IOT: DATA AGGREGATION

SENSOR NETWORKS SOURCES-SINK ARCHITECTURE

Typically, a sensor networks is made of n nodes and a sink node, where the source nodes perform the data gathering, processing and communication.

The sink node, like a base−station, is deployed to collect the information

However, sensor nodes have limited battery power, constraints on lifetime and limited bandwidth for communication.

In several applications (e.g. temperature monitoring), the nodes transmit redundant or correlated information to the sink, which wastes the bandwidth

This leads to extra and unrequired network capacity and draining quickly the device battery depletion.

Therefore, in order to save energy and network capacity data aggregation is required.

MAJOR SOURCE FOR POWER DRAIN

Transmitting one bit over radio is at least three orders of magnitude more expensive in terms of energy consumption than executing a single instruction.

Thus in order to save energy, we should work on reducing the communication time and amount of data to be exchanged.

This can be achieved through data aggregation, which can reduce communication by reducing the number of data packets transmitted among nodes.

DATA AGGREGATION

Data generated by different sensors can be jointly processed while being forwarded towards the sink by locally (in each node) processing raw data before this is transmitted.

One important field of data aggregation in sensor networks is called in-network aggregation.

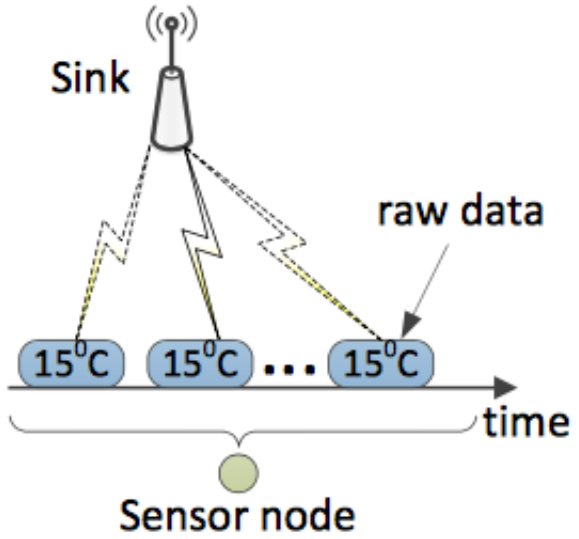
In-network aggregation deals with the distributed processing of data within the network.

Data aggregation techniques are tightly coupled with how data is gathered at the sensor nodes as well as how packets are routed through the network, and have a significant impact on energy consumption and overall network efficiency by reducing the number of transmissions or the length of the packets to be transmitted

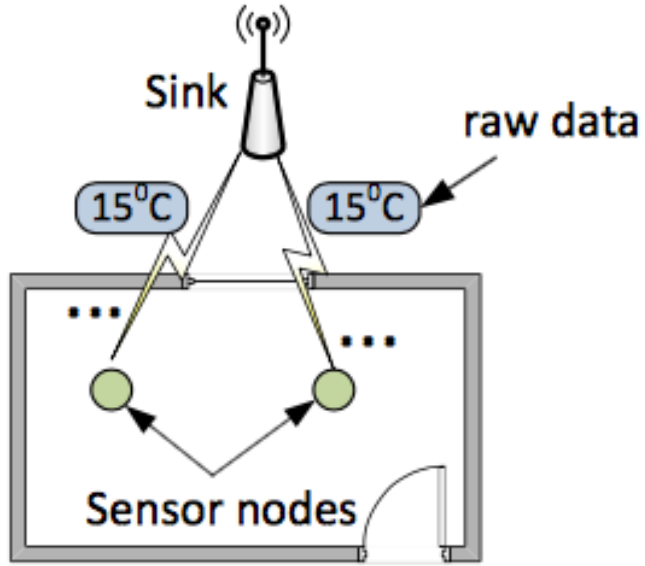
In-network data aggregation can be considered a relatively complex functionality, since the aggregation algorithms should be distributed in the network and therefore require coordination among nodes to achieve better performance. Also, we emphasize that data size reduction through in-network processing shall not hide statistical information about the monitored event.

DATA CORRELATION

- The node sensing the temperature in an area would report the same reading during a period of one hour.
- Also, when two nodes are deployed in a same room to monitor temperature the data obtained can be the same.
- Thus, a data aggregation mechanism process the raw data into a digest, and only send the digest to the sink.

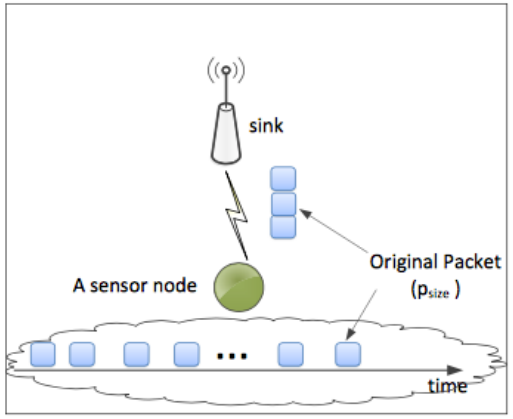


Temporal Correlation

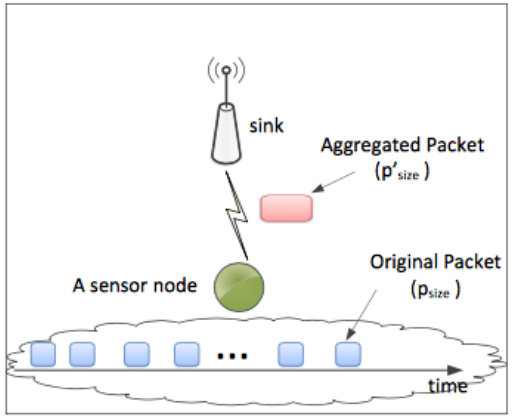


Spatial Correlation

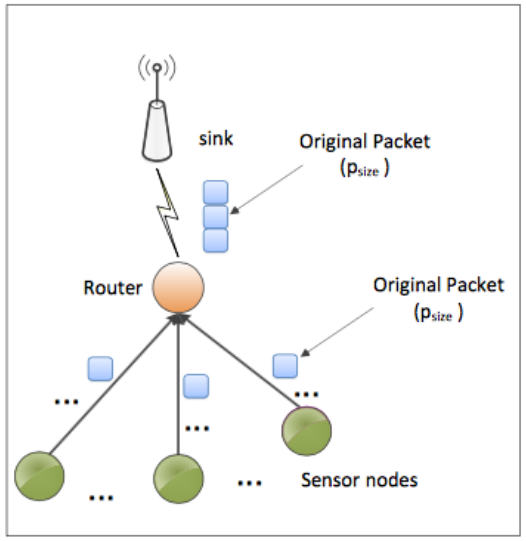
EFFECT OF DATA AGGREGATION



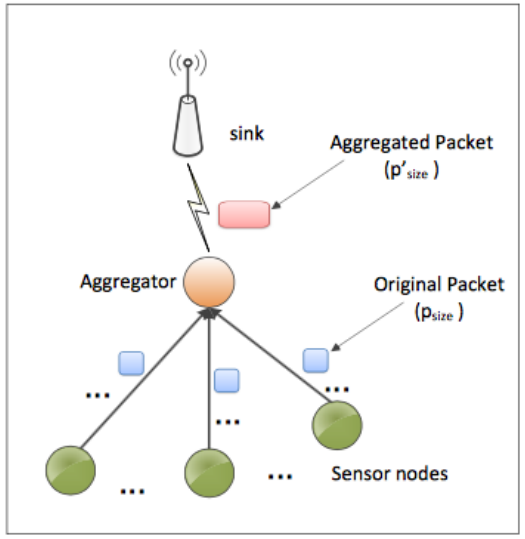
(a) Temporal view without aggregation



(b) Temporal view with aggregation



(c) Spatial view without aggregation



(d) Spatial view with aggregation

FACTORS IN DATA AGGREGATION

1) **Energy saving factor**: Data aggregation reduces the redundant or correlated transmissions in a network, which directly minimizes the energy consumption for the whole network.

2) **Accuracy factor**: Data accuracy is the accuracy change between the recovered data and raw data. Sensor nodes aggregate raw data into a digest which can lead to the loss of information.

3) **Network capacity saving**: Bandwidth constraints of sensor nodes limit the network capacity of the sensor networks. By sending less packets to the sink, data aggregation can save network capacity. Furthermore, how much network capacity has been saved can be seen as a metric to evaluate the aggregation technique.

Other factors might include, security/privacy/latency.

MAIN COMPONENTS IN DATA AGGREGATION

In-network data aggregation techniques require three basic components:

1. Suitable networking protocols.
2. Effective aggregation functions.
3. The aggregation schedule.
4. Efficient ways of representing the data.

ROUTING PROTOCOLS

In-network data aggregation requires routing protocol that are data centic.

Classic routing protocols typically forward data along the shortest path to the destination. If, however, we are interested in aggregating data to minimize energy expenditure, nodes should route packets based on the packet content and choose the next hop in order to promote in-network aggregation.

According to the data centric paradigm, as a node searches for the relay nodes, it needs to use metrics which take into account the positions of the most suitable aggregation points, the data type, the priority of the information, etc.

Altogether, the application scenario, routing scheme, and data aggregation mechanism are closely interrelated.

AGGREGATION FUNCTION

The most important part for data aggregation focuses on how the nodes aggregate raw data into a digest.

An efficient and useful aggregation function helps sensor nodes to reduce energy consumption dramatically.

Two correlations can be found (temporal and spatial correlations) in raw data, and data aggregation function mainly benefits from these correlations.

The aggregation function mainly include simple operations such as Average, MAX/MIN, SUM, COUNT, and Median.

The requirement of high fidelity leads to more complicated aggregation functions.

DATA AGGREGATION SCHEDULING

Data aggregation needs to process the data and then transmit them, thus it leads to delay between source node and sink.

Aggregation scheduling is proposed to solve this problem by defining when a node should aggregate data and when the node should forward data.

The purpose of aggregation scheduling is to boost the quality of aggregated data and balance that with reducing the delay caused by the data aggregation mechanism itself.

In data aggregation the best strategy may not always be to send data as soon as it is available. Waiting for information from neighboring nodes may lead to better data aggregation opportunities which might improve the performance.

Timing strategies are required especially in the case of monitoring applications where sensor nodes need to periodically report their readings to the sink.

DATA AGGREGATION SCHEDULING

Periodic simple aggregation requires each node to wait for a pre-defined period of time, to aggregate all data items received, and then to send out a packet with the result of the aggregation.

Periodic per-hop aggregation is quite similar to the previous approach, the only difference being that the aggregated data is transmitted as soon as the node hears from all of its children. This requires that each node knows the number of its children. In addition, a timeout is used in case some children's packets are lost.

Periodic per-hop adjusted aggregation adjusts the timeout of a node, after which it sends the aggregated data, depending on the node's position in the gathering tree.

DATA REPRESENTATION

Due to its limited storage capabilities, a compact format for data storage is required

The used data structure may vary according to the application requirements.

It might be required that the data structure is not to be common to all nodes, and it should be adaptable to node-specific or location specific characteristics.

Q-DIGEST

A query processing framework for a sensor database to support both single valued queries such as AVG as well as more complex queries like HISTOGRAM.

Q−digest approaches the data aggregation problem as that of trade between compression and accuracy. The more compression we aim for, the less is the transport cost, the less accurate will be the answers to the queries.

It is a data structure for compressed data used to acheive in−network processing and that captures the distribution of sensor data in approximated aggregation.

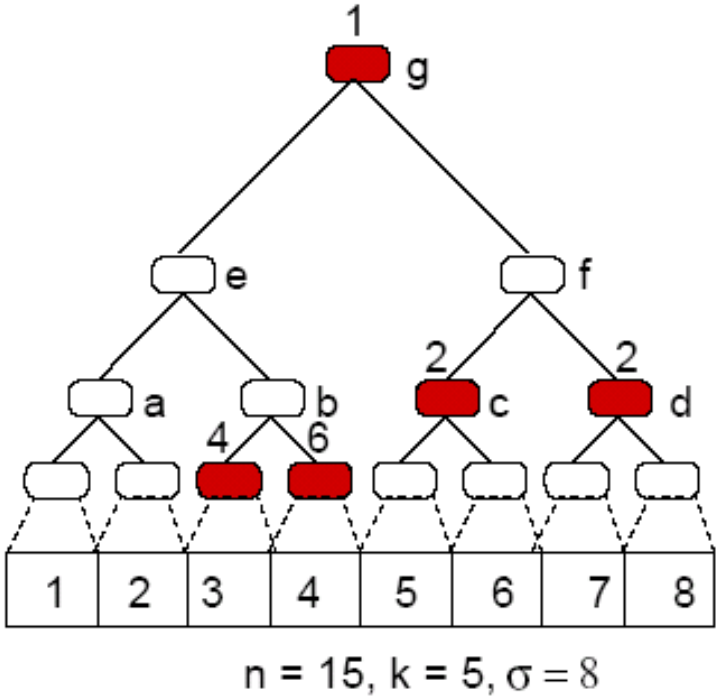
Q-DIGEST PROPERTIES

1. Error–Memory Trade–off : users decide for on the message size and error trade–offs. The error conscious user can set a high maximum message size and achieve good accuracy. A resource conscious user can specify the maximum message size to tolerate.

2. Detailed information concerning data values which occur frequently are preserved in the digest, while less frequently occurring values are lumped into larger buckets resulting in information loss.

3. Multiple Queries : Once a q–digest query has been completed the q–digest at the base station contains a host of interesting information. We can extract information on quantiles, data distribution and consensus values from this structure without further querying the sensor nodes

PROPERTIES OF Q-DIGEST



n = 15, k = 5, $\sigma = 8$

n number of measurements; k compression factor, $\boldsymbol{\sigma}$ the data range

Each node v in tree T is a bucket;

- Whose range [v.min, v.max] defines the position and width of the bucket; E.g., root has a range $[1, s]$ and its two children have ranges $[1, s/2]$ & $[s/2 + 1, s]$
- Has counter; count(v);
- Each bucket v has a count, count(v). vs (sibling count), vp (parent count).

A q−digest is a set of buckets of different sizes and their associated counts;

Q-DIGEST COMPRESSION FUNCTION

Compression factor k determines the size of the q-digest

Q-Digest properties

- 1. count(v) <= $\lfloor n/k \rfloor$
- 2. count(v) + count(vp) + count(vs) > $\lfloor n/k \rfloor$

Root and leaf nodes are exceptions

- A leaf's frequency can be larger than $\lfloor n/k \rfloor$
- No parent & sibling for root

Property 1: Unless it's a leaf, no node should have a high count

Property 2: If two children which are siblings have low counts, we merge them into their parent

BUILDING A Q-DIGEST

Going bottom up to check whether any node violates digest property (2)

- If yes, delete itself and its sibling, and merge to its parent;

Algorithm 1 COMPRESS(Q, n, k)

```
1: ℓ = log σ − 1;
2: while l > 0 do
3:    for all v in level ℓ do
4:       if count(v) + count(v_s) + count(v_p) < ⌊n/k⌋ then
5:          count(v_p) += count(v) + count(v_s);
6:          delete v and v_s from Q;
7:       end if
8:    end for
9:    ℓ ← ℓ − 1;
10: end while
```

$$count(v) \leq \lfloor n/k \rfloor, \quad (1)$$

$$count(v) + count(v_p) + count(v_s) > \lfloor n/k \rfloor. \quad (2)$$

BUILDING A Q-DIGEST

Consider a set of n = 15 values in the range [1, 8] as shown (a).

The leaf nodes from left to right represent the values 1, 2, . . . , 8 and the numbers next to the nodes represent the count.

The number of buckets required to store this information exactly is 7 (one bucket per non−zero node).

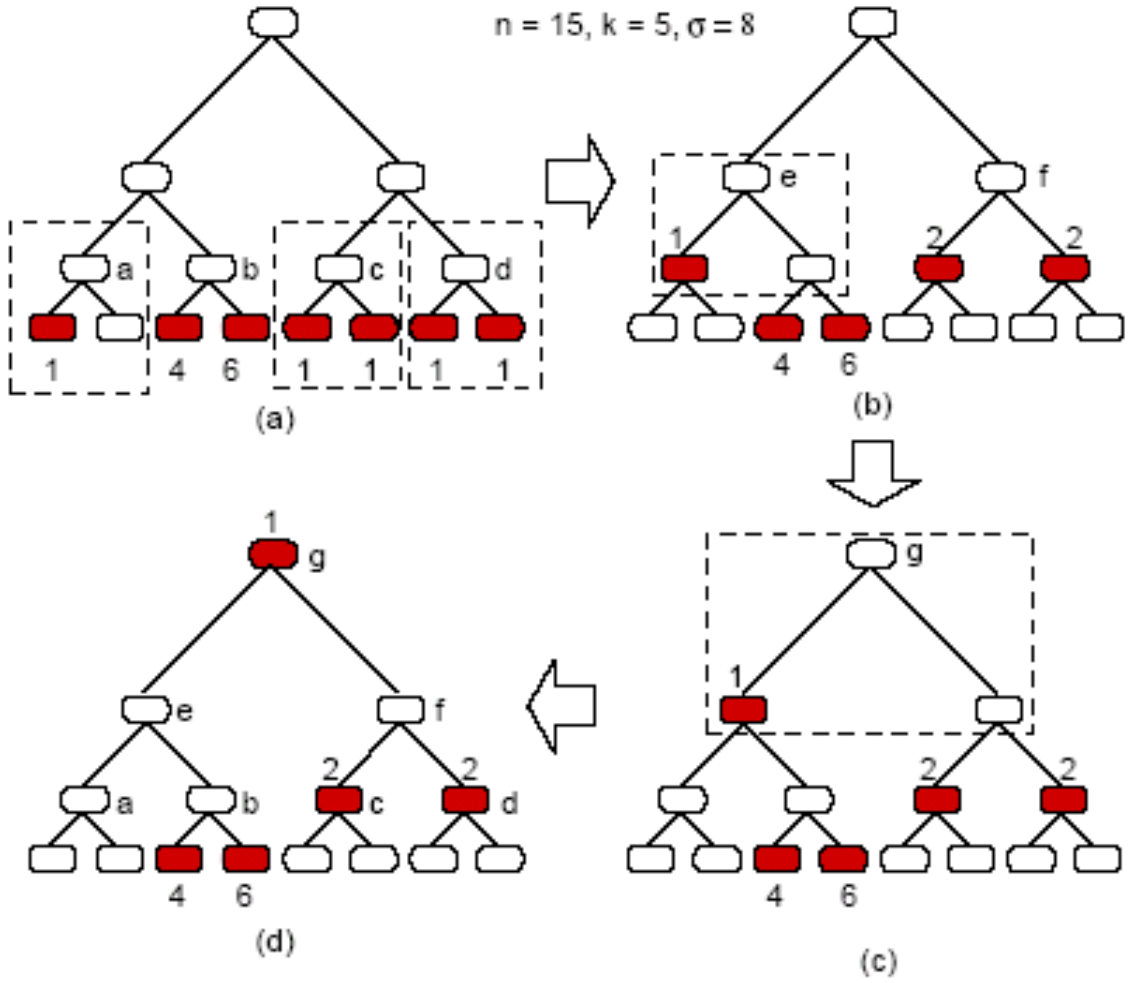
Let us assume a compression factor k = 5, n/k = 3.

In (a), children of a, c, d violate digest property (2). So we compress each of these nodes by combining their children with them.

Thus we arrive at the situation in (b). At this point node e still violates the digest property. So we compress node e and arrive at(c).

Node g still violates the digest property and so we compress g and arrive at our final q−digest shown in (d).

Only 5 nodes instead of 7 are required to store the data representation



MERGING Q-DIGEST

Parent node merge Q1(n1,K) and Q2(n2,K) from children

Algorithm 1 COMPRESS(Q, n, k)

```
1: ℓ = log σ − 1;
2: while l > 0 do
3:    for all v in level ℓ do
4:       if count(v) + count(v_s) + count(v_p) < ⌊n/k⌋ then
5:          count(v_p) += count(v) + count(v_s);
6:          delete v and v_s from Q;
7:       end if
8:    end for
9:    ℓ ← ℓ − 1;
10: end while
```

Algorithm 2 MERGE$(Q_1(n_1, k), Q_2(n_2, k))$

```
1: Q ← Q_1 ∪ Q_2;
2: COMPRESS(Q, n_1 + n_2, k);
```

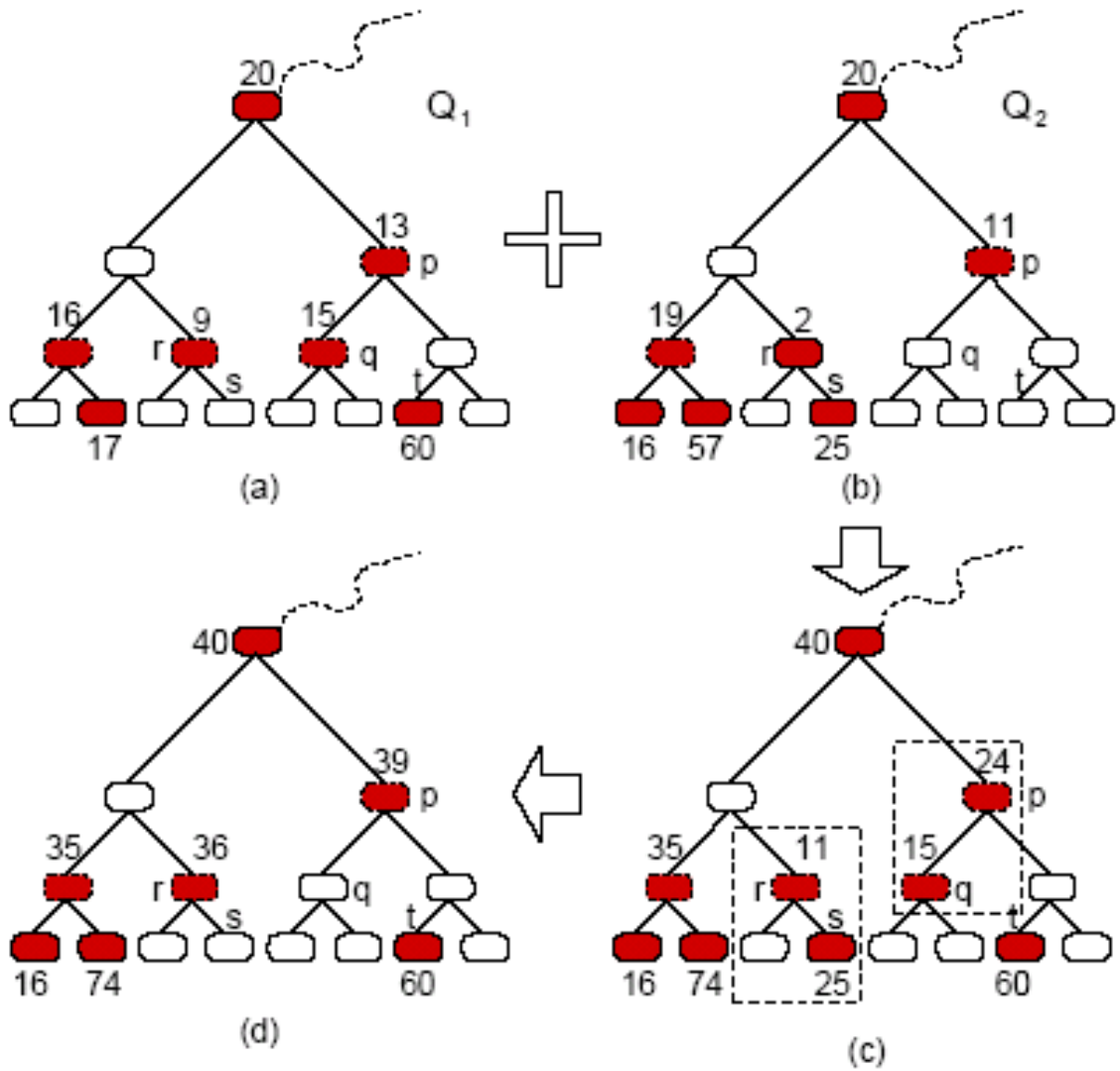
Merging two q-digests Q1 and Q2.

For this example, n1 = n2 = 200, k = 10 and σ = 64.

The tree on the left (a) shows a portion of Q1, and tree in (b) shows the corresponding portion of Q2.

The dark nodes are the nodes included in the q-digest, whereas the light ones are just for visualization. For the final q-digest, n = n1 + n2 = 400 and n/k= 40.

In (c), nodes r and p violate property 2 ($\Delta r = 36 < 40$, $\Delta p = 39 < 40$), so we merge them and produce (d)



ANALYSIS OF ERROR BOUNDS

THEOREM 1. *Given memory m to build a q-digest, it is possible to answer any quantile query with error ϵ such that*

$$\epsilon \leq \frac{3 \log \sigma}{m}$$

PROOF. Choose the compression factor k to be $m/3$. Lemma 1 says that the memory required is m. The error in quantile query:

$$\epsilon \leq \frac{\log \sigma}{k} = \frac{3 \log \sigma}{m}$$

DATA REPRESENTATION

After computing the q-digest structure, each node has to pack it, and transmit it to its parent.

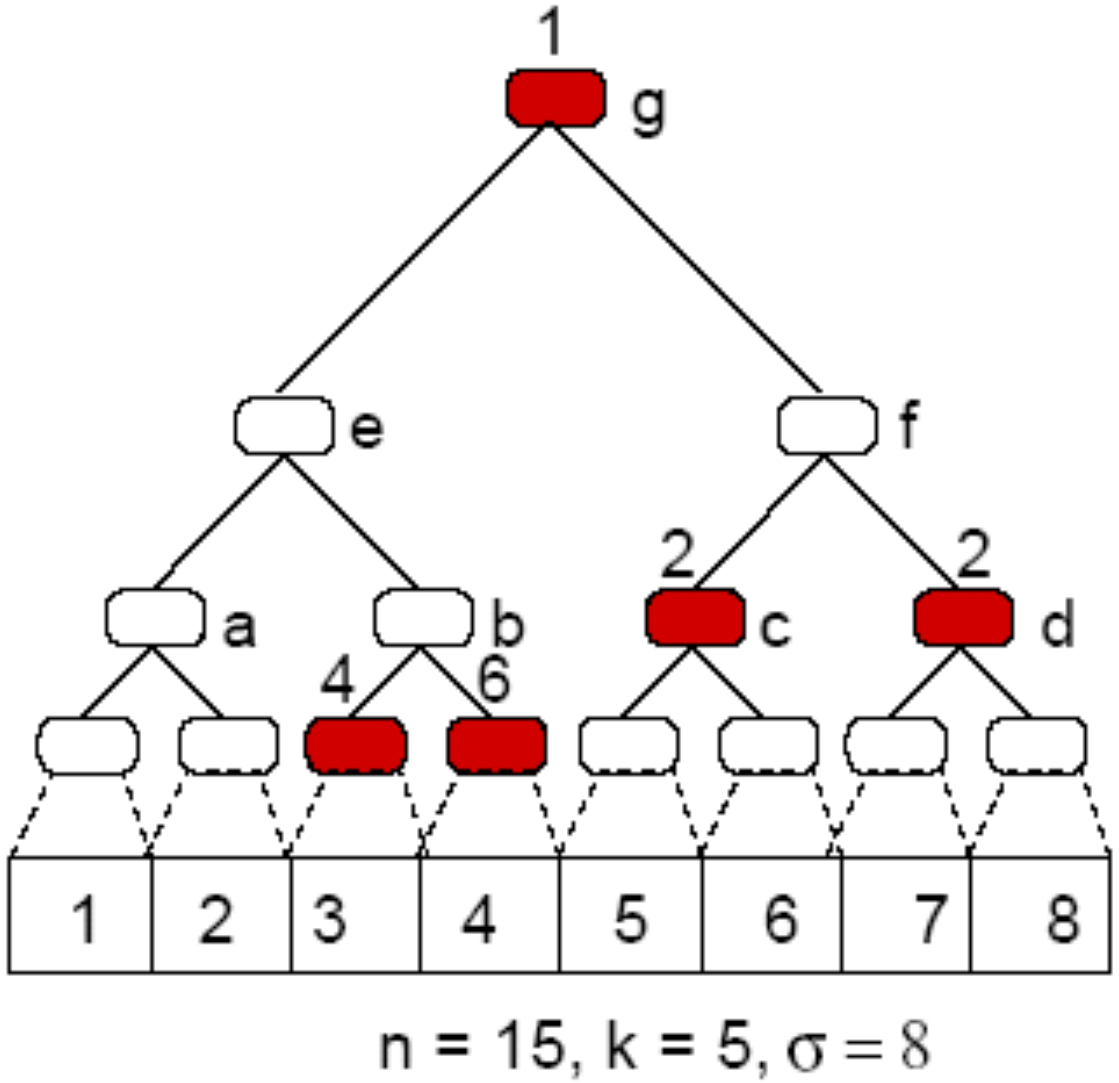
The main limitation of sensor networks is their limited bandwidth.

To represent the q-digest tree in a compact fashion the q-digest is sent as a set of tuple of the following form:

<nodeid(v), count(v)>.

For example, the q-digest in this figure is represented as:

{<1, 1> , <6, 2> , <7, 2> , <10, 4> , <11, 6> }



Quantile Query: Given a fraction q belongs to $(0,1)$, find the value whose rank in sorted sequent of the n value is nq.

- Sort the nodes of q−digest in increasing right endpoints (max values)
- Post−order traversal of list nodes in q−digest
- Scan list L (from the beginning) and add the counts of nodes as they are seen. For some node v, this sum becomes more than qn, we report v.max as our estimate of the quantile.

EXAMPLE

- Q: MEDIAN query on q-digest Q shown in Fig 1. - {<1,1>,<6,2>,<7,2>,<10,4>,<11,6>}
- A: (*post-order*)
  - The sorted list is {<10,4>,<11,6>,<6,2>,<7,2>,<1,1>}
  - The count at node <11,6> will be more than 0.5n (8). Then the answer is 4

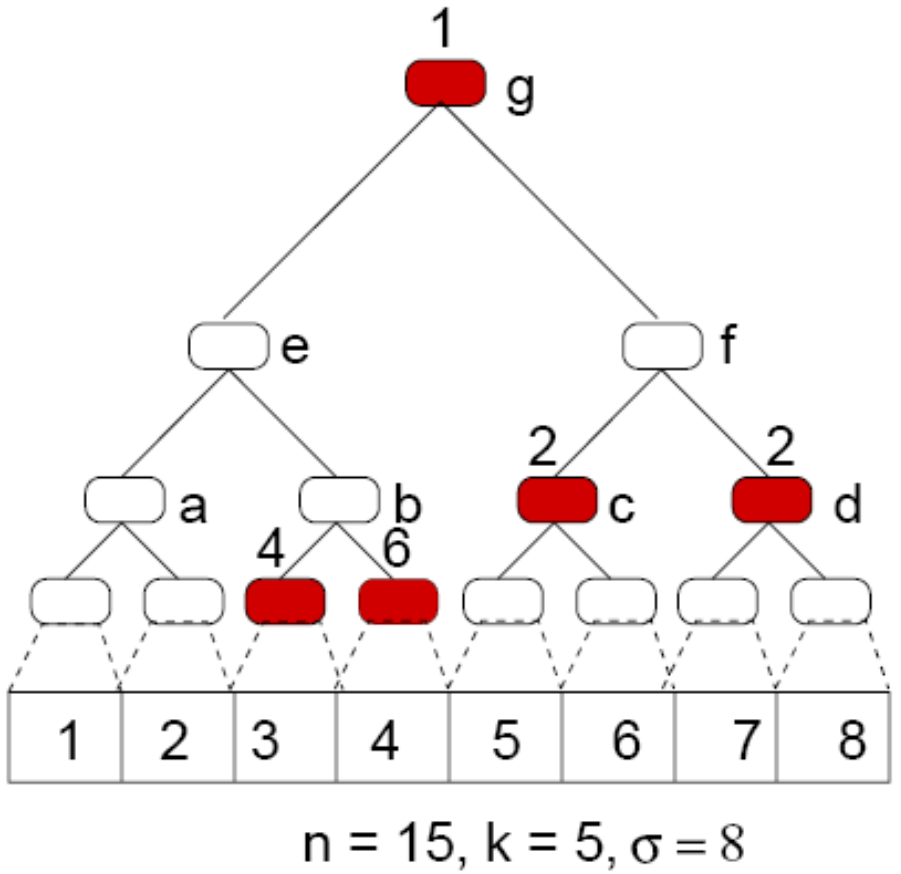


Figure 1: q-digest: Complete binary tree T built over the entire range $[1 \ldots \sigma]$ of data values. The bottom most level represents single values. The the dark nodes are included in the q-digest Q, and number next to them represent their counts.

4.2 Other Queries

Once the q-digest is computed, it can be used to provide approximate answers to a variety of queries.

- **Inverse Quantile**: Given a value x, determine its rank in the sorted sequence of the input values.

  In this case, we again make the same sorted list (L), and traverse it from beginning to end. We report the sum of counts

of buckets v for which $x > v.max$ as the rank of x. The reported rank is between $rank(x)$ and $rank(x)+\varepsilon n$, $rank(x)$ being the actual rank of x.

- **Range Query**: Find the number of values in the given range $[low, high]$.

  We simply perform two inverse quantile queries to find the ranks of low and $high$, and take their difference. The maximum error for this query is $2\varepsilon n$

- **Consensus Query**: Given a fraction $s \in (0, 1)$, find all the values which are reported by more than sn sensors. This can be thought of finding a value on which more than certain fraction of sensor *agreed*. These values are called *Frequent items*.

  We report all the unit-width buckets whose count are more than $(s - \varepsilon)n$. Since the count of leaf bucket has an error of at-most εn (Lemma 2), we will find all the values with frequency more than sn. There will be a small number of false positives; some values with count between $(s - \varepsilon)n$ and sn may also be reported as frequent.

4.3 The Confidence Factor

In Theorem 1 we proved that the worst case error for a q-digest of size m is $\frac{3 \log \sigma}{m}$. But this worst case occurs for a very pathological input set, which is unlikely in practice. Choosing the message size according to these estimates will lead to useless transmission of large messages, when a smaller one could have ensured the same required error bounds. So if the q-digest is computed by setting m to a value for which it is *expected* to deliver the required error guarantees, we still need a way to certify that those guarantees are met. For this, we provide a way to calculate the error in each particular q-digest structure. We call this the *confidence factor*.

If we define the *weight* of a path as the sum of the counts of the nodes in the path, the weight of the path from root to any node is equal to the sum of its ancestors. So the maximum error is present in the path of q-digest with the maximum weight. We define the confidence factor θ as: θ = (maximum weight of any path from root to leaf in Q) / n.

This ensures that the error in *any* quantile query is bounded by θ. Hence, now we can find out the maximum error in any q-digest and discard the query if it does not satisfy the required precision. In experiments, for example, we work with $\sigma = 2^{16}$ and $m = 100$, the theoretical maximum error is $\frac{3 \log \sigma}{m} \approx 48\%$, but we get a confidence factor of $\approx 9\%$ for the q-digest at the base station. This leads to huge savings in terms of transmission cost. Notice that the actual error in query can still be much smaller than θ (in experiments the actual error in the median was close to 2%).

EXPERIMENTAL EVALUATION

Setup

- C++
- Network topology – routing tree
- All pair of nodes within a fixed radio range can be considered as neighbors.
- 1000*1000 area and 1000 sensors (then 2000*2000 area and 2000 sensors)
- Sensor data value: random and correlated

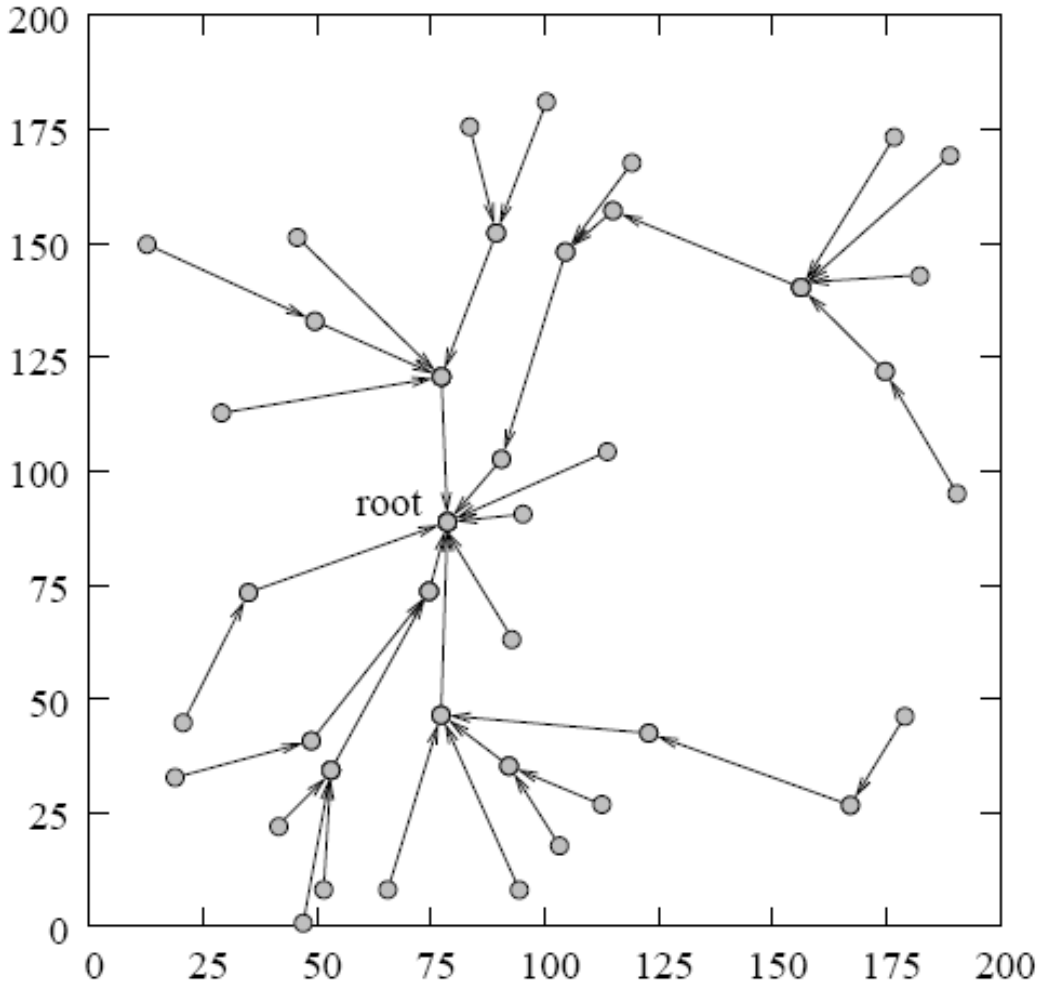


Figure 4: A typical network routing tree for 40 nodes placed in a 200×200 area.

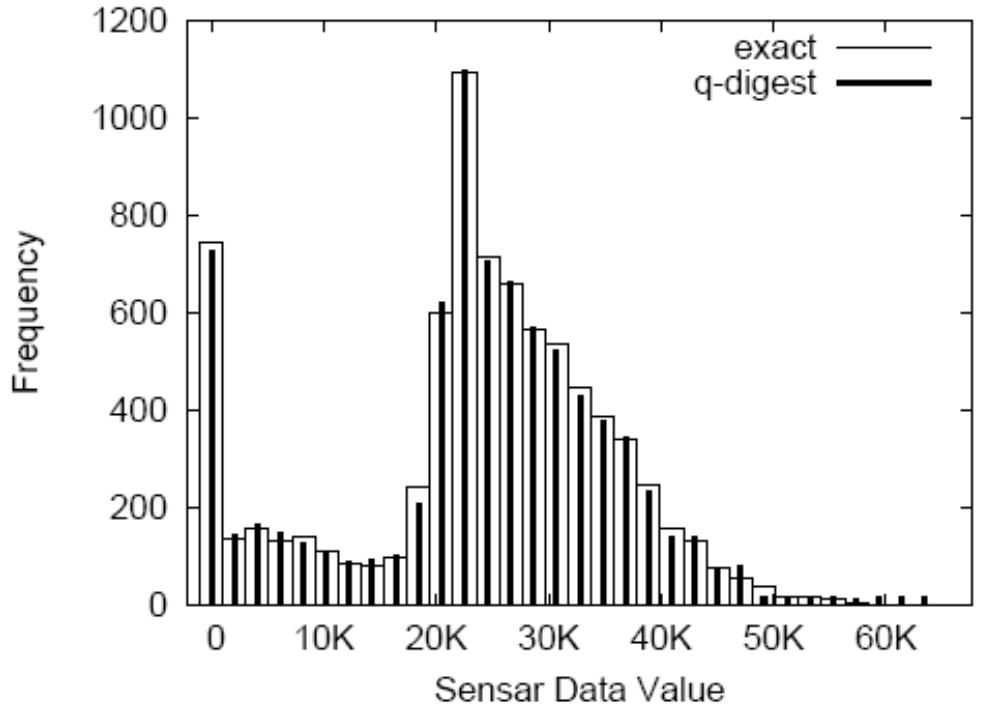


Figure 6: Exact and approximate histogram of input data shown in Fig 5. The open boxes represent the exact histogram while the solid thin bars represent the approximate histogram obtained from q-digest.

RANGE QUERIES AND HISTOGRAM

ACCURACY AND MESSAGE SIZE

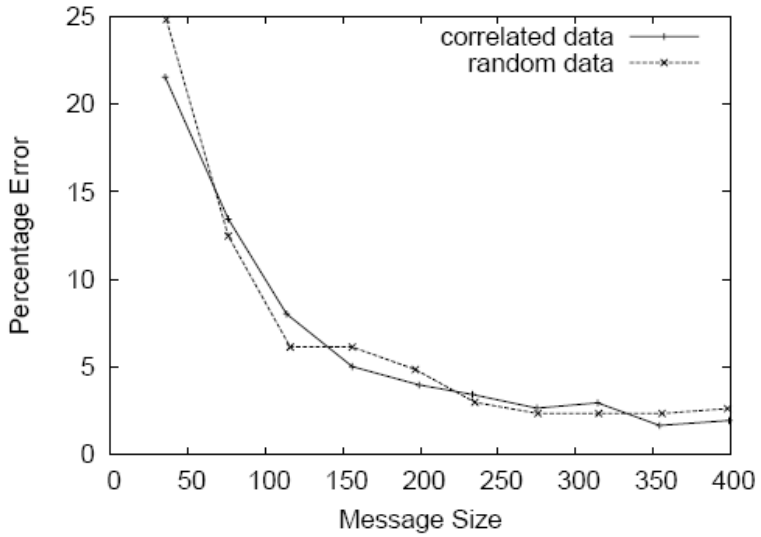


Figure 7: Measured percentage error in median vs message size (in bytes) for an 8000 node network.

ACCURACY AND MESSAGE SIZE

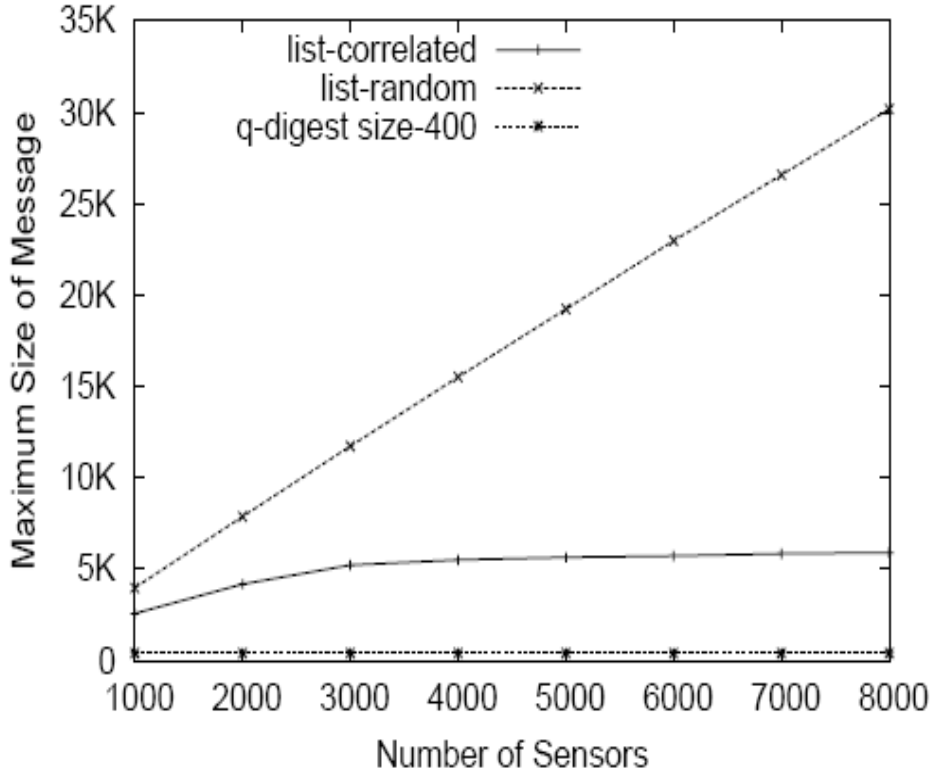
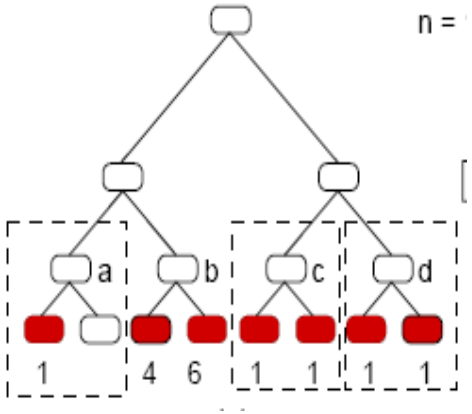


Figure 8: Maximum message sizes for different numbers of sensors for naive unaggregated algorithm and our aggregation algorithm. We fixed message size at 400 bytes which gives about a 2% error (see Fig 7).

What is the *list*?

Q: Why the list-correlated and list-random looks different.



ACCURACY AND MESSAGE SIZE

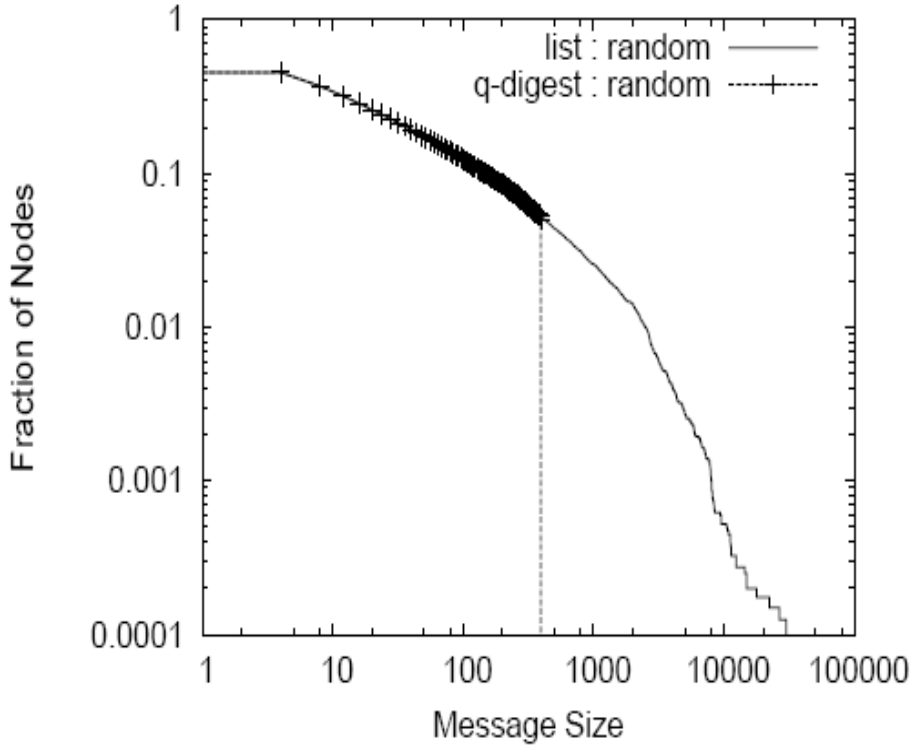


Figure 9: Cumulative Distribution of number of nodes as a function of message size. On the horizontal axis we have message size m, while on the vertical axis we have number of nodes which transmitted messages of size larger than m. Total number of nodes is 8000.

Given a message size m, we ask the question: What fraction of total nodes transmitted messages of size large then m?

TOTAL DATA TRANSMISSION

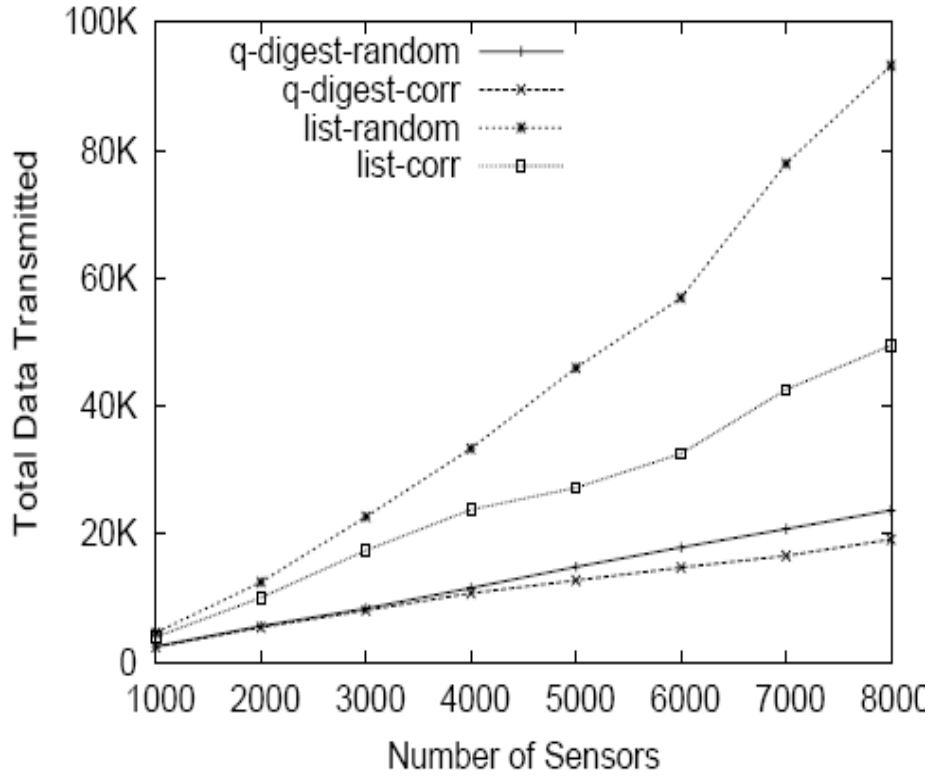


Figure 10: Total data transmitted plotted as a function of total number of sensors for both random and correlated input. The message size for the aggregated scheme was set at 160 bytes.

Sine the number of distinct values is less for correlated scenario, the amount of data transferred is lower for correlated data.