Real-Time Data Reduction at the Network Edge of Internet-of-Things Systems

Apostolos Papageorgiou, Bin Cheng, Ernö Kovacs
NEC Laboratories Europe
Heidelberg, Germany
apostolos.papageorgiou@neclab.eu, bin.cheng@neclab.eu, ernoe.kovacs@neclab.eu

***Abstract*—The expected huge increase in the number of IoT data sources (sensors, embedded systems, personal devices etc.) will give rise to network-edge computing, i.e., data pre-processing, local storage, and filtering close to the data sources. Specifically, data reduction at the network edge (e.g., on an IoT gateway device or a mini-server deployed locally at an IoT area network) can prevent I/O bottlenecks, as well as dramatically reduce storage, bandwidth, and energy costs. However, current solutions face two main obstacles towards achieving this benefits of network-edge computing. Firstly, the most efficient algorithms for data reduction of time series (which is one of the prevailing kinds of data in IoT) are developed to work a posteriori upon big datasets and they cannot take decisions per incoming data item. Secondly, the state of the art lacks systems that can apply any of many different possible data reduction methods without adding significant delays or heavyweight re-configurations. This paper presents a solution that automates the switching between different data handling algorithms at the network edge, including an analysis of adjusted data reduction methods, as well as three flavors of a new algorithm that is capable of performing real-time reduction of incoming time series items based on the concept of Perceptually Important Points. The potential benefits are evaluated upon real datasets from street, household, and robot sensors, showing that our solution achieves accuracies between 76,1 % and 93,8 % despite forwarding only 1/3 of the data items, without adding significant forwarding delays.**

I. Introduction

Wireless and mobile devices are amongst the main enablers of what is currently being described as the Internet-of-Things (IoT). Among others, IoT is supposed to integrate all data sources that have a tight relationship to natural objects or people (e.g., measuring or monitoring them) and provide homogenous and enhanced access to them. Metering sensors (such as smart meters or medical devices), multimedia sensors (such as cameras), machines (of industrial systems or home equipment), mobile phones, cars, planes, people with attached devices, robots with their sonars and thousands other data sources, everything could attempt to forward huge amounts of data points continuously and in real time, so that all kinds of IoT-based applications (public safely, smart transport, e-health, industrial optimization, and more) can be built on top of them. A very high-level view of a typical architecture with which IoT can function is shown in Figure 1. This high-level view is in agreement with reference architectures of related standardization activities (e.g., [4]), but also embraced by industrial research and products [3], [9]. In this scheme, a network-edge device (e.g., a gateway or edge router) collects the data from end devices (usually embedded systems and sensors), and propagates it through the network core to the backend system (usually Cloud databases).

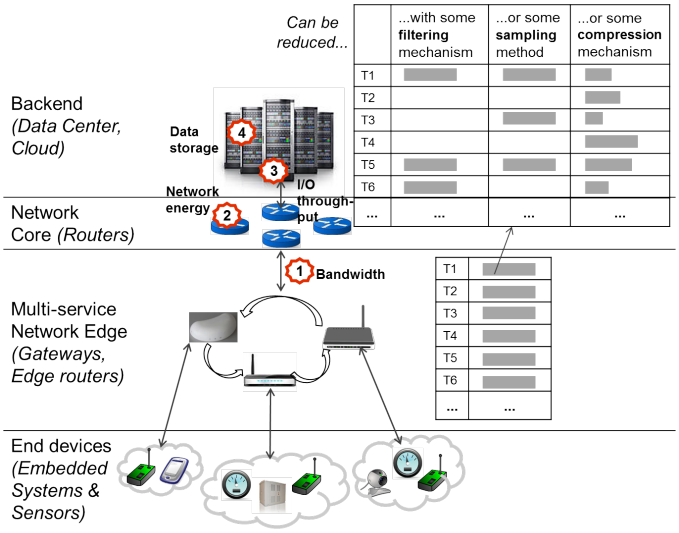

Fig. 1. Illustration of data reduction and its four main benefits upon a high-level view of a typical broadband-based IoT architecture

However, with IoT growing to support billions of devices [7], which might also send lots of unnecessary data items that remain unused, it is already becoming clear that network-edge computing (also referred to as "fog computing" [3], e.g., by Cisco) is necessary, mainly in order to handle and reduce data directly at the network edge. Data reduction at the network edge can relieve four potential bottlenecks of an IoT system. As shown in Figure 1, these are (1) network bandwidth (especially if big data sizes are involved, e.g, with multimedia sensors), (2) network energy consumption (especially if there are many routers dedicated to or heavily loaded by IoT traffic), (3) I/O throughput (especially when big numbers of data items from various sources are reported in short intervals), and (4) Cloud storage and traffic costs (especially if pay-per-volume or pay-per-connection schemes apply). For example, an IoT system that forwards 1 PB per month (ca. 35 TB per day) to the Google Cloud would probably get a bill of ca. $ 2.6M per year (computed based on [8], assuming that only "1 year history" is stored there). These numbers are very realistic even for moderately big systems, provided that a single IP camera can forward 2-50 GB per day, while a single building controller can report Gigabytes of data daily (validated by our test IoT installation in a football stadium [10]), not to mention extreme cases like the 10 TB produced every half an hour by the sensors of a jet.

978-3-901882-77-7

As Figure 1 also shows, data in the form of time series (which are the kind of IoT data that typically causes the aforementioned problems, and thus are the focus of this paper), can be reduced with various approaches, e.g., sampling, selective filtering, or compression. However, the most sophisticated and successful approaches for reducing this kind of data are designed for "a posteriori" data reduction, i.e., once a complete data set is in a database. Such approaches cannot function upon streams, i.e., they cannot take "filter or forward"-decisions per incoming data item. Further, even with many data reduction techniques being known, current systems still require a big effort from a network-edge developer in order to apply, evaluate, and customize them for concrete IoT data sources.

Thus, this paper presents a solution which includes new stream-capable equivalents of sophisticated data reduction algorithms, along with mechanisms for easily instantiating, combining, and comparing them with various other data handlers on IoT network-edge devices. Section II explores the state-of-the-art, Section III presents details of our solution, while Section IV evaluates the accuracy and the delay of our solution upon three data sets from real sensor deployments, showing that our solution achieves accuracies between 76,1 % and 93,8 % despite forwarding only 1/3 of the data items, and thus without adding significant forwarding delays (compared to the standard delay of sending and writing the data into the Cloud databases, which is anyway part of the communication).

II. Related Work

There are two main categories of approaches that can be used for (IoT) data reduction at the network edge (although to-date they are usually applied in different parts of the architecture): i) time series compression approaches and ii) event- or policy-based engines.

General **time series compression and aggregation techniques** intend to reduce time series data by applying techniques that are based on sampling, summarization, approximation, or similar. For example, [15] is based on piecewise approximation (i.e., replacement of every n points with their arithmetic mean), while [5] and [11] are based on selecting only the "Perceptually Important Points" (PIP) of the time series (measuring importance based on the distance from other PIPs and on local minima and maxima, respectively). **However**, such solutions are not designed to act upon streams, i.e., per incoming data item, but rather on already collected data sets ("a posteriori"). This will be better understood in the next sections. Further, they are not complemented by data quality control mechanisms, thus they are often avoided because of the information loss that selective forwarding or data filtering inherently implies. Finally, to the best of our knowledge, they have not been sufficiently applied or evaluated upon modern types of IoT time series, e.g., sensor data from traffic sensors, smart meters, industrial automation, or even smartphones.

Data handling **solutions that use event- or policy-based engines** have often (implicitly) a similar final goal as the above, namely data reduction, though they usually rely on domain-specific rules rather than on time series characteristics. For example, [13] describes how to use IMS (IP Multimedia Subsystems) policies in order to perform -among others- a kind of data aggregation on network-edge (gateway) devices in Machine-to-Machine (M2M) systems. Also Krikkit [6], an open-source solution initiated by Cisco, is in the process of specifying a data format and a mechanism for "telling the network-edge devices" which data to forward and how. **However**, these approaches stop at designing policy engines and policy languages for data reduction, without developing or providing actual data reduction techniques. Thus, they practically require from the network edge developer to implement the data reduction logic from scratch. A recent solution, called FlexAnalytics [17], took a step further by providing flexible placement of analytics for data reduction between the network edge and the backend system. It also includes experiments showing how different data reduction techniques (mainly compression and visualization-based data reduction) perform when applied at the network edge. However, it does not enable new data reduction techniques (or ease existing ones, when they are applied at the network-edge).

III. NECtar Agent: A solution for real-time per-item data reduction

A. Core innovations

There are two main ideas that differentiate this solution from the state-of-the-art:

- The idea to "stream-ify" data reduction techniques that are otherwise not used in real-time scenarios, but rather on complete data sets.
- The idea to enable "single-click" instantiation of data handlers which are runnable at the network-edge and supported by performance indicators, namely indicators of the impact of the data reduction they perform.

However, the solution includes various other differentiating features, e.g.:

- A combination of local (network-edge) caching with remote storage (driven by the IoT scenario that mini data centers can be used close to the data sources).
- A gateway-side library of data reduction handlers (driven by the fact that different IoT data sources require different data reduction techniques).
- Various techniques for avoiding delays in pre-processing and forwarding data items (driven by the principle of real-time identification of important data items).

All of the above are explained in the following subsections, while the "streamified" algorithms and the pre-processing delays are also evaluated separately in Section IV.

B. Method of operation

The middleware of the NECtar agent is normally deployed on a network-edge IoT device (e.g., a gateway) close to the monitored data sources. The middleware provides an API to the gateway applications that collect this data. Although the rest of the paper focuses on time series, the NECtar agent and its data handler switching mechanisms can be used simultaneously for many data sources of different types, e.g., multimedia or semantically-enriched data. With the help of the mentioned API, the gateway applications can instantiate and customize different kinds of data handlers for the data source that they are monitoring, customize the data pre-processing,

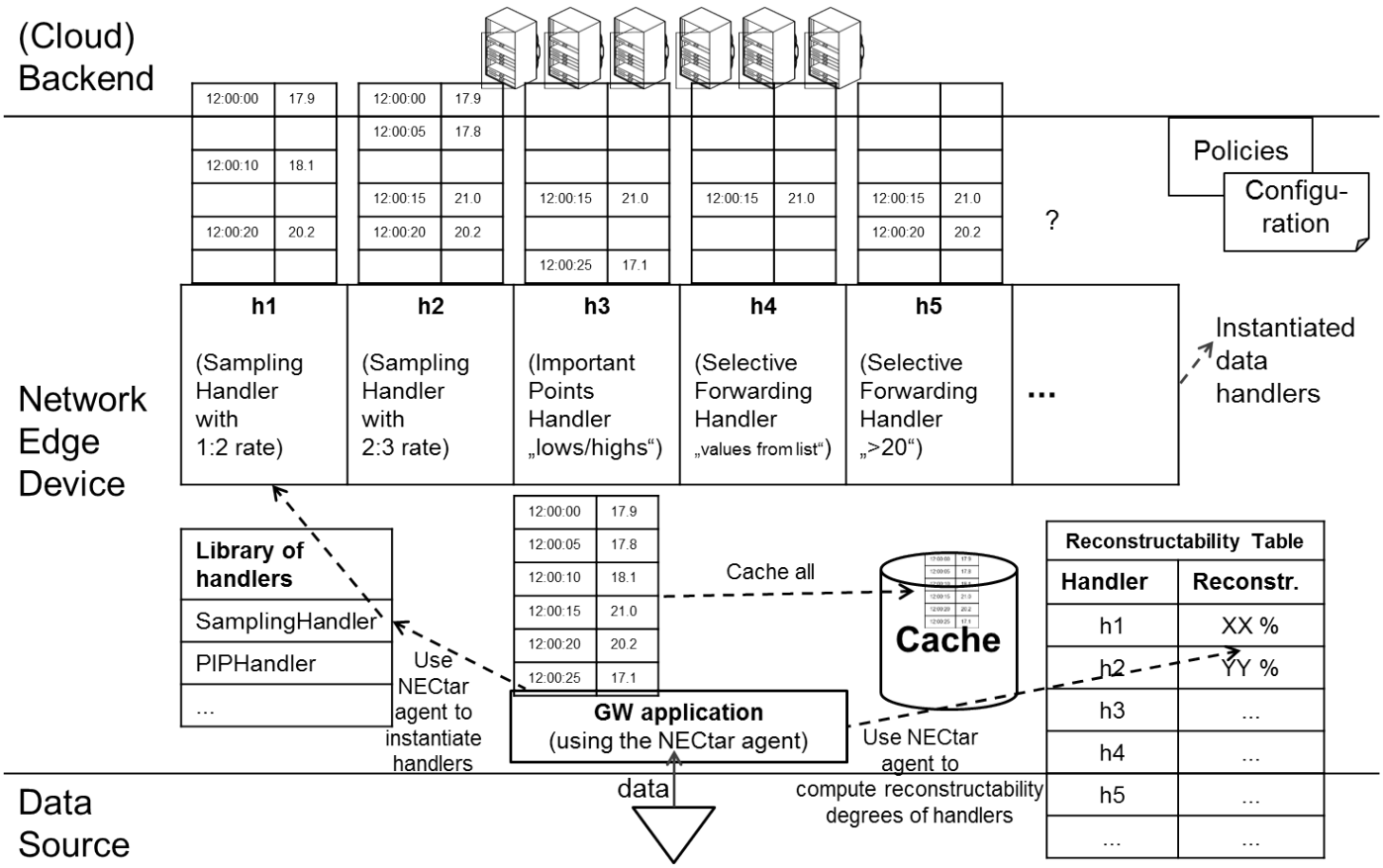

Fig. 2. Overview of the architecture and the operation method of the NECtar agent

and compute information about the reconstructability of the data that is being forwarded.

Figure 2 illustrates how the overall solution operates, providing also some examples of simple instantiated data handlers. It depicts, for example, how certain instances of “sampling”, “important points”, or “selective forwarding” handlers would reduce a given sample of a time series. The data reduction algorithms that are enforced by the handlers, as well as our respective contributions, are discussed in the two following subsections. However, independently of the logic of the data handlers, the given compilation of modules and the chain of actions that takes place upon the instantiation of a data handler enables a gateway application to enforce a data reduction logic in a single line of code.

Further, Figure 2 shows also that for every instantiated data handler, a respective entry is added to a Reconstructability Table, i.e., a table which stores information about how accurately the original time series can be reconstructed based on the items forwarded by the given handler. Support for reconstructability information is part of the framework. However, the details about the definition, the computation, and the way of using reconstructability information in order to select appropriate handlers are hot research topics but out of scope for this paper. Figure 2 depicts also the time series cache, which has multiple functions, namely it is used for (i) potential late retrieval of any data items that were not forwarded, (ii) keeping the history which is required by sophisticated data reduction algorithms, (iii) computing reconstructability degrees based on cache samples. Finally, Figure 2 includes some configurations and policies, which are used for customizing the instantiated data handlers, providing additional forwarding rules, communicating requirements about the reconstructability degree of the forwarded data, and more.

C. Developed or adapted data reduction algorithms

As explained in the previous subsection, the target is to equip the NECtar agent with different handlers that perform data reduction techniques which can be useful in the Internet of Things. Although most of these handlers are based on existing approaches about time series reduction and compression techniques, the NECtar agent faces three additional requirements for its handlers:

- Being applicable in real-time and per item.
- Not delaying the data item forwarding.
- Performing well for important Internet-of-Things scenarios, e.g., safety monitoring, industrial automation.

Therefore, both the adjustment of existing data reduction algorithms and the development of new ones come into play. The latter is especially interesting in the case of domain-specific rule-based selective forwarding algorithms.

However, even without going deep into the specifics of concrete domains and applications, algorithmic work is needed

TABLE I. NECTAR DATA HANDLERS

Name of Handler	Variations	Summary	Sources and new elements
Sampling Handler	Rate = 1:2	This handler performs regular sampling without looking into the data, e.g., for rate 1:2 it forwards every second data item that it receives.	Sampling is very simple and it has been analyzed in numerous works, e.g., [1]. The only thing that needs to be taken care of when applying sampling to a time series stream is the consistent keeping of state (in order to know when its forwarding time), as well as the decision if sampling should be also based on timestamps or simply on the data item sequence.
	...		
	Rate = M:N		
Piecewise Approximation Handler	Window size = 2	This handler forwards an average value for every N values (depending on the used window size), potentially taking variation into account.	This technique is also quite simple and its state-of-the-art form is described in [15]. Its version for IoT data streams should just take care to update the average iteratively upon every incoming item in order to avoid delays when window sizes are too big. Further, it needs to carefully exclude non-numeric values from the process (and maybe forward).
	...		
	Window size = N		
Selective Forwarding Handler	GEQ	Forwards values greater than a threshold.	Selective forwarding is also an obvious technique but it is included here because its operators and their combinations can be used to implement more sophisticated domain-specific rule- or event-based forwarding techniques which are of interest for IoT applications. [16] gives a good example of how selective forwarding could work in the logistics domain.
	LEQ	Forwards values lower than a threshold.	
	IN	Forwards values in specific ranges.	
	LIST	Forwards values from a concrete list.	
	ALL	Forwards everything.	
Important Points Handler	Based on distance	These algorithms forward so-called Perceptually Important Points. For general-purpose solutions, importance is judged based on one of various possible statistical criteria (time series minima-maxima etc.), though the combination of these criteria with domain- or application-specifically important points is also very promising for future IoT data handlers.	This is the most characteristic class of algorithms that normally works upon entire time series and requires new solutions when it comes to data streams. For example, [5] orders the data items iteratively in order of importance based on their distance to previously selected (important) items. This cannot be applied as such for an incoming item of a stream, firstly because an incoming item is always the last point of a series and would always be selected as most important, and secondly because it might be a bad idea to perform the process for the entire series for every item. [11] and [14] judge importance based on local minima/maxima and on a lattice (layers of important points), respectively. For streams, they face similar issues like [5]. Therefore, new data handlers have been developed for the NECtar agent, based on similar concepts. The next subsection provides details.
	Based on local extremes		
	Based on a lattice		
Change Detection Handler	Lossless	Forwards every data item that is different from the previous one in the series.	The threshold-based version of this handler can become complicated when the change is measured with complex metrics, e.g., for images or other binary data, if one of the possibilities described in [12] is used. However, implementing these solutions for streams does not involve any particular new challenges until new, IoT-specific metrics are investigated.
	Threshold-based	Forwards items for which the difference from the previous one in the series exceeds a threshold.	

also for general-purpose handlers. Thus, Table I describes the handlers that have been developed in the NECtar agent, summarizing which of their respective data reduction algorithms include novel elements. For most algorithms, the trick lies in making them capable of handling data streams (i.e., real-time time series). From now on, this is referred to as "streamification" of the algorithms. Next subsection explains streamification in more detail by focusing on variations of the "Important Points Handler".

D. "Streamification" of data reduction algorithms

This subsection explains the idea of streamification by focusing on the algorithm used by the distance-based variant of the Important Points Handler.

The concept of measuring importance based on the distance to already selected Perceptually Important Points (PIP) stems from related approaches, e.g., [5]. Therefore, the following description focuses more on the parts that have been conceived for the efficient handling of the "per-item", "real-time" aspect of the solution, namely the "**cache projection**" and "**cache reduction**" methods. At the same time, the presented algorithm helps to understand better a core idea of the NECtar agent, namely the idea of having a uniform *handleData()* interface method which must be implemented by every data handler. With that, switching between different data handling solutions for an incoming stream can be achieved "with a single click", because it is possible to have a watcher in the NECtar agent, which sees a "change request" in a policy file and the only thing it has to do is replace the current handler object with an object that a different implementation of the handleData() method.

The data handling logic of the "Important Points Handler" is shown as pseudocode in Algorithm 1. Note that this is the method which is called for every incoming item of the data series as soon as this item is retrieved by the gateway application. First, the information about the new data item is gathered in order to create a storable data item. This item is unconditionally stored into the cache like every other incoming item. Then, a copy of the cache is made[1] and it is "projected"

[1]This copy can, of course, be maintained, in order to avoid creating it from scratch at each invocation of the handleData method. However, this optimization is hidden from the pseudocode for the sake of simplicity.

to the future with one of three possible **strategies**:

- by appending a copy of the current item (CLONE),
- by appending a duplicate of the entire cache (TWIN),
- by appending an item with an average value (AVG).

These strategies correspond with different notions of the importance of the future and the past (e.g., the CLONE strategy does not assume that patterns will repeat or does not care much if they will) and can lead to different results (as will be shown in the evaluation). As soon as this **cache projection** is in place, the items of the projected cache are sorted in descending order of importance according to a PIP ordering process, and the position of the current item in the sorted list determines if it will be forwarded or not. The following steps take place in the following order:

- The first and the last item of the projection are selected as the first two PIPs.
- The line that connects them is drawn and the item with the biggest distance to this line will be the third PIP.
- The item with the biggest distance to the line that connects its two adjacent PIPs will be the fourth PIP and so on.
- If the current item is in the top $X\%$ of the sorted PIPs list, then it is forwarded to the backend. X is a configurable parameter of the Important Points Handler.

However, such computations (for identifying important points) are traditionally performed only once, i.e., upon a complete data set. It can be very resource intensive to perform them for each incoming item. This is handled by maintaining the size of the cache at a level that does not cause significant delays in the pre-processing and forwarding of items (last lines of the pseudocode). It is challenging to find a cache size that can give good results and avoid delays at the same time. Therefore, the **cache reduction** is performed in order to reduce the number of points analyzed in each step in a way that does not harm the quality of the analysis but makes the delay insignificant compared to the transmission delay. Our evaluation will give some further insights into this aspect. Finally, note that this cache reduction does not actually mean deleting items from the cache of the network edge device, but simply reducing the size of the cache that is considered by the data handling method (i.e., kept in the "cache" variable of the pseudocode).

Algorithm 1. Pseudocode of the algorithm of the handleData method of the distance-based Important Points Handler

```
DEFINITIONS

tsName:        The time series name, provided before calling handleData
cache:         The cache, initialized upon instantiation of the data handler
projStrategy   The cache projection strategy (CLONE, TWIN, or AVG)
threshold:     Configuration parameter (between 0.0 and 1.0),
               which implicitly enforces the filtering ratio
-----------------------------------------------------------------
// Inputs: value (a value of the time series that has just been received)
//         timestamp (the timestamp for this value, captured by the GW application)
// Output: - (none)

currentItem.name = tsName;
currentItem.value = value;
currentItem.timestamp = timestamp;

cache.add(currItem);

// Create cache projection (NOTE: Maintaining a copy of the cache can be optimized.
      This is skipped here for easier understanding)
for each cacheItem in cache) {
   cacheProjection.add(cacheItem);
         sum = sum + cacheItem.getValue;
}
currItemCopy = currItem;
if (projStrategy == CLONE) {
         cacheProjection.add(currItemCopy);
} else if (projStrategy == TWIN) {
   for (i = 1 : cache.size) {
      cacheProjection.add(cache[i]);
   }
} else if (projStrategy == AVG) {
   currItemCopy.setValue(sum/cache.size);
   cacheProjection.add(currItemCopy);
}

order = [] // list that will store the indices of the cached items in descending
       order of item importance

// Now the first and the last item of cacheProjection can be included as PIPs
order.add(0);
order.add(cacheProjection.size-1);

// iterate as many times as the (rest of the) items that need to be ordered
for (int j=1; j<cacheProjection.size-1; j++) {
   selection = j; // the item currently selected to become the next PIP
   // iterate over all items (ignoring the already selected ones) in order to find
         the one with the maximum distance to its adjacent PIPs
   for (int i=1; i<cacheProjection.size-1; i++) {
      if (order.contains(i)) continue; // item has already been selected as a PIP
            previously. Ignore it.
      tmpItem = cacheProjection[i];
      // get the line to which the distance of the currently examined item should be
            measured, i.e., the line that connects the two closest PIPs between
            which the currently examined item is located
      currItemRelevantLine = getLineOfAdjacentPIPs(tmpItem); // details of this
            subroutine are omitted, functionality is obvious
      distance = currItemRelevantLine.getDistance(tmpItem); // details of this
            subroutine are omitted, functionality is obvious
      if (distance > maxDistance) {
         maxDistance = distance;
         selection = i;
      }
   }
   // add the item with the maximum distance to its adjacent PIPs to the already
         selected ones, thus making it the next PIP
   order.add(selection);
}

currItemNormalizedImportance = 1.0 - itemPositionInOrder/order.size;

if (currItemNormalizedImportance > threshold) {
   sendToCloud(currItem);
}

if (cache.size == cache.maxSize) {
   cache.remove(0);
}

return;
```

IV. Evaluation

A. Scope and setup

This evaluation focuses on two main aspects, which are related to core features that differentiate the NECtar agent from the state of the art.

- Firstly, it is investigated to which extent the NECtar agent mechanisms achieve to eliminate the data forwarding delays caused by the extra pre-processing involved in "streamified' data reduction algorithms.
- Secondly, it is investigated how well our "streamified" data reduction algorithms perform in terms of accuracy of the forwarded data compared to the respective "a posteriori" solution, i.e., the one that performs the reduction upon the complete data set (note: "accuracy" will be defined in the following).

Evaluated approaches: Stream-capable solutions of the Important Points Handler of NECtar agent, namely *NECtar-IPH-Clone*, *NECtar-IPH-Twin*, and *NECtar-IPH-Avg*[2], are compared with *PIP-post* (the "a posteriori" version of the important points-based solution, described in [5]).

Metrics: Firstly, the *delay* (in milliseconds) caused by the pre-processing at the network edge is measured and presented

[2]Corresponding with the three strategies of the Important Points Handler described in the previous section

for the different approaches. This delay must be evaluated in terms of its significance compared to the network and server-side delays for forwarding the data items to the Cloud. These delays (i.e., the time needed for sending a data item to a server, storing it into a database, and confirming the action) would be common for all approaches and they are usually in the order of a few hundreds of milliseconds (for Internet-based connections). Then, the achieved *data reduction* (in number of items and as percentage of the incoming items) has been regulated to be similar for all approaches (although PIP-post performs the reduction at the backend), so that we could focus on comparing the *accuracies* of the algorithms. Accuracy is defined as the similarity of the forwarded data set to the original time series, after interpolation has been applied on the forwarded data set in order to reconstruct a set that has the dimension of the original one. More concretely:

If $TS_o = [(t_{o_1}, v_{o_1}), (t_{o_2}, v_{o_2}), ..., (t_{o_n}, v_{o_n})]$ is the original time series and $TS_f = [(t_{f_1}, v_{f_1}), (t_{f_2}, v_{f_2}), ..., (t_{f_m}, v_{f_m})]$ is the forwarded time series ($m < n$, $TS_f \in TS_o$), then the reconstructed time series is $TS_r = [(t_{o_1}, v_{r_1}), (t_{o_2}, v_{r_2}), ..., (t_{o_n}, v_{r_n})]$, for which:

$$(t_{o_x}, v_{r_x}) = \begin{cases} (t_{o_x}, v_{o_x}), & \text{if } t_{o_x} \in [t_{f_1}, t_{f_2}, ..., t_{f_m}] \\ (t_{o_x}, v'_{o_x}), & \text{otherwise, where } v'_{o_x} \text{ is given by: } (t_{o_x}, v'_{o_x}) \textit{ is on the line that connects its two adjacent items} \end{cases}$$

Then, accuracy a is defined based on the Jaccard coefficient as:

$$a = \frac{\sum_{i=1}^{n} \min(v_{o_i}, v_{r_i})}{\sum_{i=1}^{n} \max(v_{o_i}, v_{r_i})} * 100\%$$

Datasets: The used datasets are publicly available real periodic measurements of three different kinds of IoT data sources. Time Series 1 (TS1) stems from a highway ramp loop sensor that measures the number of cars passing[3], Time Series 2 (TS2) stems from a home-installed smart meter that measures power, and Time Series 3 (TS3) stems from the sonar of a robot. All of the above are available through [2].

Environment and measurement details: Data reduction and accuracy depend only on the data sets and therefore no repetitions of the experiments are needed. Delay measurements have been repeated 1000 times and only averages are presented. The variations were insignificant and are omitted for the sake of simplicity, but also because the delays are not strictly compared to each other, but rather evaluated in terms of their relative significance when added upon the standard (network- and server-side) delay. Provided that the evaluated algorithms are designed for running at the network edge (i.e., on IoT gateways, edge routers, or mini-servers that are deployed to control an IoT area, e.g., a factory or a street), the experiments were run on a Virtual Machine (VM) which simulated the capabilities of a lightweight Gateway (700 MHz CPU / 512 MB RAM, e.g., Raspberry Pi), a modern residential or M2M Gateway (1.5 GHz CPU / 1.5 GB RAM), and a commodity PC (2.6 GHz CPU / 8 GB RAM). The simulation was done by using the CPU execution cap and the RAM regulation features of the Virtual Box VM execution environment.

[3]These loop sensor measurements were obtained from the Freeway Performance Measurement System (PeMS, http://pems.dot.ca.gov/)

B. Results and discussion

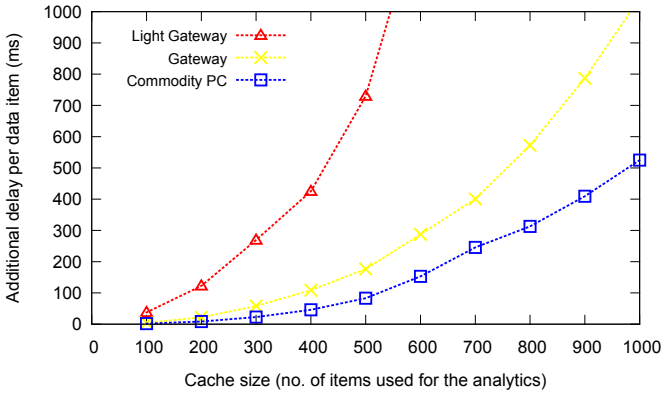

Fig. 3. Additional delays caused by the pre-processing of the Important Points handlers of the NECtar agent

The additional delays caused by the pre-processing at the network edge (before the forwarding or filtering) are shown in Figure 3 for various sizes of the (reduced) cache, i.e., by varying the amount of (previous) items that are considered in the data analytics process of computing the important points. Note that, even if optimizations for this specific algorithm are possible, the results are indicative for any algorithm that has a similar complexity such as this one, i.e., a complexity that rises sub-exponentially with the number of items considered in the analytics (no matter if these analytics are for finding important points or for doing anything that involves sorting, comparing etc.).

What we actually want to learn from the measurements of Figure 3 is:

- If we can use cache reduction to make the pre-processing delay small enough compared to the standard end-to-end delay of sending the item to the back-end, without getting down to too big cache reductions, i.e., reductions that severely affect the quality of the algorithm.
- Up to what cache sizes we can go on different kinds of edge devices (light GW, standard GW, commodity PC) without adding big delays.

The results for cache sizes up to 400-500 items are very encouraging for standard GWs and commodity PCs, while they can be acceptable even for lightweight GWs if the applications have no strict real-time (or "freshness") requirements. To give some examples, the additional average delays for a cache size of 100 items are 1.8 ms (for commodity PC), 4.6 ms (for a standard GW), and 37.5 ms (for a light GW), which are normally insignificant compared to the hundreds of milliseconds which will be needed anyway until the backend receives and stores an item that shall be provided to applications. With a cache size of 500, the measured times rise up to 83.2 ms, 176.3 ms, and 829.5 ms, respectively for PCs, GWs, and light GWs. These delays can be significant for applications with strict time requirements, showing that cache reduction can be

very useful in practical situations. It remains to be analyzed if the reduced cache sizes can still "do the job". This is discussed in the following.

Tables II - IV summarize the results of the algorithms for the three datasets, respectively, using a reduced cache size of 100 items. Table V summarizes the results of the algorithms for the robot sonar dataset using a reduced cache size of 500 items. The cache size of 100 items have been chosen because we have already shown in the delay measurements that no significant additional delays appear for this size. Note that the results of PIP-post show (in brackets) how many items are kept in the database after the backend (a posteriori) reduction, in addition to the items that have been forwarded (which, in the case of PIP-post, are always all incoming items).

The first thing that can be noted in the results is that the NECtar algorithms achieve accuracies in the range 76,1 % - 93,8 %, although they have always been configured to forward approximately one third of the original data, namely ca. 33 % - 35 % in most of the cases. Although the achieved accuracies are by far not as high as the ones that are achieved if the reduction is performed later at the backend (PIP-post accuracy ranges between 91,5 % and 99,6 %), it might be good enough depending on the application, because the desired accuracy depends on the application indeed. Finally, the accuracy of PIP-post is provided only in order to give a feeling and cannot be compared to the accuracy of other algorithms. The algorithms are not comparable, because PIP-post does not filter any data at all.

Another key observation is that the three different cache projection strategies of the streamified Important Points Handler (CLONE, TWIN, and AVG) lead to different results depending on the dataset. TWIN seems to deliver the best results thanks to its assumption of "long-term repetition of history", but the other strategies might also make sense for different datasets. Note that the TWIN strategy is not using a bigger cache than the others, because the size of 100 items is counted together with the "projected" items. This is a very important observation because it proves that the conception and development of the different projection strategies makes sense and provides the operator with a variety of choices that can prove very beneficial in terms of accuracy.

Finally, Table V (cache size = 500) shows that different accuracies might be achieved indeed by fine-tuning the cache reduction. In this case, increasing the cache size brought enhancements in the range of +0,4 % to +6,5 % (comparing Table V with Table IV). Thus, if the additional delays caused by the increased cache size (refer to Figure 3) are not critical for the application, then this technique can be considered. However, the accuracy enhancements do not seem to be very high and it becomes clear that no matter how big caches sizes are used, the accuracies cannot get close to the accuracies that can be achieved with "knowledge of the future" (PIP-post). Similar experiments, i.e., with a cache size of 500, have been performed for TS1 and TS2, as well, showing even smaller accuracy enhancements. It must be clear that bigger cache sizes do not necessarily always mean significant enhancements in accuracy. The reader should simply think of the extreme case of a dataset where all values are almost equal. All algorithms would give ca. 100% accuracy no matter the cache size.

TABLE II. COMPARISON OF NECTAR DATA HANDLERS FOR TS1 (HIGHWAY RAMP LOOP SENSOR) USING A REDUCED CACHE OF 100 ITEMS

Time Series 1 - loop sensor				
Data Handler	Incoming items	Forwarded items		Accuracy
		Amount (No.)	% of incoming	
PIP-post	41851	41851 (14689)	100 % (35.1 %)	91.5 %
NECtar-IPH-Clone	41851	13044	31.2 %	78.7 %
NECtar-IPH-Twin	41851	14384	34.4 %	85.7 %
NECtar-IPH-Avg	41851	14391	34.4 %	81.8 %

TABLE III. COMPARISON OF NECTAR DATA HANDLERS FOR TS2 (HOUSEHOLD SMART METER) USING A REDUCED CACHE OF 100 ITEMS

Time Series 2 - smart meter				
Data Handler	Incoming items	Forwarded items		Accuracy
		Amount (No.)	% of incoming	
PIP-post	44642	44642 (16813)	100 % (37.7 %)	99.6 %
NECtar-IPH-Clone	44642	15038	33.7 %	93.8 %
NECtar-IPH-Twin	44642	15199	34.0 %	92.9 %
NECtar-IPH-Avg	44642	16813	36.7 %	80.1 %

TABLE IV. COMPARISON OF NECTAR DATA HANDLERS FOR TS3 (ROBOT SONAR) USING A REDUCED CACHE OF 100 ITEMS

Time Series 3 - robot sonar				
Data Handler	Incoming items	Forwarded items		Accuracy
		Amount (No.)	% of incoming	
PIP-post	2324	2324 (802)	100 % (34.5 %)	99.3 %
NECtar-IPH-Clone	2324	830	35.7 %	75.4 %
NECtar-IPH-Twin	2324	786	33.8 %	84.3 %
NECtar-IPH-Avg	2324	781	33.6 %	76.1 %

TABLE V. COMPARISON OF NECTAR DATA HANDLERS FOR TS3 (ROBOT SONAR) USING A REDUCED CACHE OF 500 ITEMS

Time Series 3 - robot sonar				
Data Handler	Incoming items	Forwarded items		Accuracy
		Amount (No.)	% of incoming	
PIP-post	2324 2324	2324 (802)	100 % (34.5 %)	99.3 %
NECtar-IPH-Clone	2324	815	35.1 %	81.9 %
NECtar-IPH-Twin	2324	787	33.8 %	84.7 %
NECtar-IPH-Avg	2324	833	35.8 %	77.6 %

C. Lessons Learned

Based on the above, the main achievements and limitations shown by our evaluation can be summarized as in the following.

Achievements:

- Algorithms that are based on the PIP concept can now be applied for streams and forward data that have accuracies much higher than the forwarding ratio.
- The newly developed cache projection strategies can lead to different results for different datasets, thus providing a further tool for fine-tuning and enhancing the accuracy of the forwarded data depending on the application.
- Reduced cache sizes can help to make the additional forwarding delay insignificant without reaching so small sizes that would seriously affect the accuracy of the forwarded data.

Limitations:

- Independently of the used cache sizes, the accuracy of the forwarded data can usually not get close to the accuracy of a posteriori data reduction.
- The required accuracies depend on the application so that our algorithms would be unusable in certain domains.

V. Conclusion

This paper has presented the NECtar agent, a solution for network-edge data reduction (for time series data, focusing on IoT data), which has the following three main differentiators from the state-of-the-art:

- Streamification: it includes new data reduction algorithms that work upon data streams, i.e., per incoming item, based on concepts of solutions that were until today designed to compress a posteriori, i.e., upon entire data sets.
- One-click data handler instantiation: Network-edge devices can select one of many possible algorithms and simply instantiate a handler that enforces the respective data reduction logic.
- IoT-specific analysis: To the best of our knowledge, this is the first time that network-edge data reduction is evaluated specifically for IoT devices (cf. time measurements in Section IV) and IoT datasets (cf. accuracy measurements in Section IV).

The "streamification" of algorithms that are based on identifying PIPs (Perceptually Important Points) has been evaluated upon time series collected from three real IoT testbeds (street sensor, smart meter, robot sonar), showing that the NECtar agent can achieve accuracies between 76,1 % and 93,8 % despite forwarding only 1/3 of the data items, and thus without adding significant forwarding delays, because the pre-processing delays at the edge can be orders of magnitude smaller than the standard delay of sending and writing the data into the Cloud databases. The low delays are achieved both due to the lightweightedness of the data handlers and due to the appropriate application of the cache projection and cache reduction techniques. The streamification of further data reduction algorithms and the customization of the algorithms for specific IoT Use Cases are important items for further investigation.

References

[1] K. Åström. On the Choice of Sampling Rates in Parametric Identification of Time Series. *Information Sciences*, 1(3):273 – 278, 1969.

[2] K. Bache and M. Lichman. UCI machine learning repository, 2014. http://archive.ics.uci.edu/ml.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16. ACM, 2012.

[4] D. Boswarthik, O. Elloumi, and O. Hersent. *M2M Communications: A Systems Approach*. John Wiley & Sons Ltd, UK, 2012.

[5] F. Chung, T. Fu, R. Luk, and V. Ng. Flexible time series pattern matching based on Perceptually Important Points. In *International Joint Conference on Artificial Intelligence, Workshop on Learning from Temporal and Spatial Data*, pages 1–7, 2001.

[6] Cisco. Krikkit open source software, February 2014. http://eclipse.org/proposals/technology.krikkit/.

[7] Ericsson. More than 50 Billion Connected Devices. *Ericsson White Paper*, (284 23-3149):1–12, 2011.

[8] Google Cloud Platform. Google Cloud Storage: Pricing and Support, 2014. Last accessed November 2014, https://cloud.google.com/storage/pricing.

[9] N. Matsuda, H. Sato, S. Koseki, and N. Nagai. Development of the M2M Service Platform. *NEC Technical Journal, Special Issue on the Network of Things*, 6(4):19—23, 2011.

[10] A. Papageorgiou, M. Schmidt, J. Song, and N. Kami. Efficient Filtering Processes for Machine-to-Machine Data Based on Automation Modules and Data-agnostic Algorithms. *International Journal of Business Process Integration and Management*, 7(1):73 – 86, 2014.

[11] K. B. Pratt and E. Fink. Search for Patterns in Compressed Time Series. *International Journal of Image and Graphics*, 02(01):89–106, 2002.

[12] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image Change Detection Algorithms: A Systematic Survey. *Transactions on Imgage Processing*, 14(3):294–307, Mar. 2005.

[13] Q. Shen, L. Huang, G. Zhang, and J. Gong. Policy Control and Traffic Aggregation for M2M Services in Mobile Networks. In *International Conference on Mechatronic Sciences, Electric Engineering, and Computer (MEC '13)*, pages 3391–3395, Dec 2013.

[14] P. Wan and P. Man. Efficient and Robust Feature Extraction and Pattern Matching of Time Series by a Lattice Structure. In *International Conference on Information and Knowledge Management*, CIKM '01, pages 271–278. ACM, 2001.

[15] B.-K. Yi and C. Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 385–394. Morgan Kaufmann Publishers Inc., 2000.

[16] S. Zöller, A. Reinhardt, S. Schulte, and R. Steinmetz. Scoresheet-based Event Relevance Determination for Energy Efficiency in Wireless Sensor Networks. In *IEEE Conference on Local Computer Networks (LCN)*, pages 207–210. EDAS Conference Services, 2011.

[17] H. Zou, Y. Yu, W. Tang, and H.-W. M. Chen. FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with IO Performance Improvement. *Big Data Research Journal*, 1(0):4 – 13, 2014. Special Issue on Scalable Computing for Big Data.