSN (a *void*), and the transaction can not be executed. Resolutions to such a problem are application-dependent and may include using sensors at the borders of the void instead, or simply discard the transaction.

#### 3.4.2 Replica request and setup

The second step of establishing a new replica is to load a copy of an existing replica onto the node. ViFuR uses *incremental recovery* [3] to load a replica of an object, or a consistent set of replicas, in use at another database node, without any locking at the other database node [14]. The *recovery source* is the database node holding the existing replica, while the *recovery target* is the database node that will receive a copy of the replica. The most suitable (typically the closest) database node is selected to be recovery source. Figure 2 shows the time line for the establishment of a replica, with database node  $N_n$  as recovery source and database node  $N_1$  as recovery target. The recovery messages are sent with best effort and the blocking time for a waiting transaction depends on both the number of hops searched and the time required for the incremental recovery, including the bandwidth available and the processing time available at the recovery source.

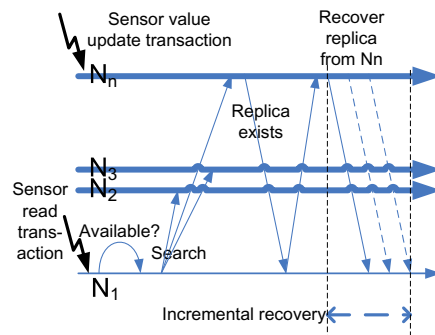


Figure 2. Setup of a new replica

There is no need for a roaming protocol when a moving client switches database node and starts to use a replica at a new database node, and no control messages need to be exchanged at the interface between the database and the database user. Transactions and their read and write operations are sufficient for declaring the data need at each database node independently. Further, the client may choose to submit a "prospective" transaction in advance of actually

needing it, to prepare a replica in advance of the actual first access. This is application-dependent and will work only when the client knows its future direction and the data that it will use there. Such "prospective" allocations may introduce replicas that are not used, but it is a scalable behavior that involves only interactions between close database nodes, and such replicas will soon be removed if they are not used.

The adaptation protocol executes as message exchanges between database nodes, and concurrent adaptations execute independently. The segments at each database node are incrementally updated with commutative add and remove operations. Concurrent adaptations to segments could introduce inconsistencies of the segment information at different nodes, since the list of nodes to inform about a new allocation or removal may be stale. This is solved by letting the recovery source act as a proxy for the recovery target until the replica has been fully established. This means that: 1) User updates for objects being established must be propagated from the recovery source to the recovery target. 2) Changes to allocations at other database nodes in the system must be propagated by the recovery source to the recovery target. Also, once the new replica is established, all other database nodes holding a replica of the established object must be updated with the new allocation by the recovery source, as a last step in the recovery process.

### 3.4.3 Replica removal

The decision to remove an unused replica is taken at each database node with a replica allocated, based on the accesses done there. When accesses cease, the replica is removed unless there is a high utility of keeping it. After removal, other database nodes with a replica of the same object are notified to update their segments. Temporarily after a removal, other database nodes may send updates to removed database objects, but such updates are ignored, and will cease once all other nodes have received the removal notification.

If an unused replica is not deallocated, it will continue to receive updates from other replicas, and it will use resources for no purpose. ViFuR achieves scalability by reducing the number of replicas, and replicas not used will reduce scalability and need to be removed as early as possible after ceased accesses. However, high availability is important for the timeliness of transaction execution, and if accesses can be expected again within a short time, there can be a value of keeping a replica. Thus, there is a trade-off decision to take at the time of removal, whether to keep the replica for high availability or to reduce the number of replicas for scalability. We approach this tradeoff by estimating a *utility* for keeping/removing a replica.

In database buffering and caching schemes, the keep/remove decision is often approached by counting references to objects over time as an estimate of expected future accesses. There is no cost of keeping an object in a local database as long as the buffer memory is not full. With cache coherence schemes there is an relatively small pro-

cessing and communication cost of having a cached copy, since consistency between the cache copy and the next level of storage must be ensured. With a ViFuR database, there are two main differences from traditional database buffering and cache coherence schemes: 1) All transactions execute locally on local replicas of database objects, and there is no secondary database storage to access. For missing objects, the transaction is blocked until local replicas of objects become available. 2) Removal of replicas from the local database storage is beneficial for scalability and for the usage of resources, and there is a relatively high cost of updating such replicas. The high cost of keeping replicas requires that ViFuR uses a different removal strategy.

Inspired by the BYHR [11] database buffering scheme, we measure an average access rate of a replica during the period of allocation. The average access rate is a good estimator of future accesses for a usage period when the access patterns are unknown [11]. More advanced estimations are possible when access patterns are known or can be predicted.

A time period with accesses to a replica is called a *A-episode*, and alternates with a period of no accesses, called a *NA-episode*. NA-episodes shift to A-episodes when accesses are detected. The shift from A-episode to NA-episode is detected by comparing the current measured rate of accesses with the average rate of accesses in the current A-episode. For this, we define the rate-estimate,  $RE$ , being the average rate of accesses during the A-episode (Equation 1, adapted from [11]), where an object  $o_i$ , is accessed  $y_{i,j}$  times by the transactions  $T_j$  during the time  $(t - t_0)$ .

$$RE_i(t) = \frac{\sum_j y_{i,j}}{(t - t_0)} \quad (1)$$

$RE$  is used for detecting the end of an A-episode. When accesses no longer arrive at the expected average rate, an NA-episode starts. There are two different types of  $RE$  calculated for each replica: 1)  $RE_{local}$ , which is based on the rate of accesses from local user transactions 2)  $RE_{remote}$ , which is based on updates received from remote nodes. At the detection of the start of an NA-episode (based on  $RE_{local}$ ), the replica is deallocated only if the cost of continue receiving updates exceeds the cost of establishing it again. If the estimated utility shows that it is beneficial to keep the replica, it is not removed at the start of the NA-episode. During the NA-episode, the utility of keeping the replicas are periodically re-evaluated. The replica will be removed as soon as there is a higher cost to keep it than to re-establish it again.

For utility estimation, we use  $j$  as the distance in hops to another replica of an object to keep at a database node. The highest communication cost to keep the replica depends on the replication cost in number of hops to the most distant other replica. The cost is  $RE_{remote} * s_i * max(hopcount_j)$  where  $s_i$  is the size of the object to keep. The communication cost of establishing a local replica depends on the distance in hops,  $k$ , to the closest recovery source, and the cost is expressed as  $s_i * min(hopcount_k)$ . The utility,  $U$ ,

of keeping the replica is the ratio between those communication costs, as formulated in Equation 2. The database nodes for other replicas of the object is known from the local segments, and the hop counts for those nodes are known by the communication layer; thus  $\max(\text{hopcount}_j)$  and  $\min(\text{hopcount}_k)$  are known.

$$U = \frac{RE_{remote} * \max(\text{hopcount}_j)}{\min(\text{hopcount}_k)} \quad (2)$$

With a utility,  $U$ , below 1 it is beneficial to keep the replica at the time of the start of the NA-episode. For an initial estimate of the NA-episode length we use the measured A-episode length.

A removal decision based on rate estimates may still not be carried out, for five different reasons: 1) If a replica is in use by a recovery source. 2) If a transaction is executing and locks the replica. 3) A newly established replica, not yet accessed by the blocked transaction, can not be removed. The start of a blocked transaction may be delayed by other processing going on at the database node. 4) *Pinned* replicas are explicitly protected from removal and never removed. Primary replicas are always pinned, to ensure that the sensor data is available with at least one database node.

## 4 Evaluation

In previous work we evaluate ViFuR-A by simulation, for a LAN setting [14]. With a simulation, a large number of database and sensor nodes can be evaluated. For this work, the simulation has been extended to model the ViFuR-ASN schemes, and typical patterns of WSN communication. The simulation is written in Java and runs on Java from version 1.5, and has been used by others for evaluations of both centralized and distributed real-time databases [5, 20].

### 4.1 Simulation model

We examine resource usage for the ViFuR-ASN scheme in typical WSN scenario settings, by measuring the usage of network and storage, and the delays of the transactions waiting for establishment of local replicas. We model a large surveillance and tracking scenario, as a field with hundreds of database nodes and several thousands of sensors. Database nodes are randomly placed within the field at a certain average density, and close database nodes connect to each other. Further, a minimum spanning tree of connections are added to the database nodes connections, to avoid partitioning of the network. Sensor nodes are simulated as database objects that have random locations in a certain vicinity of each database node. Update transactions are periodically executed at database nodes with primary replicas, to simulate sensor value updates to the sensor objects. Clients move over the simulated field and connect to one database node at a time, by executing transactions for sensor objects. We measure the scalability of the scheme by increasing key scale factors, such as number of nodes and clients, while

measuring the resource usage. We also measure the behavior at increasing loads, by increasing the number of objects used per moving client and also the speed of moving clients, while measuring storage and bandwidth usage, and transaction delays.

The following simulations were conducted: S1: Clients move at a certain speed across the simulated field while they connect to database nodes and execute transactions for a fixed set of sensor objects. Sensor replicas are established from other (typically close) database nodes which have the replicas required by transactions. The number of clients is increased, while bandwidth, storage and delays are measured. Also, the number of database nodes are increased, while bandwidth, storage and delays are measured. As a baseline comparison, we use the same client behavior for two alternative replication strategies, a database that is fully replicated and a database with no replication of database objects. No replication requires transactions to be moved and executed at the database node of the primary replica, and then returning the result to the database node of client. Simulation S1-1 uses a network with 50 database nodes, where we let 2 to 50 clients move across the field. Simulation S1-2 uses 10 moving clients, and we increase the number of nodes in steps from 10 to 100. S2: Clients move at a certain speed across the simulated field while they connect to database nodes and execute transactions for a fixed set of sensor objects. Further, clients search and request a bounded number of sensor data objects of interest in front of the path of the movement, and establishes replicas for these object at the current database node. The lookup and establishment of frontal objects simulates tracking and following of a target in the field, which is a typical behavior for the intended applications. While clients move, sensor replicas are established at the local database node, for both the fixed set of objects and the frontal database objects, at increasingly higher request rates, where the client speeds are increased and the set of objects accessed is enlarged. With increased adaptation rates resource usage increase, and the adaptation cost in resource usage is measured as storage, bandwidth and delays.

### 4.2 Simulation settings

We model the WSN scenario according to the parameters in Table 1 as default settings, and vary a single or a few parameters for the simulation S1 and S2. We choose to study systems up to 300 database nodes, and up to 50 moving clients, since we consider this to be typical settings in a large scale WSN application. The radio range model for the database nodes is simplified, which can be justified by the fact that we simulate operation in large open field. A few sensor objects are used and updated at each database node, and clients use a few database objects. For simulation S1 we take 10 samples for each measured value for reasonable simulation times, while for simulation S2 we take 30 samples. The Figures for S2 show the 90% confidence interval of our measurements. The field is a square with the size  $\sqrt{N\rho}$  meters, where  $N$  is the number of database nodes and

$\rho$  is the database node density. Using our standard parameters, with 50 nodes and one node per 20000 square meters on average, the field side will be 1000 meters.

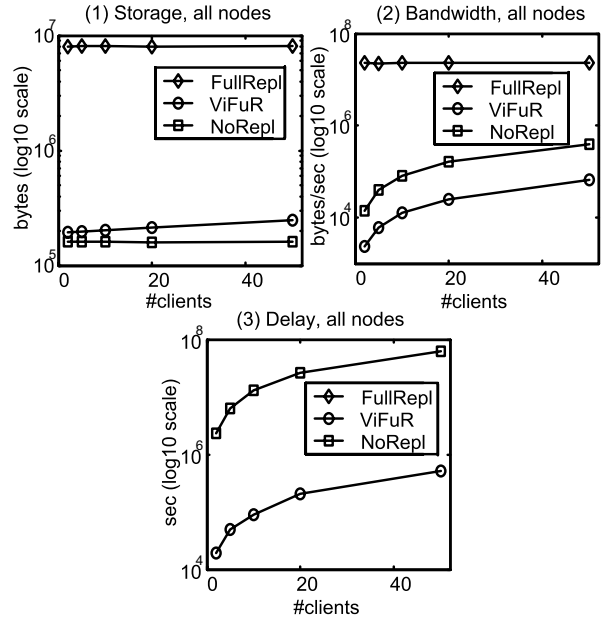
Standard Parameters		
#Database nodes	50	
Database size	50 <i>objs/node</i>	
Data object size	Uniform (1-128) <i>bytes</i>	
Network delay	0.05-1.2 <i>ms</i>	
#Clients	10	
Client speed	30 <i>m/sec</i>	
Simulated time	120 <i>sec</i>	
Database/Database node radio range	100 <i>m</i>	
Database/Sensor node radio range	50 <i>m</i>	
Sensor sensing range	10 <i>m</i>	
Database node density	1/20000 <i>m<sup>2</sup></i>	
Workload		
Sensor update-transaction period	50 <i>ms</i>	
Sensor update-transaction size	50 <i>objs/node</i>	
Client transaction period	50 <i>ms</i>	
Client transaction size	10 <i>objs/client</i>	
Simulation S1 Parameters		
#Clients	10 samples/measurement	
#Database nodes	2,5,10,20,50	S1-1
	10,20,30,50,100	S1-2
Simulation S2 Parameters		
Frontal sensor search angle	30 samples/measurement	
Frontal sensor search distance	+/- 30 <i>degrees</i>	
Max #front sensor objects	200 <i>m</i>	
#Database nodes	10	S2-1
#Clients	20,50,100,150,200,300	S2-1
Client speed	2,5,10,20,50	S2-1
Client transaction size	2,10,20,30,50 <i>m/s</i>	S2-2
	2,10,20,30,50 <i>objs</i>	S2-2

**Table 1. Simulation parameters**

### 4.3 S1: Comparing to alternative strategies

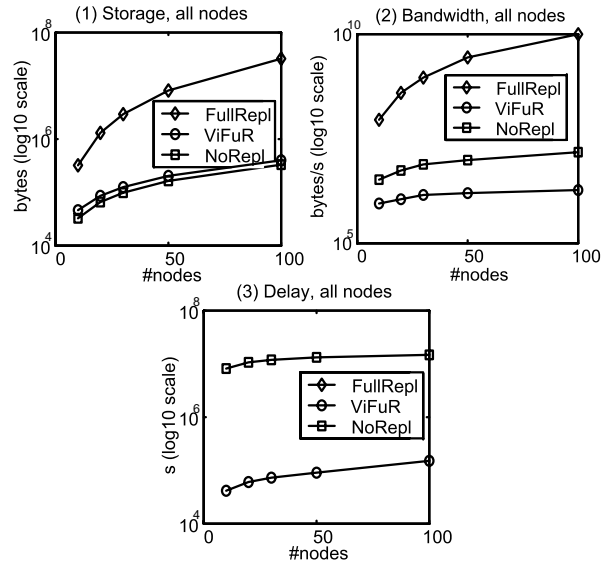
#### 4.3.1 S1-1: Number of clients

We measure the total usage of storage, bandwidth over all nodes, and the summed transaction delay time over all nodes during the simulated time. Our measurements show that full replication ('FullRepl') requires largely more storage and uses largely more bandwidth than the other schemes (Figure 3.1 and 3.2), as all data objects are stored and need to be updated at all nodes. There are, however, no delays for accessing replicas. With no replication ('NoRepl'), storage and bandwidth usage is low (Figure 3.1 and 3.2), while transaction delays are high (Figure 3.3). ViFuR ('ViFuR') uses some more storage than NoRepl (used for the segment data) (Figure 3.1), but uses less bandwidth (Figure 3.2). NoRepl gives very long waiting delays for transactions missing replicas (Figure 3.3), while ViFuR has much lower delays. The storage used with ViFuR increases linearly with the number of clients (Figure 3.1). In a typical workload, ViFuR-ASN reduces storage needs close to the storage used by NoRepl, while bandwidth usage is a magnitude lower than with NoRepl and several magnitudes lower than with FullRepl. Replica setup delays transactions are several magnitudes lower with local transactions, compared to distributed transactions as used for NoRepl.



**Figure 3. S1-1: Storage, Bandwidth and Delay vs. #clients**

#### 4.3.2 S1-2: Number of nodes



**Figure 4. S1-2: Storage, Bandwidth and Delay vs. #database nodes**

For each database node added, another 50 objects are added to the database, while the 10 clients continue to use the same 10 sensor object replicas. For FullRepl the entire database is stored at every node, increasing the database at every database node with 50 objects for each such

node added. For NoRepl and ViFuR each database node will store only the 50 primary replicas. With ViFuR, another 10 replicas are stored for each client at the current database node of the client, which explains the constant difference in storage between ViFuR and NoRepl with more nodes added. FullRepl sends updates for the 10 primary replicas to all other nodes, while updates are sent to one other replica with ViFuR under the load used. The network usage for NoRepl consist of moving transactions to the node of execution, and NoRepl uses up to a magnitude more bandwidth than local execution with ViFuR, including the bandwidth cost both for updates and adaptations. Transaction delays for ViFuR increase slightly will larger system, and can be explained by that some adaptations use more hops.

#### 4.4 S2: Scalability and load

##### 4.4.1 S2-1: Number of nodes and clients

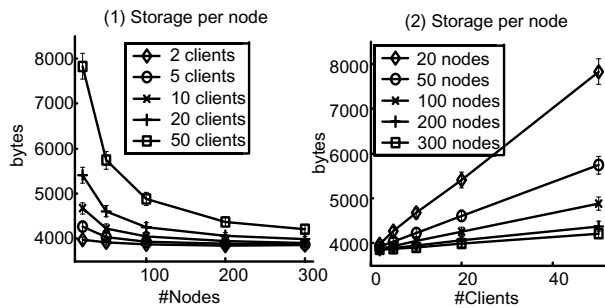


Figure 5. S2-1: Storage vs. #database nodes and #clients

For simulation S2 we measure the resource usage and delay per node. Both storage and bandwidth usage increase linearly with the number of clients (Figures 5.2 and 6.2), since the number of replicas increase linearly with the number of clients. For a database with 50 objects (of average size 64 bytes) we expect to store 3200 bytes of data at each node. With more clients that requests replicas this will be higher, the more clients are in use. The extreme case with 50 clients at 20 nodes is unlikely in a typical system, while the right side of Figure 5.1 is more relevant to the typical storage for 2 to 50 clients. Bandwidth usage increases slightly with number of nodes, since update replication and replica setup use more hops (Figure 6.1). Delays increase linearly with the number of nodes for the same reason: there are more hops in a larger network, and increasing the number of clients does not delay transactions more (Figure 7.1).

Our measurements show that storage per database node increases linearly with the number of clients (Figure 5.2), as would be expected. Each client added to the scenario will add 10 replicas for each database objects it uses. Adding nodes to the system will add to the total database size, since each database node adds 50 unique database objects to the

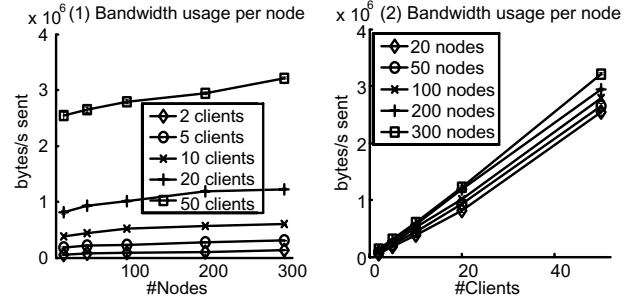


Figure 6. S2-1: Bandwidth vs. #database nodes and #clients

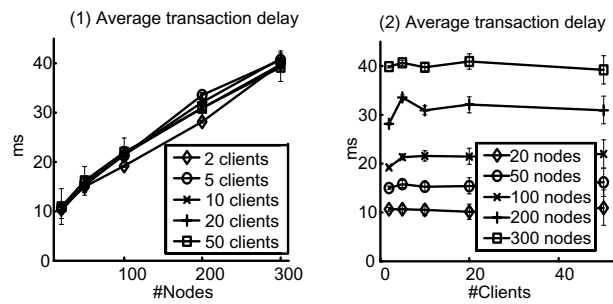


Figure 7. S2-1: Delay vs. #database nodes and #clients

database, of the average object size of 64 bytes. When increasing both the number of nodes and clients there is an exponential growth in storage. The bandwidth usage per database node increase only slightly with more database nodes (Figure 6.1), since there are no more replicas to be updated for more nodes. Database objects not used by clients will need no update messages to be sent out from the database node of the primary replicas. Increasing the number of clients will increase the bandwidth usage linearly, as more replicas are required at the nodes of the clients. The average transaction delays per database node is close to constant (Figure 7.2). Possible resource limitations that could delay replica establishments are not reached for the scale factors we have chosen.

##### 4.4.2 S2-2: Client speed and number of objects/client

Our measurements show that storage increases linearly both with the speed that the client moves over the field (Figure 8.1), and when clients use larger sets of objects (Figure 8.2). The overall database size does not increase, since the number of nodes is constant, but the number of replicas will increase since there are more objects used by each client. Also, with faster moving clients, deallocation will detect later that a replica can be removed, which causes

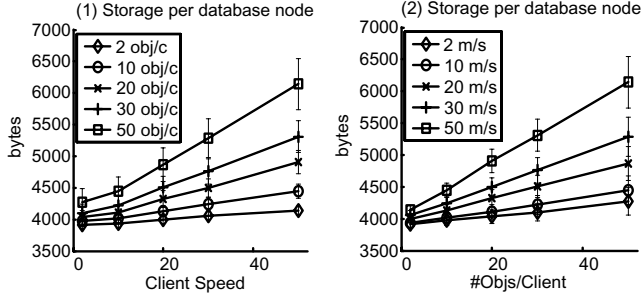


Figure 8. S2-2: Storage vs. Client speed and Objs/Client

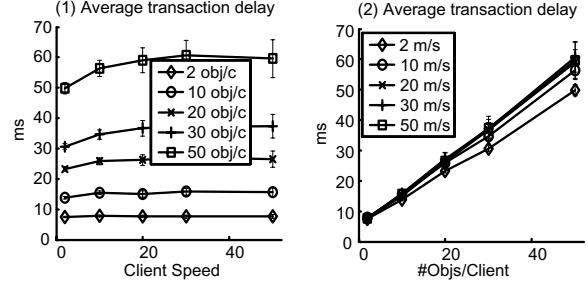


Figure 10. S2-2: Delay vs. Client speed and Objs/Client

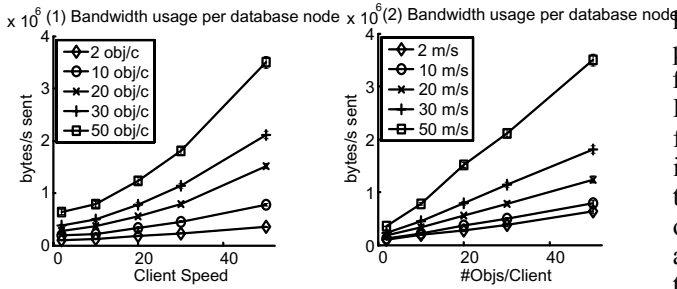


Figure 9. S2-2: Bandwidth vs. Client speed and Objs/Client

more replicas to be concurrently allocated in the database. The nonlinear growth of bandwidth can be solved by letting accesses at new nodes implicitly mean that all but the latest old non-pinned replica may be immediately removed. We will use this improvement in future work. The bandwidth usage correlates with the number of replicas currently in the system, and thus increases close to linearly to the usage of larger sets of objects and the speed the client moves (Figure 9.2). The delays for establishing replicas is close to constant for increasing client speeds (Figure 10.1), meaning that data establishment is always faster than the speeds used for clients in this simulation, and resources are enough for moving data along the client movement. With more data to move in terms of number of objects used by clients, the delays increases, since there are more objects to establish and remove at the nodes.

## 5 Related work

Several alternative data-centric communication approaches exist for WSNs. The Data-Centric Storage (DCS) by Shenker et al. [17] relies on localization geographical routing and peer-to-peer lookup of data, similar to our approach. Updates are propagated towards a limited number of access points and all nodes are assumed to have the same capabilities, although it is recognized that some nodes may

have more resources. Multiple tiers are possible, but not pursued. Much of what is described with DCS can be fulfilled by distributed database, but that is not used by DCS. DCS is mainly intended as a communication-level approach for large-scale systems, but it is not evaluated for scalability. The Cougar sensor database system [4] gives persistent views of sensor data, by applying distributed queries on time-series at the sensors. This reduces communication and enables scalability, since less data is propagated over the network. User's queries are declared at the network edge, and sent as distributed queries by Cougar to the sensor nodes. There is no structured storage in Cougar as with a database, and edge queries are static, since the distributed queries are compiled from user queries off-line. DSWare [8] is a data-centric service middleware for WSNs. It caches data along the propagation path and improves the real-time performance for users of the data. To reduce communication, it uses an event service with a model of normal operation at sensors, to detect anomalies and send events over the network only when needed. Sensor readings are correlated within groups of sensors to improve confidence in sensor readings, and groups also manage faulty sensors. Events to detect are predefined and static, and no evaluation of scalability has been done. TTDD [9] is a tiered approach that assigns a subset of nodes in a course grid within the WSN, where nodes of the grid act as brokers for setting up communication between distant sensor nodes. The grid overlay reduces flooding of searches for data in the network, since the grid nodes represent the sensors in its grid. Multiple mobile sinks are possible, and up to to eight sinks were used in evaluation. TTDD is self-organizing, such that grid maintenance are communication paths are dynamic. TTDD sets up communication per source-sink connection which can be expected not to scale well.

## 6 Discussion and Future work

ViFuR-ASN does not explicitly address global timeliness of updates, or freshness of data. Instead, the proposed infrastructure enables predictable local access time by making local replicas available, and also resolves possible conflicts introduced by independent updates. To ensure timely

data access many other approaches require real-time communication and full knowledge about the combined worst-case load from all database clients. An important novelty of ViFuR-ASN is that it allows most of the data accesses to be done in predictable time by mobile clients, for any sensor data in the network, independently of communication delays and a pre-specification of data needs.

ViFuR-ASN does not make use of differences in frequencies between sensor updates and client reads. With a lower frequency of client reads than sensor updates, the database may propagate updates with the client read frequency, rather than the sensor update frequency. In future work we will utilize such frequency differences. Segmentation of the distributed database may give further benefits not yet pursued by us. Active rules enable filtering when updating the database. Also, active rules can be used for aggregation or information fusion at other nodes than where the primary replica is. In an Information Fusion infrastructure, the database can be segmented for multiple levels of fusion, where fusion nodes reads from segments shared with the lower tiers, and writes into segments shared with upper tiers. In this way a multi-tier distributed real-time database infrastructure can deliver multiple levels of fused data within the area covered by the distributed database. Our next step is to implement ViFuR-ASN for specific WSN applications, such as automatic vehicle detection and precision agriculture, represented by our industrial partners.

## 7 Conclusions

In wireless sensor network, energy-efficient data propagation, scalable behavior, and dynamic reconfiguration are major research issues. We propose a data-centric approach where a distributed real-time database uses adaptive virtual full replication (ViFuR-ASN) to ensure that data is locally available to moving clients of sensor data, for timely access of any sensor data in the surveilled area. We have evaluated scalability and the behavior under increasing load, and we have shown that resource usage for bandwidth and storage scales with the number of simultaneous replicas, and that adaptation delays increase linearly with the scale of the distributed database, in terms of number of nodes. We conclude that a distributed real-time database with virtual full replication is suitable as a scalable white-board approach, for communication in wireless sensor networks, under the typical load described.

## References

[1] S. F. Andler, M. Brohede, S. Gustavsson, , and G. Mathiason. DeeDS NG: Architecture, design, and sample application scenario. In I. Lee, J. Y.-T. Leung, and S. H. Son, editors, *Handbook of Real-Time and Embedded Systems*. CRC Press, 2007. ISBN 1584886781.

[2] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Efring. DeeDS towards a distributed and active real-time database system. *SIGMOD Record*, 25(1):38–40, 1996.

[3] S. F. Andler, E. Leifsson, and J. Mellin. Diskless real-time database recovery in distributed systems. In *Real-Time Systems Symposium (WiP)*, 1999.

[4] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of MDM '01*, pages 3–14, 2001.

[5] K. D. Kang, S. H. Son, and J. A. Stankovic. Service differentiation in real-time main memory databases. In *5th IEEE International Symposium on Object-oriented Real-time Distributed Computing*, Apr 2002.

[6] A. Koubaa, A. Cunha, and M. Alves. A time division beacon scheduling mechanism for ieee 802.15.4/zigbee cluster-tree wireless sensor networks. In *19th Euromicro Conference on Real-Time Systems*, pages 125–135, 2007.

[7] S. Krishnamurthy, T. He, G. Zhou, J. A. Stankovic, and S. H. Son. RESTORE: A real-time event correlation and storage service for sensor networks. In *Proc. of INSS 2006*, May 2006.

[8] S. Li, Y. Lin, S. H. Son, J. A. Stankovic, and Y. Wei. Event detection services using data service middleware in distributed sensor networks. *Telecomm. Systems*, 26(2):351–368, 2004.

[9] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang. TTDD: two-tier data dissemination in large-scale wireless sensor networks. *Wirel. Netw.*, 11(1-2):161–175, 2005.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[11] T. Malik, R. Burns, and A. Chaudhary. Bypass caching: Making scientific databases good network citizens. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 94–105, Washington, DC, USA, 2005. IEEE Computer Society.

[12] G. Mathiason and S. F. Andler. Virtual full replication: Achieving scalability in distributed real-time main-memory systems. In *Proc. 15th ECRTS (WiP)*, pages 33–36, 2003.

[13] G. Mathiason, S. F. Andler, and D. Jagszent. Virtual full replication by static segmentation for multiple properties of data objects. In *Proc. of RTIS 2005*, pages 11–18, 2005.

[14] G. Mathiason, S. F. Andler, and S. H. Son. Virtual full replication by adaptive segmentation. In *Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2007.

[15] A. Rowe, R. Mangharam, and R. Rajkumar. RT-link: A time-synchronized link protocol for energy- constrained multi-hop wireless networks. In *Proc. of 3rd IEEE Sensor and Ad Hoc Communications and Networks (SECON'06)*, volume 2, pages 402–411, 2006.

[16] Y. Saito and M. Shapiro. Optimistic replication. *ACM Computing Surveys*, 2005.

[17] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensor networks. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.

[18] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou. Real-time communication and coordination in embedded sensor networks. *Proceedings of the IEEE*, 91(7), 2003.

[19] S. Syberfeldt. *Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases*. PhD thesis, Linköping University, 2007. ISBN 978-91-85895-27-4.

[20] Y. Wei, S. H. Son, J. A. Stankovic, and K. Kang. QoS management in distributed real-time databases. In *Proceedings of the 24th IEEE RTSS*, pages 86–97, Dec 2003.