CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Continuous Distributed Tracking of Large CPS and IoT Sensor Networks : A Top-k Related Problem

Master's thesis in Computer science and engineering

Colin Owusu Adomako
Silav Ahmed

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Continuous Distributed Tracking of Large CPS and IoT Sensor Networks : A Top-k Related Problem

Colin Owusu Adomako
Silav Ahmed

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Continuous Distributed Tracking of Large CPS And IoT Sensor Networks : A Top-k Related Problem

Colin Owusu Adomako
Silav Ahmed

Supervisor: Romaric Duvignau, Department of Computer Science and Engineering
Examiner: Marina Papatriantafilou, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Continuous Distributed Tracking of Large CPS And IoT Sensor Networks : A Top-k Related Problem

Colin Owusu Adomako
Silav Ahmed
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Continuous distributed tracking models can serve as a vital building block for constructing large-scale continuous data and event tracking applications, such as load-balancing, fleet management, IP traffic monitoring (DDoS attacks), network anomaly detection (Internet worms), sensor tracking and control, and grid resource tracking. Due to the widespread use of this model in emerging technologies or applications, it has attracted significant attention in recent times. At the core of these applications is a distributed tracking system that aggregates information and performs continuous tracking of values over collections of physically-distributed and rapidly-updating data streams.

A continuous distributed tracking of IoT sensors, implies deploying a number of sensors remotely to continuously read values in time steps or rounds. Values read by these remote sensors (eg. temperature, speed, distance, etc.) are sent to a designated coordinator node to determine the maximum or minimum value(s) at a time step (not within a sliding window) using a specific function. The maximum or minimum value(s) are known as the top-k values in the field of distributed tracking.

We consider the maximum top-k value in this thesis. We also take different approaches to determine the top-k values by implementing and simulating various existing top-k algorithms (naive-polling, basic-exact, simple-ε-approximation, adaptive filters-δ precision and online top-k position) on real datasets to determine their efficiency in terms of communication, time complexities and the accuracy with which they determine the top-k value(s). Strengths of each of the algorithms are selected and used in the implementation of our own adaptation algorithm.

We have thoroughly explored the trade-off between communication complexity and accuracy of top-k algorithms on 4 large datasets of sensor data. Based on our experimentation, we have identified the precise cost in communication for increasing the accuracy of the monitoring. Our results are particularly useful to the practitioners willing to pick a top-k tracking method over a distributed set of sensors or CPS.

Keywords: top-k, tracking, distributed, continuous, monitoring, algorithm, communication, coordinator, read, receive.

Acknowledgements

We say a big thank you to the Creator for bringing us this far. Much appreciation and thanks goes to Romaric and Marina, our supervisor and examiner respectively for their immense contribution through constructive feed backs and directions. We also thank all our lecturers and lab partners whose work ethics and dedication have helped shaped us.

Lastly, we will like to thank all our family members and friends for their support and prayers.

Colin Owusu Adomako and Silav Ahmed, Gothenburg, July 2024

Contents

List of Figures

List of Tables

1

Introduction

Recent years have seen a clear growth in the number of Internet of Things (IoT) devices [16] and cyber physical systems (CPS). IoT devices are being used as smart, reliable, and low-cost technologies that may be used for measurement tasks [14] to support most functionalities in CPS. Thus, most CPS and adopted IoT devices in recent times embed sensors used for different sensing categories, such as for safety, diagnostic, traffic, assistance, environment, IP traffic monitoring, to mention but a few. For example, the safety sensors are used for sensing and observing accident hazards and unusual traffic events whilst diagnostic sensors help monitor human health almost in real-time [22].

In order to achieve accuracy in these measurements or actuation for effective performance analysis, optimisation, and anomaly detection, there is a need to track the measurements by the sensors or nodes which are characterised by a high decentralisation level. In some circumstances, the sensor(s) with the highest measurements may be of interest whilst in other circumstances the one(s) with the lowest may be of interest and thus need to be tracked. In such settings, sensors within these decentralised architectures communicate their measurements through networks to a designated coordinator for the highest or lowest measurements to be determined. This has been applied to achieve load balancing and can also be used to address network related security threats such as DDoS, WiFi hijacking, eavesdropping, etc. It must be emphasised that, although it is too costly to centralise all these measurements and computations, it is quite challenging to also design solutions which provide an exact or good approximation to the current measurement, while optimising the communication cost of the decentralised nodes. [12].

Notwithstanding the challenges and difficulties, various research such as [12, 24, 10] has attempted to tackle this particular problem which has been identified as top-k in the field of continuous distributed tracking. This corresponds to tracking every node or sensor receiving data continuously from an input or data stream that is only recognised by the respective node. On the other hand, they could be observing a specific function whose value is changing over time [24]. To the best of our knowledge, although several adaptations of these top-k tracking algorithms have been presented by these different research, there is no research work that seeks to evaluate these algorithms against each other in terms of communication and time complexities. A trade off exist between communication performance and accuracy in top-k computation. Thus, designing efficient top-k algorithms that aim at minimising the communication (overhead cost), i.e., the number of messages, between the designated coordinator and the distributed sensors whilst present exact

answers or values can be very challenging.

This thesis therefore seeks to implement different top-k algorithms and if possible design our own adaptations and modifications as well as potentially new algorithms that will account for communication needed to track a large network of CPS and IoT sensors (nodes) based on the shortfalls from the existing ones.

We thus make the following contributions.

1. We provide a comprehensive survey on several top-k models and approaches.
2. We compare existing top-k algorithms from different research communities through extensive experiments with a variety of real datasets obtained from different sources.
3. We report comprehensive findings obtained from the experiments. We provide new insights into the strengths and weaknesses of existing algorithms that can guide practitioners to select appropriate algorithms for various scenarios.
4. We implement and evaluate our own adaptation or modification of the top-k algorithm, taking into consideration the strengths and weaknesses of the existing ones.

1.1 Context

Emerging CPS technologies such as the development of smart cities, autonomous vehicles, medical diagnosis equipment, electricity and water distribution systems, etc require the use of a large network of IoT sensors. These sensors produce numerous data streams from multiple remote sources and must be transmitted to a central processing system where tracking takes place [30]. The data arrives at a very fast pace [23], sometimes as fast as several gigabytes a second, and therefore requires real-time processing of data stream. Due to this, the stream is modelled as a continuous or unbounded stream [44] using Continuous Distributed Tracking models such as top-k tracking. Examples of important data stream applications of frequent item analysis or top-k tracking analysis include the following:

- *Environmental Data App/API*: Wireless sensor networks (WSNs) with stationery sensors are deployed to continuously collect environmental data such as humidity, temperature, wind speed, rainfall pattern, etc, to better understand environmental changes [5].
- *Detection of Network Anomalies*: Certain network attacks follow some patterns or exhibit some characteristics. An example is how worms are detected by the frequent occurrence of substring patterns in traffic flows [21]. Another example is the detection of distributed denial of service (DDoS) attacks which have recently been established that identifying destination addresses that have received a large number of packets over a given time can be used to detect DDoS attacks [9, 3, 32].
- *Network Flow Management*: Large portions of bandwidth in a network are accounted for by few flows. To allocate bandwidth more fairly, knowledge of these flows is a prerequisite [34].

The central idea of tracking models is to incur minimal communication when there is nothing important being observed, but at the same time to enable rapid (near-

instantaneous) updates when necessary. Hence, the continual transmission of a large number of rapid data streams to a central location can be impractical or expensive [30]. Additionally, it leads to high overhead cost. These can be attributed to data streaming rates exceeding the capacity of the tracking infrastructure (data collection points, transmission lines, processing centre, etc) in terms of storage, communication, and processing resources [10].

Our work will be to analyze and evaluate various top-k tracking algorithms both existing and new ones. And also, implement an adaptation or modification of the existing top-k algorithm with the objective of optimising the communication needed to a large network of IoT sensors. Similar works have been done and our work seeks to build on the recent one [24].

1.2 System Model

A single-level hierarchical architecture forms the basis of the distributed system environment[10] defined in this paper. It consists of a designated or central coordinator and n nodes or IoT sensors with n distributed data streams. These sensors $S_1, S_2,\ S_n$ are tasked to read data streams and are referred to as monitoring nodes. The responsibility of the coordinator is to continuously keep track of the monitoring node with the lowest or highest top-k value over the union of n distributed data streams. A notable feature of our model as depicted in Figure 1.1, is that there is no direct communication amongst the monitoring sensors or nodes though they can communicate with each other through the coordinator. Hence, communication is conducted between the monitoring nodes and the coordinator. The monitoring nodes communicate to the coordinator by sending the highest value read over a time period tp. The coordinator, on the other hand, communicates to the monitoring nodes by sending the current top-k value after comparing the values received from them.

The distributed monitoring sensors $S_1, S_2, ..., S_n$ are used to continuously read the corresponding data streams $m_1, m_2, ..., m_n$. These data streams consist of a sequence of values ordered according to time of occurrence such that v_1^1 is the value read in time step 1, v_1^2 in time step 2 and v_1^3 in period 3 by Sensor S_1. These values are generally presented in a series of tuples and each tuple takes the form $<v_i, t_i>$, where v_i is the value being read by Sensor i, and t_i represents the timestamp of the tuple. On the other hand, in some datasets, t_i represents the round number of the tuple. It must be emphasised that, the value may be repeated any number of times in a data stream. An example of a data stream, corresponding to the monitoring node S_1, may be represented as $m_1 = \{<2, 0.024>, <2, 0.029>, <1, 0.050>, <0, 0.056>\}$ where $m_1 = \{0, 1, 2, 3\}$. If a sensor does not read a new value at time t_i, the coordinator will use the previously communicated value at time t_{i-1}. Also, if it reads several values within a single time period, an average of the values be taken. An average is taken into account because sensors are susceptible to read outliers and an average gives a fair representation. For example, if a time period $t_p = 1$ second and values $2.2, 2.7, 2.5$ are read within a second, then an average of these values which is 2.5 will be sent to the coordinator.

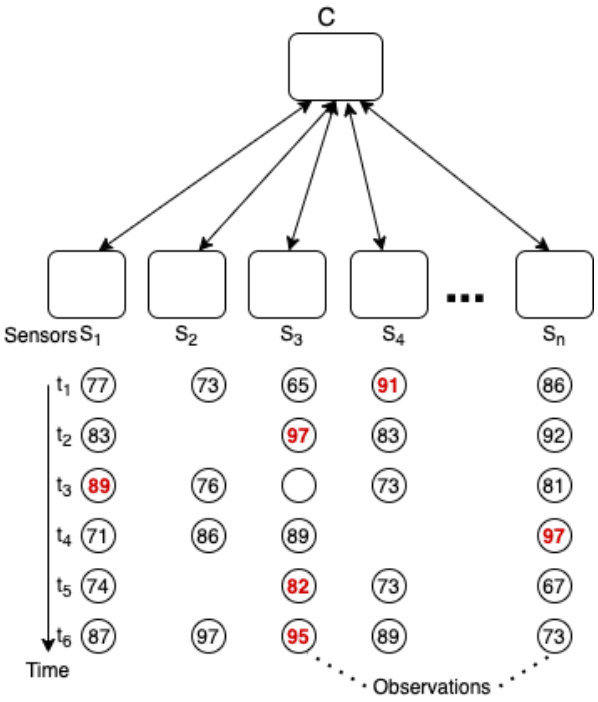

Figure 1.1: Single-Level Hierarchical Architecture.

As stated earlier, the coordinator in some cases communicate with the monitoring sensors by sending the top-k value(s) at a time step (by way of the broadcast if they are on the same network) whilst the distributed sensors communicate to the coordinator by sending the values read at a time step or round as demonstrated in Figure 1.1. It is also seen in this figure that, in some time steps there is a single reading by the sensor. In such a situation, the single value is taken into account. In certain time steps, several values are read by a sensor and an average of these values will be computed in this instance. Furthermore, if no value is read as it is in time period t_2 by sensor S_2, the previous value read in t_1 will be taken into account. In the figure, the top-k value in each time step is inscribed in red.

1.3 Goals And Challenges

This thesis consists of three parts, the first part aims to implement existing top-k algorithms currently being used to track a large network of IoT sensors. The second part will consist of implementing and designing a modification of a continuous distributed tracking (top-k) algorithm that will help reduce or optimise communication needed to track a large network of IoT sensors. Finally, we will design a set of experiments and compare all implemented algorithms to the existing in terms of performance, that is communication, top-k computation accuracy (correctness) and time complexity.

Below is the summary of what this research work seeks to achieve:

1. Implement different top-k algorithms for tracking a large network of IoT sensors on sample datasets, more precisely, the online algorithm introduced in

[24] and the heuristic generic algorithm of [10].

2. Implement our own adaptations, modifications, etc, of top-k algorithms based on the ideas from the existing ones and evaluate them on the same datasets.
3. Design a set of experiments and compare our algorithm to the existing ones in terms of performance (communication, top-k computation and time complexity).

The only challenge encountered in this work was the difficulty in coming up with the right design choices which will enable all our implementations accept the four dataset files without modifications.

1.4 Structure Of The Report

This report is organised into six main chapters: Chapter 1 is the introduction which presents a general overview of the entire thesis. This includes the goal, context, system model, and related works of this thesis.

In Chapter 2, we present the background where we perform a literature investigation in continuous distributed tracking systems. We elaborate on the state of the art models in CDT and state of the art top-k algorithms.

The algorithms under investigation in this report are presented in Chapter 3. In this section, we describe some primary top-k algorithms and also implement our adaptation algorithm.

The evaluation methodology forms Chapter 4 of this thesis. Over here we present experiments, datasets, implementation choices and how the efficiency of the existing algorithms will be evaluated in terms of performance (ie. communication and time complexities).

Results from the experiments are analysed and discussed in chapter 5. The chapter is subdivided into four with each presenting the results of a different experiment.

The final section, that is, Chapter 6, also draws a curtain to this research work by providing a conclusion for the entire thesis. In this chapter, we also outline future research directions.

2
Background

The purpose of this section is to familiarise the reader with the concept of continuous distributed tracking systems. It commences with a problem description, followed by a presentation of different models that can be used when developing such systems. We conclude this section by also presenting a literature review in this subject area.

2.1 Problem Description

In a distributed system, a set of sensors communicate directly to a coordinator C whose function is to continuously keep track of the sensors or locations where the highest k values are read or observed. This can incur a lot of overhead costs (performance) if the values change rapidly over time due to the frequent communication by the sensors to the coordinator about all the changes read somewhere in the system [10]. However, the most existing state of the art top-k tracking algorithm usually relies on the fact that values read at the different streams are expected to slowly change, stay the same or similar enough over time to reduce the needed communication in practice. Rapid data updates, with unique values, thus, requires an improved top-k tracking algorithm to efficiently track the values read by the distributed sensors whilst optimising the communication between these sensors and the coordinator C. [18].

We assume a set of n remotely distributed IoT sensors, identified more precisely with unique identifiers $\{S_1, S_2, S_3..., S_n\}$ which continuously read data stream $\{v_1, v_2, \ldots, v_n\}$ at time t_1, t_2, t_3. Data is read or observed exclusively and in a synchronised fashion by the respective sensors. It must be noted that, at time t_1, a sensor S_i receives $v_i^{t_1}$ and does not know in advance the value of any $v_i^{t_2}$, where $t_2 > t_1$, in other words, t_1 precedes t_2. The sensors can communicate to a single coordinator which keeps track of the sensor with the highest k value(s) but cannot communicate directly with each other except through the coordinator as depicted in Figure 1.1.

A practical example is a set of n IoT sensors, where $n = 17$, deployed remotely into the sea to track the highest velocity or temperature value and a single node as a coordinator to track the values being read by all the remote sensors at each time unit t (eg. where $t = 1$ second). And these sensors communicate their reading to C which is referred to as the coordinator which also keeps track of the node with the highest temperature value at each time unit. To decrease network cost and increase the sensor's battery life, the central idea of the tracking models used in such systems is to incur minimal communication when there is nothing important

being read or observed, but at the same time to enable rapid (near-instantaneous) updates when necessary. Indeed, the continual transmission of a large number of rapid data streams to a single coordinator can be impractical or expensive [30].

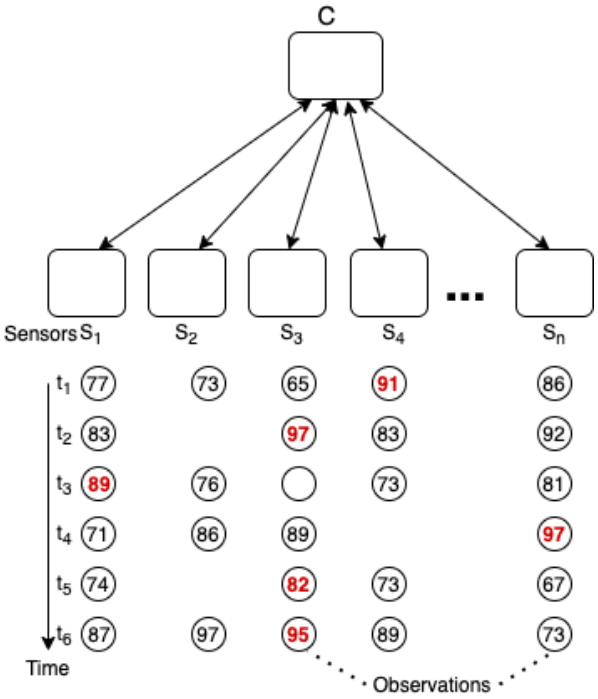

Figure 2.1: Tracking Of The Highest Value, An Example Of Top-k Tracking.

In Figure 2.1 above, there are n remotely distributed sensors communicating to a single coordinator about temperature values being read. The task of the coordinator is to keep track of the sensor with the highest k value at each time step t using a tracking function. The highest k value at each time step is highlighted in red. For example, it can be seen that sensor S_n holds the highest k value at time step t_1 whilst sensor S_3 holds it for time unit t_2 and so on. It must also be emphasised that, sensor S_2 read nothing at t_2 and therefore its reading for t_1 is used in the k computation at t_2

The top-k problem has many applications in most emerging technologies such as cloud computing, smart cities, autonomous vehicles, medical diagnosis equipment, electricity or water distribution infrastructure, IP networks, and other cyber physical systems which require the use of a large network of IoT sensors to observe data values. As stated earlier, these sensors read unbounded data streams at a rapid rate from multiple remote sources. The data must be transmitted to a central processing system C where tracking takes place. This leads to high communication complexities.

2.2 Distributed Monitoring Models

Various approaches or models have been deployed over the years in the field of distributed monitoring. In this section, we will outline state-of-the-art approaches

being used for this purpose.

2.2.1 Simple Approaches

The simplest state-of-art approach for continuous distributed monitoring is for each node to communicate all its observations to the coordinator. The shortfall of this approach is that it is not scalable with a large number of nodes and observations. To address the issue of scalability and to specify the optimisation's properties, polling and sampling approaches were introduced [12].

Polling relates to retrieving data at a constant rate from observers, and collecting this information together. However, since this approach waits for the latest poll to get a snapshot about the current situation, it loses some of the characteristic features of a continuous tracking model. A drawback with this approach is that it is not possible to predict how polling can work, when the goal is to measure some non linear functions for all distributed nodes or when trying to discover if some complex events have happened. Another drawback is that the tracking procedure is based on the frequency of polling. When an event takes place, a careful balance is required in setting the frequency of the polling event. When it sets a very narrow gap the network becomes overloaded. But setting a larger gap, the delay between occurring and detecting a significant event by the protocol may become too large as shown in Figure 2.2 [12], thus, may lead to a reduction in the accuracy [12].

On the other hand, sampling relates to retrieving data at a constant rate from observers or nodes whilst fetching data from a few nodes to reduce the number of measurements on the system. It must emphasised that with sampling, you can easily miss the top-k nodes, for instance, if you want the top-1% of the nodes. Therefore, polling and sampling inherently possess a number of drawbacks and considerable research has been invested in recent years to circumvent those drawbacks with more elaborate forms of monitoring, which are described hereafter.

2.2.2 Distributed Monitoring Approaches

The focus of these approaches is not on solving a function on every time step but to compute a function of all values read over a sliding window. They were developed to alleviate the drawbacks of the simple approaches. Two essential characteristic features make these approaches more efficient than the simple approach. Firstly, processing of data stream in done locally by each distributed monitoring nodes. Secondly, communication is not done each time step, but only when a set criteria is met or when results are requested. Additionally, whenever there is communication, only the processed data by the monitoring nodes are transmitted. This helps reduce the amount of communication needed and thus decrease processing time by use of the distributed nodes.

These distributed approaches can be categorised into two, namely one-time distributed approaches and continuous distributed approaches.

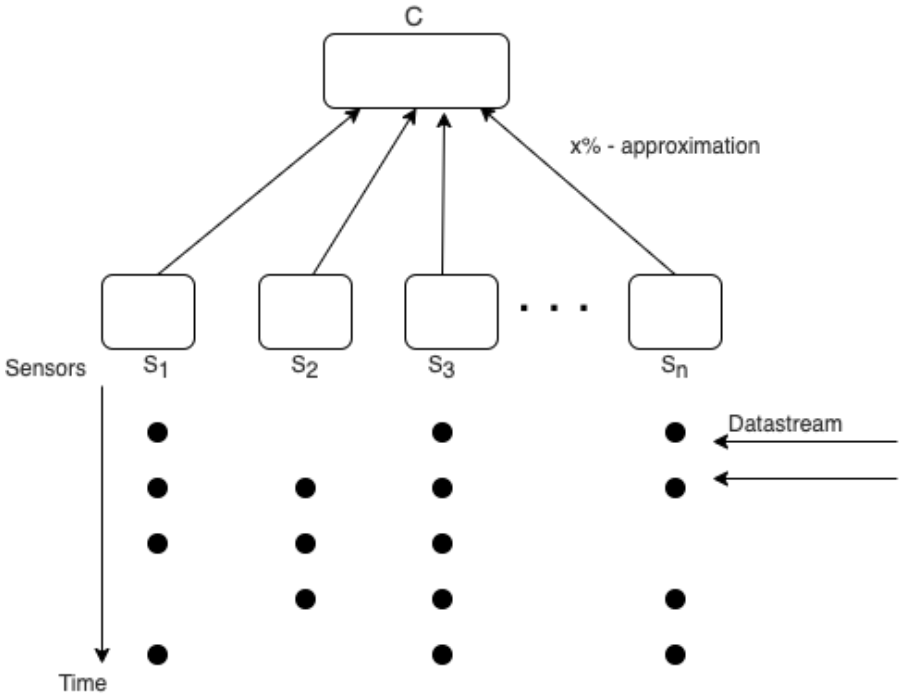

Figure 2.2: System Architecture Of A Distributed Monitoring Model

One-Time Distributed Monitoring Model

One-Time Distributed Monitoring Model (OTDM) is quite similar to polling but is made up of those methods that use one-time query approaches which only deliver results once a milestone is reached or at a point in time [10]. Although these methods are not designed to be continuous in their approach of tracking events, they can be repeated with short time intervals to simulate results [10]. Manjhi et al. [3] sought to tackle this problem of determining time-sensitive (recent) frequent items over a distributed data sources by installing an ε-approximate counting technique at each monitoring node and later transmitting local frequency counts over T time units to a centralised node. The received local counts are then joined with the previously received counts. By deprecating the previously received counts, more emphasis is placed on recent items occurrences in an exponentially decaying fashion.

Manjhi et al. [3] have shown that transmitting all frequency counts to a designated centralised node would yield an excessive communication cost. Also, a large number of monitoring nodes means the centralised node will be flooded with the amount of data received. A multi-level hierarchical communication architecture was proposed as a means of reducing the load on the centralised node. This is quite similar to the single hierarchical architecture that is going to be used in our work, except for the introduction of an additional node between the monitoring nodes and the designated centralised node [13]. This is described in Figure 2.3.

The introduction of the additional or intermediate nodes paved a way for the authors to find a point where frequency counts received from their child or monitoring nodes were cumulatively combined. This further introduced the concept of a pre-

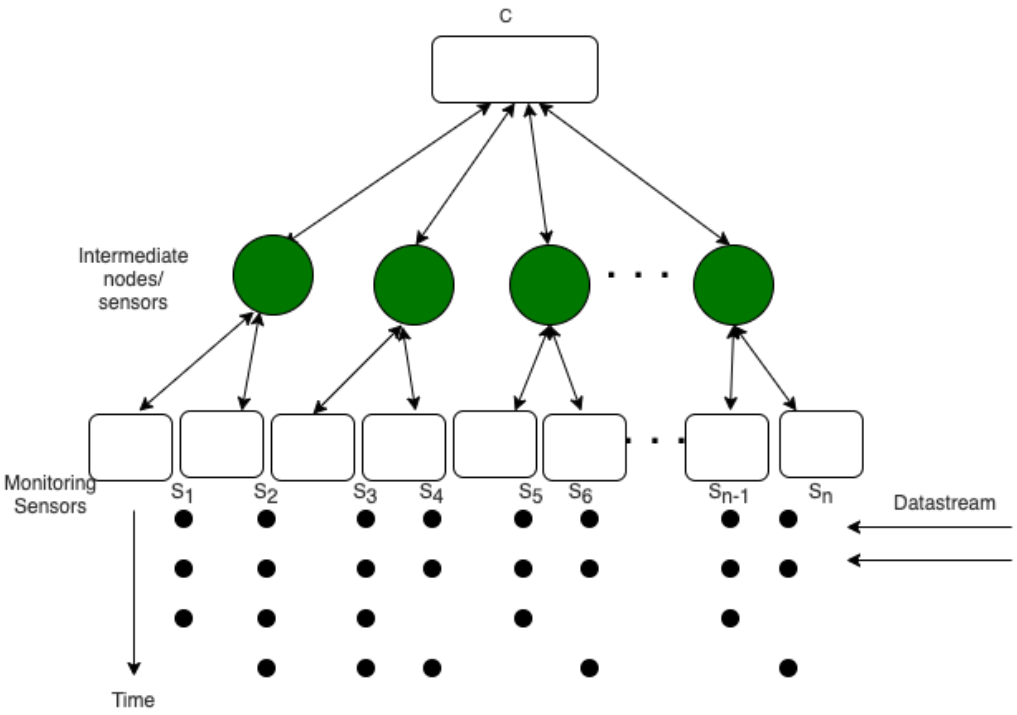

Figure 2.3: A Multi-Level Hierarchical Architecture Of A One-Time Distributed Model

cision gradient to reduce communication load on a single link. With this in place, the ε-approximate counting techniques installed on each leaf node counts the true frequency of an item. Communication is further reduced by dropping all zero values to avoid transmission to the centralised node. In addition, the degree of error tolerance was varied at each level of the node.

Continuous Distributed Monitoring Model

Continuous Distributed Monitoring (CDM) is a continuous monitoring in which an aggregate function is computed repeatedly over time and the results provide a view of the changes in the behaviour of a system for a certain time-frame. The key idea of this model, is to afford minimum and efficient communication between the nodes and the coordinator [19] when there is nothing important being read. It again seeks to empower rapid essential upgrades at the same time and also to take a particular decision based on the founded updates [12][18]. Each node acts as an observer in the system and transmits data about locally detected events (eg. pressure, temperature, humidity, etc) to a coordinator through a bidirectional channel. The coordinator then adds and computes the results using a monitoring function. Thus, in this model, there are n, remote nodes denoted as $S_1, S_2, ...S_n$ and a single coordinator also represented by C. Also, $A = (a_1, a_2, ...a_m)$ denotes a sequence of items where $a_i \subset [n]$ was received by one node at time t_i where $t_1 < t_2 < .... < t_m$. Hence, the monitoring function here is defined as $f : [N]^m \rightarrow \mathbb{R}$ with $A(t)$ representing the multi-set of data received from all nodes up until time t. It must be emphasised that, the goal of a monitoring algorithm that is expressed based on this model is

to approximate the value of A at time t $f(A(t))$ with a relative error of ϵ where 0 $< \epsilon < 1$, this is called the approximation model represented in [11] where it stores and remember the latest item the node shared with the coordinator and compare it with the new value observed, it transfers to the coordinator in case it lies ϵ far from the value of A [16].

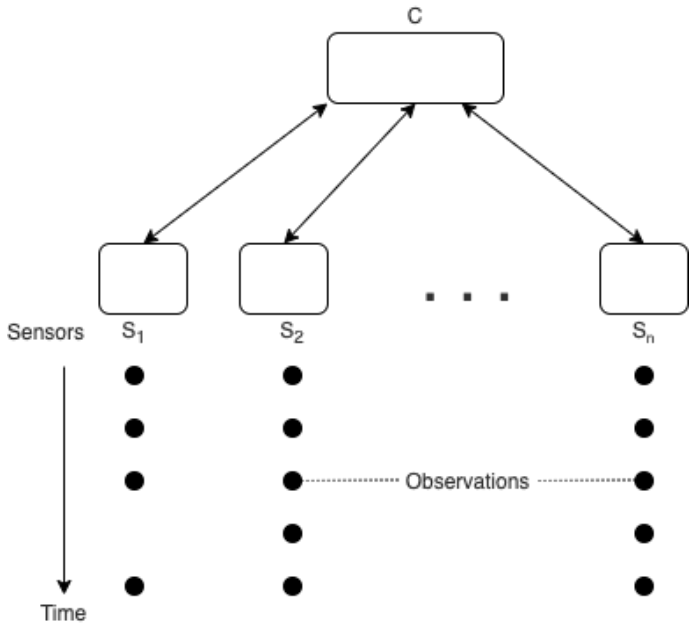

Figure 2.4: System Architecture For CDM Models.

The most prevalent problem in this model is counting the number of events occurring during some sliding window. Another common problem of these monitoring systems is calculating the frequency of items [13] and very popular items [41]. Some assumptions well noting in this model are that, a connection is only allowed to be initiated by a remote node upon arrival of data. Also, nodes do not communicate with one another about local results.

2.2.3 Distributed Data Streams Model

With Distributed Data Streams (DDS), monitoring distributed systems require very high communication overhead [33]. The DDS model finds ways to reduce the communication between the nodes and the coordinator [11]. It is a variant or an adaption of the Continuous Distributed Monitoring (CDM) model. The difference lies in the communication architecture, that is, whereas communication between nodes in this model is unidirectional as depicted in Figure 2.5, it is bidirectional in CDM. In other words, the coordinator in this model does not communicate or share the global state with the remote nodes, hence the nodes perform their local aggregation and computations based precedence to decide when to communicate to the coordinator about their local observations [20] Another notable difference is that, unlike the continuous monitoring model, distributed observers in the data streaming model do not compute a function of all their inputs aggregated together, but instead keep a sub-

linear amount of information to approximate the result of a monitoring function. On the other hand, a continuous distributed monitoring model does not require each observer to use sub-linear space; it treats space as a property of an algorithm used instead.

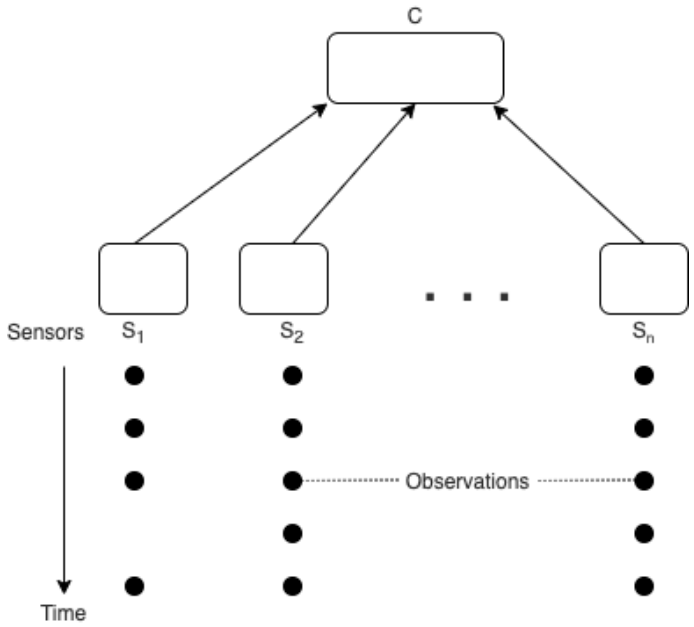

Figure 2.5: System Architecture Of A Distributed Data Streams Model

In this model, the data stream of each node is considered to be a sequence of items or values from an ordered U that defines the set of possible distinct items of the data stream can contain. All items in set U haves their corresponding arrival time-stamp based on a local clock. A data stream is represented by σ and we have $S_1, S_2, ..., S_n$ remote nodes (or observers). There is also a single designated coordinator S. For a given node S_i, its corresponding data stream is denoted by σ_i. Therefore, for a given stream σ, let c_j, σ and c_σ represent the count of item $j \in U$ and all items such that $c_j = \Sigma_\sigma c_{j,\sigma}$ and $c = \Sigma_\sigma$ represents the count of j and all items in all streams put together. It must be noted that specific and entire approximation counts for items are needed for various statistical computations.

Algorithms based on this model either applies to the whole stream or recent window of size w during statistical computations. The Count-based and time-based sliding window includes w items in each stream and items received in the last w time units respectively. This model differs from the continuous distributed tracking model in that, more than one item can be associated with a specific time t in the time-based sliding window. It must be emphasised that existing algorithms need to be adapted to suit sliding-window algorithms for the whole-stream case with respect to the space complexity as a result of the sliding-windows converting monotonic functions to non-monotonic functions because of "deletions". For instance, if we are interested in counting the frequency of a specific item in the case of the sliding window case, this is achieved by keeping a single counter. This is due to the fact that, any algorithm implemented based on this model, the space complexity and error guarantee bound

is of great importance since observers and the coordinator are assumed to not be able to hold all the received items in memory. Therefore, each site can only use sub-linear amount of space and maintains its statistics approximately. Hence, this will require a space of $\Theta(1/\varepsilon log^2(\varepsilon w))$bit for the sliding-window if ε of relative error is allowed [29].

2.2.4 Distributed Online Tracking Model

DOT is a system characterised by a general-tree structure. It is a recently introduced unidirectional [40] and synchronous model that features the possibility to have intermediary nodes between the coordinator and the remote nodes. This intermediary nodes function as relay nodes and are tasked with aggregating statistics locally before forwarding some data to the coordinator. The remote nodes lie at the leaves and the coordinator is always located at the root of the tree. The relay nodes do not observe a function that participates in the calculation directly, but it can receive messages from remote nodes and forwards them to the coordinator [37]. In the setting of [37], the cost of the communication is represented by the total amount of messages sent in a topology.

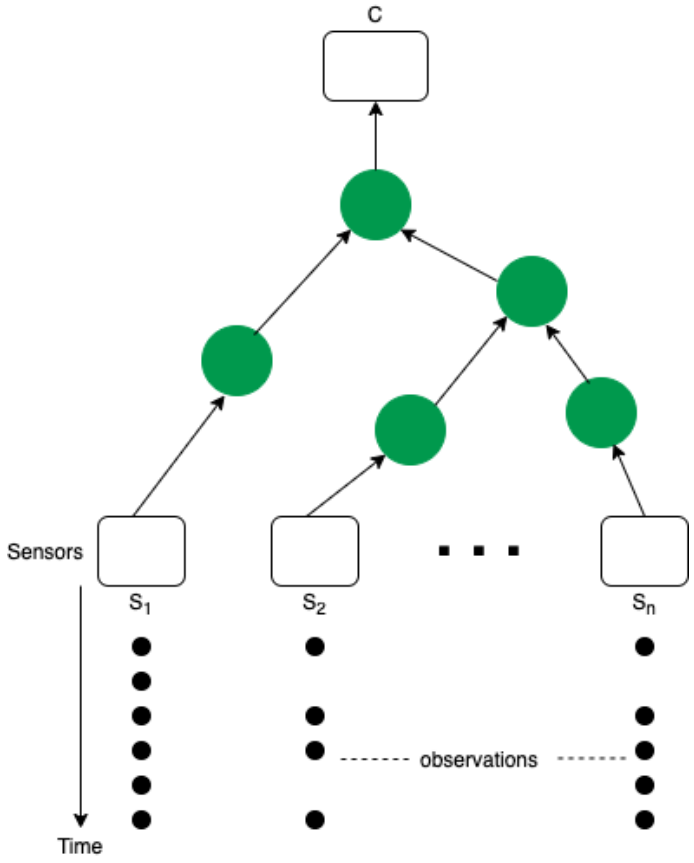

Figure 2.6: System Architecture Of A Distributed Online Tracking Model

The figure above provides a schematic view system being described. S_i represents the remote sites or node whilst the relay nodes are shown in coloured circles. It

must be emphasised that, any system with a general-tree is suited for DOT model. Directly connected remote nodes to the root nodes can also be a valid system in DOT just as in the DDS model. In such a case, a number of arbitrary relay nodes and sub-trees can be inserted into the tree as depicted. To better understand this, let there be n remote nodes denoted as $S_1, S_2, ...S_n$ and a designated coordinator. The aim of the coordinator is to compute a function f over t that is, $f(t)$ for all given values of t. The function f, takes a list of values as input with each value in the list being a representation of what is being read or observed by the remote node at time t. It then computes an aggregate result for t. To accurately compute $f(t)$ requires flooding the network with messages or values to be read by each monitoring node therefore the aim of the coordinator is to approximate $f(t)$ by computing $g(t)$ such that $g(t) \in [f(t) - \Delta, f(t) + \Delta]$ for any time step t and some user-guaranteed error threshold Δ.

Hence, if $f(t)$ and $f_i(t)$ denote a local function at node S_i and its value at time t respectively, then f at the coordinator node is a function that receives all the results from the local functions (i.e $f_i(t)$) as input for time t. Therefore, $f(t) = f(f_1(t), f_2(t), ..., f_n(t))$ for time t. And based on the choice of a monitoring function, f and f_i can be either one-dimensional or multi-dimensional. For example, let us consider an aggregate "max" function that determines the highest value read at time t since initialisation. Hence, for each node, $f_i(t)$ is the local highest or maximum value read at time t, and $f(t)$ is the highest value read in the entire system initialisation at time t.

2.2.5 General Heuristics And Other Dedicated Approaches

Heuristic approaches to distributed queries have been explored in parallel and analytical works. In these approaches, several adaptations and experiments are run on an existing model until the desired result or outcome is achieved. A classical example is the use of adaptive filters introduced in [28] for tracking heavy hitters for DDoS attacks for effective protective mechanisms to be put in place. With the adaptive filters, the authors introduced a natural “filter” approach, which assigns a local filter to each site so that if the current value is within the filter, it does not need to be reported. When a site’s value falls outside its filter, the current value is reported, and the filter is re-centered on this value. Over time, some filters can be widened and others narrowed so that the total uncertainty remains bounded, but more slack is allocated to values that are less stable. Other heuristic approaches were as outline in [43] are score-based algorithms and iterative reduce top-k algorithms.

The score-based algorithms assign each item m_i with a score and the item with the highest score is selected as the top-k. These algorithms include:

- **Election Algorithms.** This approach can be used to compute the score of an item. There are two well-known election algorithms, that is, *BordaCount* [8] and *Copland* [6].
- **Max Algorithms** Proposed by Guo et al.[7], it is a graph-based algorithm to compute the maximal item, which can be used to assign a score for each item.
- **Ranking Algorithms.** Most of these algorithms in this category focus on ranking documents or images.

The iterative reduce top-k algorithms work by iteratively eliminating lowly ranked items that have small possibilities in the top-k results, leaving only the highest scored k items. These algorithms also include:

- **Iterative.** This is an adaptation of the max algorithm by Guo et al.[7] to improve its quality. It first uses the score-based method to compute the scores of each item and then removes half of the items with small scores. It then recomputes the score for the remaining items and repeats the iteration until the k items are left.
- **PathRank.** This is used to perform a "reverse" depth-first search (DFS) for each node, which traverses the graph by visiting the in-neighbours of each node. When the path with a length larger than k is found, it eliminates the item as k items have already been better than the item. [17].

A shortfall of these heuristic approaches is that, they do not take into account the exact function that is being monitored. To determine the top-k items with this approach is also NP-Hard [7].

Essential modifications to theoretical models have been affirmed in previous attempts to make CDM algorithms feasible in practice as e.g. for sensor networks [35]. It is well noting that, a large amount of research has been invested in CDM algorithms and techniques to find and prove (matching) lower and upper bounds on the communication complexity. Notwithstanding, this research has not yet been replicated in large scale Cyber Physical Systems and IoT networks. This will thus, form the basis of our discussion in our thesis.

2.3 Exact and Top-k Tracking

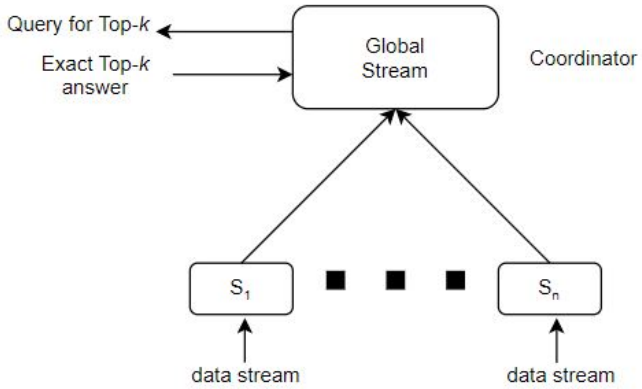

Figure 2.7: System Architecture Of An Exact Top-k Tracking

As the name depicts, exact top-k tracking is related to the designated coordinating sensor's ability to continuously present the exact top-k set of values. This is necessary in applications where correctness in the top-k values is mandatory or a requirement. That is, an algorithm with ε-approximation with $\varepsilon = 0$, the coordinating sensor must continuously report the exact top-k set [10]. For example, if

S_1, S_2, S_3, S_4, S_5 reads $3, 9, 1, 7, 5$ respectively at time t, then an exact top-k algorithm must present 9 as the top-1 value at time t.

2.4 Approximation Top-k Tracking

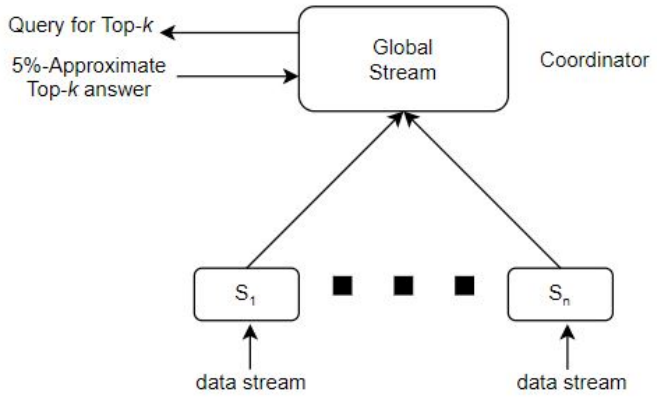

Figure 2.8: System Architecture Of An Approximation Algorithm

As afore-mentioned, since data streams are unbounded in nature, it becomes sometimes impossible to retrieve exact answers to top-k queries. Thus, when exact answers to top-k tracking queries are not mandatory or relevant and a controlled degree of error is acceptable (i.e. $\varepsilon > 0$), approximate top-k can be applied [10]. This has thus motivated the creation of numerous top-k tracking algorithms that adopts approximate top-k solutions at the expense of correctness [4]. It is worth noting that, these approximate top-k solutions require only a limited amount of memory and provide an approximation of the top-k values. Hence, to solve the problem of top-k value within approximate bounds, the ε-deficient or ε-approximation could serve as a stepping stone to achieve exact top-k as proposed in the work of [1].

The ε-approximate top-k problem allows a degree of error on the frequency counts which must fall with a range or be bounded by a user defined error of tolerance. An example is an algorithm by [27] that guarantees approximating tuple ranks to a factor $0 < \varepsilon < 1$. The more the value of ε is closer to 0, the more the approximation algorithm reduces to the exact algorithm. Another example is Threshold Algorithm (TA) [31] which also has approximate variants. It defines a parameter $\varepsilon > 1$ which represents the level of approximation, such that a value $x \notin (top - k)$ satisfies the condition ε-approximation $\leq x$. On the other hand, $x \in (top - k)$ if ε-approximation $< x$. The downfall of this approach is that the parameter ε varies from application to application, thus, there is no general scheme for deciding its value [24].

As stated above, choosing the value ε can be very problematic and varies from application to application. It must be noted that approximate answers are applicable when they are associated with some accuracy guarantees [38], hence the low approximation value. This means that, if a value v_i at time t_2 is within ε range from the value at t_1, it is dropped and not communicated to the coordinator. The choice of

our ε-approximation value was carefully selected due to the nature of the datasets used. The datasets is characterised by very small differences between the values which are almost negligible. Figure 2.8 is an illustration of an approximate top-k tracking.

2.5 Top-k Tracking Applications

Given a set of n distributed sensors continuously reading values and a designated coordinator C, the top-k problem seeks to determine the top-k value whenever there are updates. Due to its widespread usage, top-k algorithms have been widely used in many real-world applications such as:

- *Load-balancing.* This relates to CPU usage, that is, a number of CPUs running similar programs can be tracked to see which ones have the top-k accesses each time period or overtime. This can help to determine the most accessed CPU so that some load can be directed from it to the less accessed CPUs [16].
- *Fleet Management.* In this example, large vehicular networks are tracked to determine the Board with the top-k usage. This in effect can help balance On-Board Workload [15].
- *Anomaly Detection.* An instance of this, is an application tracking sensors to determine the sensors data using non-parametric models [36].

2.6 Literature Review Of Top-k Algorithms

As aforementioned, the main objective of top-k algorithms is to track the k values (eg, temperature, distance, etc) at a central node (coordinator) from numerous remote sites whilst optimising the communication between the coordinator and the remote sites (ie. IoT sensors in our case). This is depicted in the pioneering work of [26, 31] which focuses on providing **exact** top-k answers to one-time top-k queries where source data is accessed through a restrictive interface. Vlachou et al., [39] also put forth a SPEERTO approach, which utilises a threshold-based super-peer selection that presents the exact results progressively to the user based on the skyline of each super-peer. Mouratidis et al., [25] also tackle the exact top-k problem using two techniques.

Similar work is put forth by J. Moraney and D. Raz in their paper, On the practical detection of the top-k flows, to take a different approach to the top-k problem and study the ability to perform monitoring tasks using efficient built-in counters available in current network devices. The purpose is to monitor network traffic for various management and security systems. This paper provided us practical insights on how top-k tracking algorithms could be used to mitigate a number of network security challenges, among other things. They describe settings that the number of active flows in a network node are much larger than the number of available monitoring resources and where there is no practical way to maintain a per-flow state at the node. According to them, this situation has led to rising in the recent interest in streaming algorithms where complex data structures are used to perform monitoring tasks like identifying the top-k flows using a constant amount of memory.

Mäcker et al [24], in their paper, Online Top-k-Position Monitoring of distributed data streams also presents a model with a single coordinator and a set of n distributed nodes connected to the coordinator. Each of the n nodes continuously receives data from an input stream that is only known to the respective node. It also brings to fore that, at any time, the coordinator knows the k nodes currently observing the k largest values. A node can exchange messages with the coordinator to be able to inform the coordinator about its current value. Additionally, broadcast messages can be sent to all the distributed nodes by the coordinator. This paper form the basis of our thesis work since the model described by Mäcker et al is similar to the one that will be used in this thesis.

Furthermore, an earlier work by Olston, Jiang and Widom focused on tracking a function over single values that could vary up and down, such as monitoring their sum using **adaptive filters** [2]. Here, some uncertainty can be tolerated, so they introduce a natural "filter" approach, which assigns a local filter to each site so that if the current value is within the filter, it does not need to be reported. When a site's value falls outside its filter, the current value is reported, and the filter is re-centred on this value. Over time, some filters can be widened and others narrowed so that the total uncertainty remains bounded, but more slack is allocated to values that are less stable.

In their work, **continuous skyline maintenance** [42], the authors also introduce a general approach to reduce communication overhead in client-server architectures when monitoring distributed data streams. The notion of filters is introduced where a problem called continuous skyline maintenance is considered, in which a coordinator is supposed to continuously maintain the skyline of dynamic objects. They use a filter method which helps in avoiding the transmission of updates from the remote sites to the coordinator in case these updates cannot influence the skyline maintained by the coordinator as a means of minimising the communication overhead between the coordinator and the observers. Thus, the filter is able to capture the exact skyline at each timestamp or point in time and usually achieves significant savings in terms of network overhead. Though the filter achieved significant savings over the naive approach of transmitting all updates, in several applications, snapshot skylines may not be essential since changes occur too fast. This makes it more interesting to keep track of the records or data that appear consistently in the skyline over several timestamps. To address this, the authors further introduced the concept of frequent skyline queries over a sliding window (FSQW). In this concept, a window W_t^s constitutes a set of consecutive snapshots ending at t, thus $W_t^s = \{S_t + 1 - s, ..., S_t\}$. According to this concept, a record only constitutes a 0-frequent skyline point in W_t^s if it occurs in at least $0.s$ snapshot skylines within the window ($0 < 0 <= 0$). As the sliding window moves along the time dimension, FSQW continuously track or reports the frequent skyline points.

The filter is easily adapted for the exact processing of FSQW and despite its ability to prevent transmission of all updates, there is a possibility of requiring a large number of message updates. This problem is also alleviated with the use of a Sampling method. With this method, updates are transmitted at certain instances depending on the desired choice between accuracy and message overhead. To find a

balance between Filter and Sampling methods, a *Hybrid* approach is developed by integrating these methods (Filter and Sampling). The *Hybrid* differentiates three modes for each record, that is *filter, sampling*, or *mixed* mode, acronymed as FM, SM, and MM respectively. It also has a characteristic feature of being balance under extreme settings, where the performance of *Filter* and *Sampling* deteriorates. This general framework for the Hybrid algorithm for processing $FSQW(0, W_t^s)$ is presented in [42].

Also, in continuation of the work of [42], [24] presents and analyses a new randomised online algorithm for the **online top-k-position** monitoring problem which forms an integral part of our work. It is referred to as online because the values being read by the nodes change over time and are not known in advance. The algorithm entails monitoring the nodes currently holding the k largest values by performing a Bernoulli trials in a setting comprised of a coordinator and n distributed nodes. They describe a set of filters as a collection of intervals, with each assigned to the nodes by the designated coordinator through a broadcast channel. The maximum or minimum protocols are used to reassign filters whenever there is a filter violation. The basic idea of assigning filters to the distributed nodes is to reduce the number of exchanged messages by providing nodes constraints defining when they can safely resign to send observed changes in their input streams to the coordinator. This is describe in detail in Section 3.5.

3

Algorithms

The purpose of this section is to introduce some existing top-k algorithms and to provide clarifications on their operations. Also, our own adaption of these existing algorithms will be presented. The knowledge and experience gained through the evaluation of the existing ones will serve as the building blocks for the implementation of our adaption. The goal of our adaptation algorithm is to reduce communication complexity between the coordinating node and the distributed tracking sensor considerably.

In the implementation of these algorithms, several functions will be repetitive, namely, read, send, receiveFrom and maxValue. Below is a description on these functions:

- *read*(). At each time step t, each sensor observes or reads a value. This function is used to read these values, being it temperature, direction or speed. It has no parameters.
- *send*(v_i^t, C). It is used by the distributed sensors to communicate their reading to the communicator. At each time step t, each sensor communicates whatever value is being read using this function. It takes v_i^t as a parameter which is a representation of the value v, the identity of the sensor, i, time step t, and C, the address of the coordinator.
- *receiveFrom*(). This function is used by the coordinator to receive values being communicated to it from the distributed nodes at each time step.
- *maxValue*(V$\{i\}$). It is a function used by the coordinator to compute the top-k value(s) at each time step t_i. The output of this function is m_i^t being the top-k value. The value i is the identity of the sensor(s) with the top-k value(s) and t depicts the specific time step.

The table below is a summary of the symbols used in the algorithms and what they represent.

Symbol	**Representation**
S_i	identity of a remote sensor
v_i^t	value read by sensors S_i at a specific time t
V	values read by all the sensor at a time
m_i^t	top-k value with identity of sensor i and time t
ℓ	list of all top-k values
p	previously communicated value to coordinator

3.1 Naive-Polling Algorithm

With the naive-polling approach for determining the top-k values, all values read by each tracking sensor S_i are sent to the central coordinator C. That is, with n set of tracking sensors $(S_1, S_2, ..., S_n)$, each continuously reading or observing a number of values $v_1^t, v_2^t, ...v_n^t$ respectively, exactly n number of values will be sent to the coordinator at each time t. Therefore, the algorithm behaves the same way in the initialisation stage, $t = 1$ and the subsequent rounds $t > 1$. This leads to increased overhead in communication since large volumes of values will be communicated to the coordinating node at each time period.

Algorithm 1: Naive-Polling Algorithm

```
/* algorithm begins at time t = 1                                    */
1  Input: Each sensor reads value v_i^t
2  Output: top-k value vm_i^t is calculated at C
3  ℓ = {}
4  Function SENSOR(S_i, t)
5      for each Sensor i do
           /* all remote sensors send all their values at each time step  */
6          v_i^t ← read()
7          send(v_i^t, C)
8      end function
9  Function COORDINATOR (t)
       /* coordinator receives all sent values from sensors at each time step  */
10     V ← receiveFrom()
11     for each received value v_i^t do
12         m_i^t ← maxValues{V}
13         ℓ ← m_i^t
14     end function
15
```

Also, it must be emphasised that a naive-polling scheme that accurately tracks the k values by forcing the distributed nodes to ship every remote value update to the coordinator is clearly impractical, since it does not only impose an inordinate burden on the underlying communication infrastructure (especially, for high-rate data streams and large numbers of remote sites), but also drastically limits the battery life of power-constrained remote devices (such as wireless sensor nodes). One notable feature of this algorithm is that the coordinator does not keep track of which monitoring node is communicating which message, hence the name naive-polling.

Notwithstanding the drawbacks of this algorithm, it has high accuracy in determining the exact top-k value(s). This is because all the values being read at each time step are communicated to the coordinator. This enables the coordinator to get the exact current values to apply its computation function on.

3.2 Basic-Exact Algorithm

This algorithm is an advancement of the naive-polling algorithm. It sought to achieve better performance than the naive-polling algorithm with respect to overheads that arise from constant communication between the monitoring sensors and central coordinator.

Algorithm 2: Basic-Exact Algorithm

```
/* algorithm begins at time t = 0                                          */
Input: Each sensor read value v_i^0
Output: top-k value m^t is calculated as C
ℓ = {}
/* distributed sensors execute this code                                   */
Function SENSOR(S_i, t)
    for each Sensor i do
        v_i^t ← read()
        /* at the initialisation stage all values are sent to the coordinator
           */
        if t_i = 0 then
            send(v_i^0)
        else
            /* at time t > 0, a value is only sent to the coordinator if the
               current value is different from the previous                */
            if v_i^t ≠ v_i^{t-1} then
                send(v_i^t, C)
    end function
/* coordinator executes the following code                                 */
Function COORDINATOR (t)
    /* if a new value is received from a sensor at t                       */
    if newValue then
        receiveFrom() ← v_i^t
    /* previously received value at t − 1 if there is no value             */
    else
        receiveFrom() ← v_i^{t-1}
    /* coordinator receives all values from remote sensors                 */
    V ← receiveFrom()
    for each received value v_i^t do
        m_i^t ← maxValues{V}
        ℓ ← m_i^t
    end function

```

It must be noted that, the basic-exact algorithm behaves like the naive-polling algorithm in the initialisation stage $t = 0$. That is, at this stage, all remote sensors communicate their values to the coordinator for the top-k to be determined as pre-

sented in the algorithm above. On the other hand, in subsequent rounds, not all values read by a monitoring sensor S_i is sent to the coordinator C except when the previously read value v_i^{t-1} is different from the current value v_i^t being read (i.e. $v_i^t \neq v_i^{t-1}$) and that value v_i^t is being observed by sensor S_i alone.

There is the tendency that, this algorithm leads to an improved communication complexity since $line11$ is only executed when $v_i^t = v_i^{t-1}$. Therefore, if the values being read by the respective tracking sensors are repetitive it will result in a reduced communication complexity since the condition at $line10$ will not be met, thus those values will be dropped and not communicated to the coordinator.

This algorithm also has a high tendency of determining the exact top-k values and thus a good fit for applications with large volumes of data where exact k are required.

3.3 Simple ε-Approximation Algorithm

This is also an adaptation of the basic-exact algorithm with the key aim of improving communication complexity. Over here, some level of the error bound is permissible. That is, a strong guarantee on approximations based upon a user defined error parameter ε is allowed. This algorithm is more suitable for applications where exact top-k values are not of the essence. At time t_0, all remote sensors send their values to the coordinator for the top-k value(s) to be determined just as in algorithm 1. However, it must be emphasised that, each remote sensors must have a mechanism to save or remember the value it previously communicated to the coordinator. This is represented by p in lines 9 and 13 in the algorithm. Below is the follow of the algorithm that precedes the initialisation round.

In the simple ε -Approximation algorithm above which represents $t \neq 1$, if the current value v_i^t being read by a tracking sensor S_i is within ε-approximation away from the previous value, v_i^{t-1} will be dropped and not sent to the coordinator. Preceding values will only be sent to the coordinator if only the value is outside ε-approximation away from the previous value.

Algorithm 3: Simple ε-Approximation

```
/* algorithm begins at time t = 1                                          */
1  Input: Each sensor reads value v_i^0
2  Output: top-k value m_i^t is calculated as C
3  ℓ = {}
   /* distributed sensors execute this code                                */
4  Function SENSOR(S_i, t)
5      for each Sensor i do
6          v_i^t ← read()
           /* at the initialisation stage all values are sent to the
              coordinator */
7          if t_i = 0 then
8              send(v_i^0 C)
9              p_i ← v_i^0
10         else
               /* at time t > 0 values are only sent to the coordinator if
                  they fall outside ε                                      */
11             if (p_i/v_i^t)/100 = ε-approximation then
12                 send(v_i^t)
13                 p_i ← v_i^t
14     end function
   /* coordinator executes the following code                              */
15 Function COORDINATOR (t)
       /* if a new value is received from a sensor at t                    */
16     if newValue then
17         receiveFrom() ← v_i^t
       /* previously received value at t − 1 if there is no value          */
18     else
19         receiveFrom() ← v_i^{t−1}
       /* coordinator receives all values from remote sensors              */
20     V ← receiveFrom()
21     for each received value v_i^t do
22         m_i^t ← maxValues{V}
23         ℓ ← m_1^t
24     end function
25
```

3.4 Adaptive Filter-δ Precision Algorithm

Algorithm 4: Adaptive Filter -δ Precision Setting

```
/* L_i = lowerBound                                          */
/* H_i = upperBound                                          */
/* δ = precisionConstraint                                   */
/* W_o = δ                                                   */
1 Function filterRESET (v_i^t, δ)
      /* shrink filters                                      */
2     if (W_o = shrink) then
3         L_i ←v_i − W_o/2
4         H_i ←v_i + W_o/2
      /* expand filters                                      */
5     if (W_o = expand) then
6         L_i ←v_i − W_o/2
7         H_i ←v_i + W_o/2
8     end of function
```

This seeks to define an answer that is approximately bounded by a pair of real values L and H that define an interval [L, H] in which the precise answer is guaranteed to lie. Precision, denoted by δ, is quantified as the width of the range [H L] that is H - L = W_o with the maximum W_o value being δ. A 0 δ value corresponds to an exact precision and ∞ represents unbounded imprecision. Due to the variation in the precision or δ, the algorithm in qualified with the selected value, for example adaptive filters-δ precision The following functions were created to successfully simulate a functioning adaptive filters algorithm.

- *filter.* Intercepts update streams from sources and calls the Constraint function to maintain periodically-shrinking bounds for the objects. Each filter forwards updates that fall outside its bound to the constraints for re-computation.
- *Constraint.* Specifies the precision constraint bounds ($[L, H]$). It periodically shrinks the bound width for each object. It reallocates width if 5 successive updates fall outside the precision constraint bounds.
- *A tracking sensor.* Reads values from the datasets and drop it if it falls outside the filters and sends to the coordinator if otherwise. If it continuously falls within the precision constraint for 5 successions, it shrinks the width (W_o). On the other hand, it expands the width if it falls outside successively for 5 occasions. It must be noted that W_o cannot expand beyond δ, that is, $W_o \leq \delta$
- *Coordinator.* Computes the top-k from all values received from the tracking sensors.

Algorithm 4 is an implementation of the filters for the adaptive filter algorithm

In the initialisation stage, all remote nodes send their readings to the coordinator. However, unlike the previous algorithms (naive-polling, basic-exact and simple ε-approximation), the coordinator communicates back the top-k value and a filter to the remote nodes. This is illustrated in the algorithm below which represents the

adaptive filter setting algorithm at time t_0.

Algorithm 5: Initialisation stage for AdaptiveFilter-δ Precision

```
/* time t = 0                                                          */
1  Input: Each sensor reads value v_i^0
2  Output: top-k i.e m_i^0 and [L_0, H_0] value is calculated as C
3  ℓ = {}
   /* distributed sensors execute this code                            */
4  Function SENSOR(S_i, t)
5      for each Sensor i do
6          v_i^0 ← read()
7          send(v_i^0, C)
8          (m_i^0, [L_0, H_0]) ← receiveFrom(C)
9      end function
   /* coordinator executes the following code                          */
10 Function COORDINATOR (t)
11     V ← receiveFrom()
12     for each received value v_i^t do
13         m^0 ← maxValues{V}
14     for 1 ≤ i ≤ n do
15         [L_0, H_0] ← filterRESET(v_i^0, W_o)
           /* C sends n messages to all nodes and j messages to nodes with
              violated filters                                          */
16         send(m^0, S_i, [L_0, H_0])
17         ℓ ← m_i^0
18     end function
19
```

Algorithm 6 presents how the adaptive filter is set and how the algorithm behaves in $t_i \leq 1$.

Algorithm 6: Adaptive Algorithm $t > 0$

```
/* Initialisation of counters for filter violations and no violations */
1  pC_shrink = 0
2  pC_expand = 0
3  ℓ = {}
   /* distributed sensors execute this code */
4  Function SENSOR((S_i, t)
5      for Sensor i do
6          v_i^t ← read()
           /* If value falls outside the filter */
7          if (v_i^t and v_i^{t-1} < L_i or v_i^t and v_i^{t-1} > H_i) then
8              pC_shrink ← +1
9              pC_expand ← 0
10             send(v_i^t, C)
               /* If value falls outside filter 5 times */
11             if (pC_shrink = 5) then
12                 send((v_i^t, C), pC_shrink)
13                 pC_shrink ← 0
14         else
15             pC_expand ← +1
16             pC_shrink ← 0
17             if (pC_expand = 5) then
18                 send((v_i^t, C expand)
19                 pC_expand ← 0
20         (m_i^0, [L_0, H_0]) ← receiveFrom(C)
21     end function
   /* coordinator executes the following code */
22 Function COORDINATOR(t)
23     V ← receiveFrom()
24     for each received value v_i^t do
25         m_t^0 ← maxValues{V}
26     for 1 ≤ i ≤ n do
27         [L_0, H_0] ← filterRESET(v_i^t, pC_expand/shrink)
           /* C sends n messages to all nodes and j messages to nodes with
              violated filters */
28         send(m^0, S_i, [L_0, H_0])
29         ℓ ← m_t^0
30     end function
31
```

3.5 Online Top-k Position

The algorithm begins with the coordinator determining the k-th and $(k+1)$-st largest value and, based on their midpoint $midP_i^0$, a filter set. A broadcast of this value is then made to all the distributed nodes. It is assumed that the coordinator and the sensors are on the same network and thus, the broadcast takes a unit communication.

Algorithm 7: Initialisation stage for Online Top-k Position

```
1  Input: Each sensor read value v_i^0
2  Output: Midpoint midP_i^0 value is calculated at C
3  ℓ = {}
4  Function SENSOR(S_i, t)
5      for Sensor i do
6          v_i^0 ← read()
7          send(v_i^0)
8          midP_i^0 ← receiveFrom(C)
9      end function
10 Function COORDINATOR (t)
11     V ← receiveFrom()
12     for each received value v_i^t do
13         m_i^0 ← maxValues{V}
14         midP_i^0 ← top−k+1/2
           /* C broadcast the current Mid-point                    */
15         broadcast(midP_i^0)
16         ℓ ← m_i^0
17     end function
18
```

And during the rounds $t > 2$, a distributed sensor is only permitted to communicate it's value to the coordinator only when there is a filter violation. A filter is violated when a node is within the non top-k sets but its current value is above $midP_i^t$ or if its within the top-k set but its current value is below $midP_i^t$. A new top-k set is calculated after a filter violation, hence a new midpoint $midP_i^t$ is also calculated and broadcasted to all the nodes (broadcast cost a unit message). Depending on whether the maximum or minimum top-k are of interest, we apply either the maximum or minimum protocol respectively.

In this paper, the maximum top-k is of interest and therefore we will elaborate on how it is determined. To determine this maximum top-k value, random values are generated for all the nodes. The coordinator then performs Bernoulli trials which a probability rate of $2^t/n$. This is synonymous with flipping a coin because based on the outcome of the computation, nodes with random values above or below the probability rate are assigned 1 or 0 respectively. Nodes with a probability rate of 1 sends their values to the coordinator and deactivates themselves. The coordinator will then compute the highest value and broadcast it to the nodes. In the next

round, the remaining nodes will begin the algorithm again by being assigned random numbers and based on the probability rate, if it is assigned 1 or 0 in a similar fashion as before. At this point, a node with 1 probability rate can only send its value to the coordinator if its value is greater than the received broadcast value, otherwise, it deactivates. If at the end of this round, a node has sent, a higher value, the coordinator computes a new maximum and broadcast to the remaining nodes to begin the algorithm. This is done iteratively until all nodes turns inactive and deactivates.

Algorithm 8: Online Top-k Algorithm at $t_i > 0$

```
1  Input: Each sensor read value v_i^0
2  Output: Midpoint value midP_i^t is calculated at C
3  ℓ = {}
   /* sensors excute this code                                          */
4  Function SENSOR(S_i, t)
5      for Sensor i do
6          if v_i^t > midPorv_i^t > top − k then
7              active ← true
8              if Node i is active then
9                  if max^{ti−1} > v_i then
10                     goto line 118
11                 p ← Result from coin flip with success-probability 2^t/n
12                 if p = 1 then
13                     send(v_i, C);
14                     active ← false
15                     if all nodes active = false then
16                         send(v_i, C, max)
17     end function
   /* coordinator executes the following code                           */
18 Function COORDINATOR (t)
       /* C receives all values sent                                    */
19     V ← receiveFrom()
20     for each received value v_i^t do
21         m_i^0 ← maxValues{V}
22         ℓ ← m^0 broadcast(top-k)
23     if ((v_i^t, C, max) = receiveFrom()) then
24         midP_i^t ← top−k+1/2
           /* C broadcast the current Mid-point                         */
25         broadcast(midP_i^t)
26     end function
```

3.6 Simple Online Top-k Position

In implementing an adaptation of the top-k algorithms, we choose the strengths in the afore-implemented algorithms. The algorithm thus poses some characteristics of the simple and online position top-k algorithm. This is ideal for applications or systems which can allow a level of error in the top-k values. Consider a single designated coordinating sensor S_0 and several distributed sensor S_n reading values at each time t (eg. $t = 1$ second). At the initialisation stage, all remote sensors send their values to the coordinator for a filter $midP_i^0$ to be determined. $midP_i^0$ is determined by setting a midpoint between the top-k and top-$k + 1^{st}$ value, that is, top-$k + 1^{st}/2$. For example if C receives {3, 7, 8, 1 6, 4, 7, 1, 5, 9} at t_0. If $k = 2$, then top-k set = (9, 8), therefore least of the top-k is 8 and top-$k + 1^{st}$ is 7. It is then broadcast to all the remote nodes with a parameter ϵ, which serves a level of approximation. At time step t_0 all sensors communicate their observation and receive the mid-point value together the approximation rate.

Algorithm 9: Initialisation stage for Simple Online Top-k Position

```
/* time t = 0                                                    */
1  Input: Each sensor reads value v_i^0
2  Output: Midpoint value midP_i^t is calculated at C
3  ℓ = {}
4  Function SENSOR(S_i, t)
5      for Sensor i do
6          v_i^0 ← read()
7          send(v_i^0)
8          (m_i^0, ε) ← receiveFrom(C)
9      end function
   /* coordinator executes this code                               */
10 Function COORDINATOR (t)
11     V ← receiveFrom()
12     top-k ← maxValues{V}
13     for 1 ≤ i ≤ n do
14         midP_i^t ← top−k+1/2
           /* C broadcasts the current Mid-point                   */
15         broadcast(midP_i^t, ε)
16         ℓ ← m_i^0
17     end function
18
```

In future rounds or time steps, the remote sensors only communicate their values to the coordinator when there is a violation of the filter. A filter is violated when a sensor outside the top-k set reads a current value that is ϵ away from the midpoint m_0^t. The value of ϵ together with m_i^t is received from the coordinator at $t \geq 1$.

Algorithm 10: Simple Online Top-k Position

```
/* time t = 0                                                    */
1  Input: Each sensor reads value v_i^0
2  Output: Midpoint value midP_i^t is calculated at C
3  ℓ = {}
   /* sensor executes this code                                  */
4  Function SENSOR(S_i, t)
5      if v_i^t − midP > ε − approximation or v_i^t > top − k then
6          for Sensor i do
7              active ← true;
8              if Node i is active then
9                  if max^{ti−1} > v_i then
10                     goto line 156
11                 p ← 2^t/n
12                 if p = 1 then
13                     send(v_i^t, C);
14                     active ← false;
15                     if all nodes active = false then
16                         send(v_i^t, C, max);
17         (m_i^t, ϵ) ← receiveFrom(C)
18     end function
   /* coordinator executes this code                             */
19 Function COORDINATOR (t)
       /* C receives all values sent                             */
20     V ← receiveFrom()
21     for each received value v_i^t do
22         m_i^0 ← maxValues{V}
23         ℓ ← m^0 broadcast(top-k)
24     if ((v_i^t, C, max) = receiveFrom()) then
25         midP_i^t ← (top−k+(top−k+1))/2
           /* C broadcast the current Mid-point                  */
26         broadcast(midP_i^t)
27     end function
```

To determine m_i^t, the maximum top-k value must first be computed in a similar manner as it's done in the online position of the top-k. This is outlined below:

1. This is done by generating random values are for all the remote sensors. The coordinator then conducts Bernoulli trials which a probability rate of $2^t/n$.
2. Sensors with random values above or below the probability rate are assigned 1 or 0 respectively.
3. The sensors that are assigned with 1 communicate their values to the coordinator and deactivates from the system.
4. The coordinator then compute the highest value and broadcast it to the sensors.

In the succeeding rounds, the remaining nodes will begin the algorithm again until all the sensors deactivate. The coordinator then computes the mid-point, that is top-$k+1^{st}/2$. The result is then broadcasted together with an ε-approximation rate of error to all the nodes.

4
Evaluation Methodology

This section seeks to systematically outline the procedure we used to determine the efficiency of each algorithm. It begins by presenting the datasets that were used in our experiments. It proceeds with the implementation choices that were taken to successfully simulate each of the algorithms. This is followed by an elaboration on the metrics used in evaluating the algorithms and how the simulations were set up.

4.1 Data Set

CPU Load Dataset
The first dataset comprises hardware statistics retrieved during a load testing procedure in an Evolved Packet Configuration (EPC) from Ericsson. During a stability test of a small EPG testing environment, the dataset was captured. There are 2021 rounds of data to be monitored by 720 nodes representing the distributed sensors or CPS. An observation made from the dataset is that, the values are small and repetitive across rounds or time steps. In sections where they differ, there exist an insignificant differences between them. The table below is a snippet from the dataset. Figure 4.1 illustrates values read by sensors S_1, S_2, S_{720}, and the top-k for each time step.

Time	**Sensor1**	**sensor2**	**Sensor3**	**Sensor4**	...	**Sensor719**	**Sensor720**
1	0.4105	0.8405	0.6815	0.8405	...	0.193865	0.208654
2	0.616333	0.744333	0.632333	0.610667	...	0.20617	0.214075
3	0.6285	0.79175	0.575	0.74	...	0.212907	0.231
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
2020	84.9191	77.1779	79.8717	80.9592	...	70.4767	71.4676
2021	84.9191	77.1779	79.8717	80.9592	...	70.4767	71.4676

Table 4.1: Snippet Of CPU Load Dataset

Packet Processing Rate Dataset
This data was also obtained from Ericsson. It is made up of values representing packet processing rates. Similarly, there are 720 nodes hence $S_n = 720$. Also, there are 2021 rounds of data to be monitored by the sensors meaning $t_j = 2021$. Though this dataset constitutes larger values, they are also quite repetitive across rounds. A

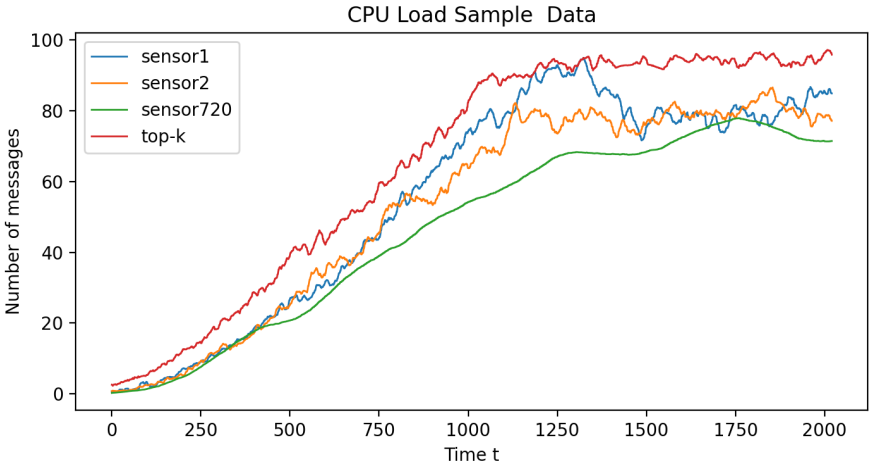

Figure 4.1: Values Read By Sample Sensors

snippet of the data is presented below. Figure 4.2 presents values read by sampled sensors and the top-k value for each time step.

Time	Sensor1	sensor2	Sensor3	Sensor4	...	Sensor719	Sensor720
1	0	0	0	0	...	6208	6896
2	1	0	0	0	...	6634	7354
3	2	0	1	7	...	7052	7777
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
2020	813	1930	1829	1865	...	20278024	20203419
2021	813	1930	1829	1865	...	20278024	20203419

Table 4.2: Snippet Of Packet Processing Rate Dataset

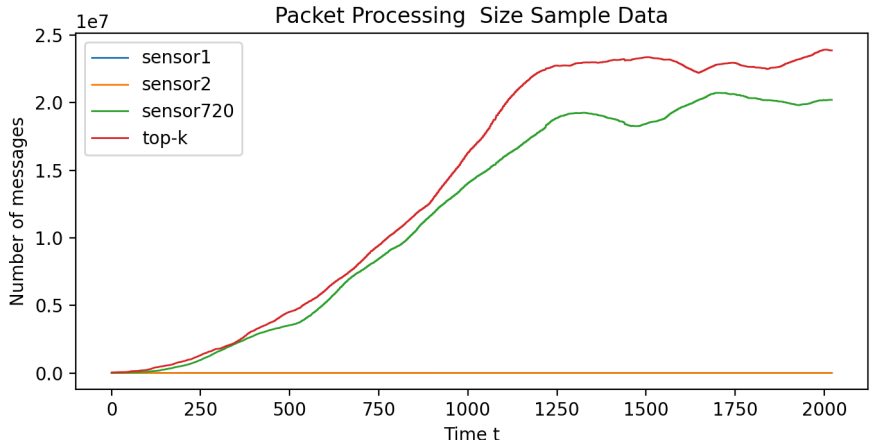

Figure 4.2: Values Read By Sample Sensors

Time	Sensor1	sensor2	Sensor3	Sensor4	...	Sensor68	Sensor69
1	-1	-1	-1	-1	...	-1	-1
2	-1	-1.1	-1.2	-1.3	...	12.99	11.78
3	-1	-1	-1	-1	...	-1	-1
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
1438	-1	-1	-1	-1	...	-1	-1
1439	-1	-1	-1	-1	...	-1	-1

Table 4.3: Snippet Of Temperature Dataset

Speed Measurement Dataset

The next dataset consists of trajectories collected within the scope of the Geolife projects by Microsoft Research Asia. This was collected on vehicular speed. When the vehicle is not in motion the sensor read -1 and it is clear from the dataset that the majority of the vehicles were not in motion in most rounds. This pattern can be observed from the table below whilst Table 4.3 gives an overview of values read by sample sensors. It constitutes 1438 nodes, a single coordinator and 9762 rounds of monitoring data.

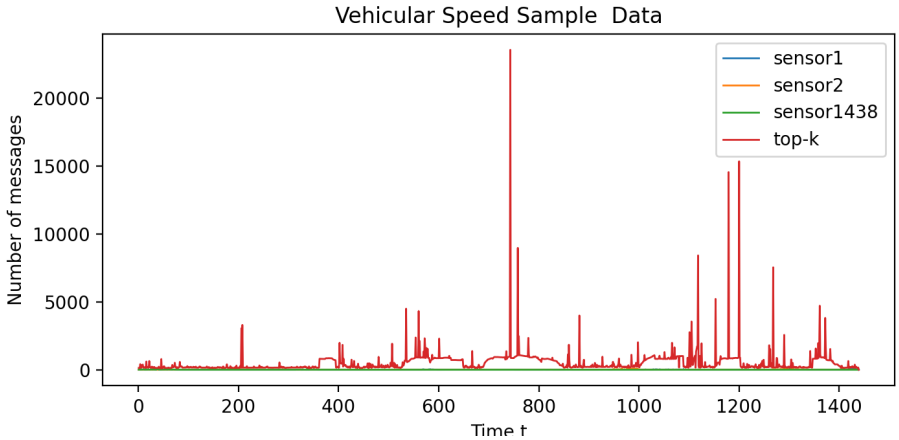

Figure 4.3: Values Read By Sample Sensors

Temperature Values Dataset

Measurements of temperature values from Inter Lab laboratory is the last dataset. If a sensor is not reading a value at a time step, it communicates 0 to the coordinator. It is observed that some sensors did not read any value through out the measuring duration. There are 68 monitoring sensors and 12500 rounds of data. The below is also a snippet of this data set.

Time	Sensor1	sensor2	Sensor3	Sensor4	...	Sensor68	Sensor69
1	-1	-1	-1	-1	...	-1	-1
2	-1	-1.1	-1.2	-1.3	...	12.99	11.78
3	-1	-1	-1	-1	...	-1	-1
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
.	.	.	.	.	...	.	.
1438	-1	-1	-1	-1	...	-1	-1
1439	-1	-1	-1	-1	...	-1	-1

Table 4.4: Snippet Of Temperature Dataset

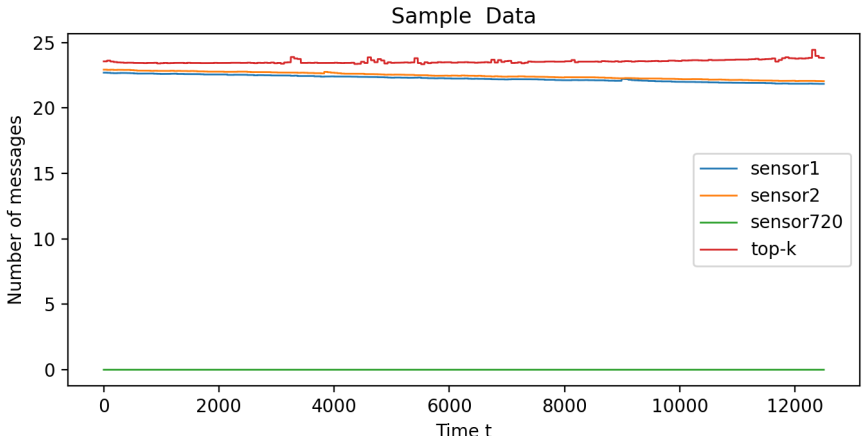

Figure 4.4: Accumulated Number Of Communication Per Time

4.2 Implementation Design Choices

This section outlines the key implementation choices that led to the successful implementation of existing algorithms and our adaptation or modification. The programming language, libraries, and data structures that were used are deliberated upon in this section.

Programming Language. Python is the scripting programming language used in the implementation of the algorithms. It emerged as the best-fit for this task because of its fast data processing capabilities. Additionally, it inhibits a clean object-oriented design, provides enhanced process control capabilities, and possesses strong integration and text processing capabilities together with its own unit testing framework. All of these contribute to the enhancement in speed and productivity. Python also has a built-in list and dictionary data structures that can be used to construct fast run-time data structures.

Libraries. As afore-mentioned, the chosen programming language provides a vast number of libraries hence, its suitability for our implementation. To better analyse and manipulate the respective datasets, pandas library was employed. Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. It helps in converting csv

files into data that is more readable by computers and also store them in data frames which can be easily accessed and manipulated anywhere in a program.

Matplotlib, a visualisation library in Python which provides an object-oriented API for embedding graphs into applications using general-purpose GUI toolkits. This library was used in our work to generate 2D graphs by plotting arrays of values comprising of the distribution in the number of communication, the average number of communication per round, top-k computation accuracy rates, etc after an entire simulation.

4.3 Evaluation Metrics

To determine the efficiency or quality of each algorithm, we needed a metric to verify this. Three important metrics were selected, that is communication complexities, computation time and accuracy rate. These were thus the basic requirements of the algorithms implemented in this work.

Communication complexity. It is the total number of communication exchanged between the designated coordinating sensor and the distributed tracking sensors. A mechanism was devised to track the communication per round and the total accumulated communication after the entire simulation. It must be emphasised that, the higher the total number of communication, the higher the communication complexity. Likewise, higher communication complexity results in higher overhead costs and vice versa. On the contrary, higher communication complexity means a lower performance with respect to achieving better communication overheads. Hence, the higher the communication complexity of an algorithm, the lower the metric of the algorithm in terms of efficiency.

Time complexity. This is the total amount of time it takes to simulate the entire algorithm. The use of similar data structures and datasets, provides a uniform platform in determining the efficiency of each algorithm with respect to time complexity. The inbuilt time function in python is called before and after the simulation to determine the amount of time used by each algorithm. The larger the amount of time used in a simulation the lower the efficiency of that algorithm in terms of time complexity.

Top-k computation accuracy rate/ correctness. The ability of an algorithm to compute valid top-k values and to accurately determine the sensor that read it for each time step, determines its computation accuracy rate. This means that, an algorithm can accurately compute the top-k value and inaccurately determine the sensor that read the value. We therefore implemented this function to track how accurate the algorithms do their computations.

4.4 Simulation Setup And Experiments

Several experiments were conducted using different non-synthetic datasets from varied sources as presented in section 4.1. A single experiment constitutes simulating

all the six algorithms independently. In each simulation, we executed a designated coordinating sensor function to receive values and another function representing the remote sensors that read and communicate their values to the coordinator. The simulations were conducted on a single core since communication latency (which is a product of distance, data size and channel of communication) was not of interest. Memory usage by these sensors was also not considered in this setting.

All the datasets were in csv files. Each column in a file represents data to be read by a distributed tracking sensor or CPS and communicated to the coordinator for the top-k value(s) to be computed. Additionally, a row represents a round or time step t. Therefore, the remote sensorsfunction was implemented to read values from a pandas DataFrame imported from the csv files. The values read by the sensorsfunction is sent to a coordinator function if a condition is satisfied (depending on the algorithm). The coordinator function keeps track of sensor communicating the maximum value (top-k) in each time step. The functions are modular and thus accept all types of values from a DataFrame. Some properties of the interaction between the coordinator and the distributed sensors functions were also modelled or implemented to monitored these properties. Listed under their respective experiments are the properties that were monitored.

Experiment 1. The first experiment was conducted using CPU load dataset from Ericsson. In this simulation setup, the following were monitored and recorded for analysis.

1. Average number of communication per time step.
2. Number of accurately computed top-k value.
3. Number of accurately computed sensor with top-k .
4. Total duration by sensors during simulation.
5. Total duration by coordinator during simulation.
6. Effects of the level of approximation bounds or precision control on communication, top-k accuracy and computation times.
7. Average deviation by the approximation algorithms per simulation.

Experiment 2. This experiment was also conducted using the packet processing sizes dataset from Ericsson. The following were also of interest and recorded for future analysis.

1. Average number of communication per time step.
2. Number of accurately computed top-k value.
3. Number of accurately computed sensor with top-k.
4. Effects of the level of approximation bounds or precision control on communication, top-k accuracy and computation times.
5. Total duration for simulation.
6. Average deviation by the approximation algorithms per simulation.

Experiment 3. Another experiment was also conducted using the temperature values dataset Interlab. The following are the vital components of this simulation setup that were recorded and subjected to further analysis.

1. Average number of communication per step.
2. Number of accurately computed top-k value.
3. Number of accurately computed sensor with top-k .

Experiment 4. In the last experiment, we used the vehicular speed dataset from Asia. The following were properties of the algorithms that were tracked and used for our analysis.

1. Average number of communication per time step.
2. Number of accurately computed top-k value.
3. Number of accurately computed sensor with top-k.

5
Results

To test the efficiency of the algorithms under investigation and to find grounds to answer our research questions, several experiments or simulations were conducted on different datasets. After the numerous simulations of the various algorithms with these datasets, several graphs and tables were generated. These graphs were then subjected to rigorous analysis and discussions. It must be emphasised that, all the algorithms are deterministic. That is to say, they will select the same top-k when presented with the same dataset at different times. The succeeding sections of this chapter present our experimental results and their analysis. Values and messages are used interchangeably.

5.1 CPU Usage Experiment

This experiment involves the simulation of CPU usage datasets from Ericsson. There were 720 distributed CPS or IoT sensors ($n = 720$) reading the load sizes and sending it to a single coordinator C. The simulation lasted for 2021 rounds (that is, $t_j = 2021$). Essentially, these simulations are applicable in distributed systems that seek to achieve better CPU load balancing. Examples of such distributed systems include autonomous vehicles, cloud computing virtual Instances, web search engines. In these simulations $k = 1$ whilst time step $t = 1$ round.

5.1.1 A Tour Of All Tested Algorithms

In this section, we will analyse the results of the simulation. Some of the vital components of the analysis include the number of messages communicated between the distributed sensors and the central coordinator, mean distribution of such communication, the accuracy of the various algorithms in computing valid top-k values, computation time, etc.

5.1.1.1 Number Of Communication

The focus here is to determine the average communication per round and the mean distribution of communication between the distributed monitoring sensors and the designated coordinator after the simulation. This is to help evaluate the communication performance of the algorithms under investigation.

The graphs in Figure 5.1 are the simulation results of the naive-polling algorithm. With all sensors transmitting values being read in every round or time step, it can

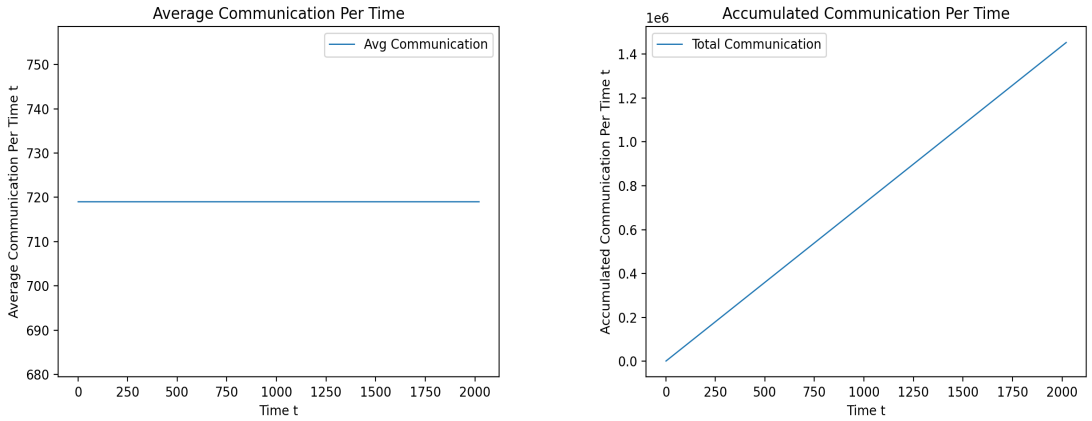

Figure 5.1: Naive-Polling Algorithm

be realised from the first graph in Figure 5.1 that, at each time step, 720 messages are sent to the coordinating sensor C. This leads to a horizontal slope. The second graph also illustrates the total accumulated number of values sent to the coordinator during the entire simulation when time t_{2021}. Also, with the same number of values being transmitted during each time step, the accumulated number of communication increases at a constant rate. This leads to a steady rise in the accumulated number of communicated values over time as depicted in the second graph in 5.1. This will lead to high overhead cost due to the volume and frequent transmission of values as claimed by [10].

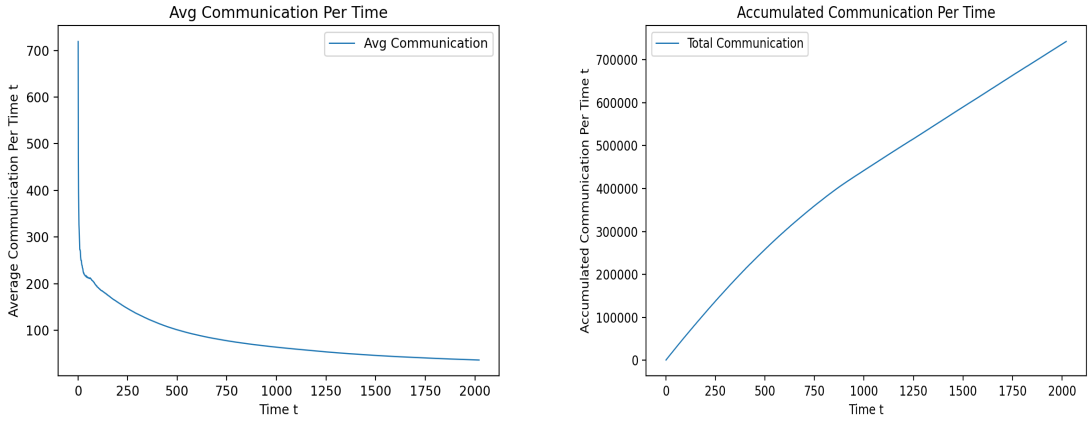

Figure 5.2: Basic-Exact Algorithm

Figure 5.2 represents the simulation results from the basic-exact algorithm. The same data sets were used as in the naive-polling algorithm. As afore-mentioned in chapter 3, if a sensor reads the same message as the preceding round, the current message is not read. It must be emphasised that, the coordinator thus uses the last communicated value in its k computation. The repetitive nature of the dataset led to a varied number of values being transmitted to the coordinator at each time

step t as depicted in the first graph in Figure 5.2. This means that, the higher the repetitive number of values in preceding rounds, the lower the number of values that are likely to be transmitted to the coordinator and vice versa. As can be witnessed from the first graph in 5.2, all values were transmitted to the coordinator during the first round ($t = 0$) because there were no preceding values to compare with. The number of communicated values decreased immensely after the initialisation round. Therefore, looking at the average number of communication in this algorithm in comparison with the naive-polling algorithm, there is a lower number of communication. This can be visualised from the slope in the second graph in Figure 5.2 which presents the accumulated number of communication recorded during the entire simulation of this algorithm.

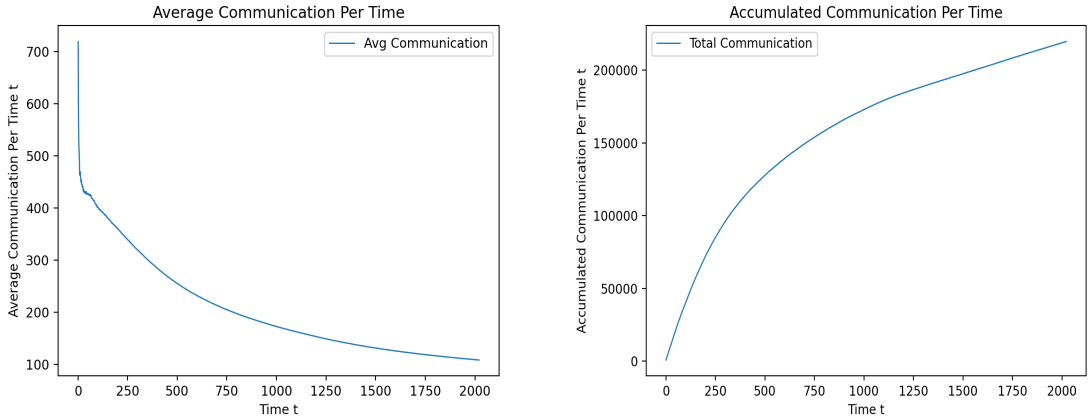

Figure 5.3: Simple 1.5%-Approximation Algorithm

Results of the simple ε-approximate algorithm are also presented in Figure 5.3. The first graph depicts the average number of values sent in each round to the designated coordinator. With some level of approximation allowed by this algorithm, 1.5 percent of error guarantee bound was permitted in the simulation, that is $\varepsilon = 1.5$. It means that, if the difference in the last communicated value and the current value is within 1.5 percent, the value is not read and the coordinator uses the value previously sent by the distributed sensor. Relatively, a lower amount of messages are sent in each round with exception of the initialisation round where all values are transmitted to the coordinator. In the subsequent rounds, the average number of communicated values reduces substantially as depicted in the first graph in Figure5.3. This explains why the slope for the accumulated number of the communicated values (that is, the second graph) flattens from time step t_{750} until the end.

In Figure 5.4, the simulation results for the adaptive filters algorithm are presented. We employed a precision control of 0.05, meaning that, the difference between the lower and upper bound is 0.05 (U - L = 0.05). Just as transpired in the previous algorithms, all values or messages were transmitted to the coordinator during the first round. The situation is different in the subsequent rounds as a number of

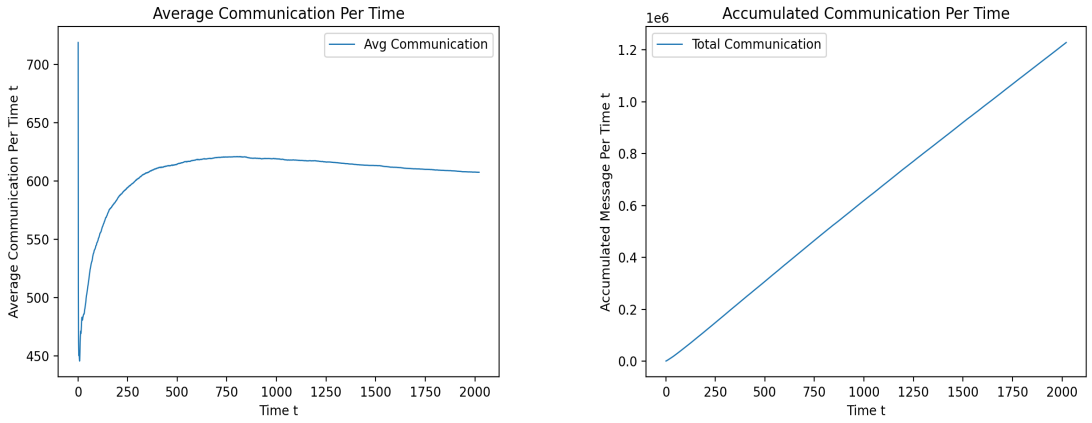

Figure 5.4: Adaptive Filters-.05 Precision Algorithm

messages are filtered out and not communicated to the coordinator. This explains why at time step t_0, 720 values are transmitted to the coordinator but decreased drastically in the leading rounds as witnessed in the first graph. At approximately t_{125}, the slope hits its lowest and begins to rise sharply and begins to flatten at time t_{750} until the end of the simulation.

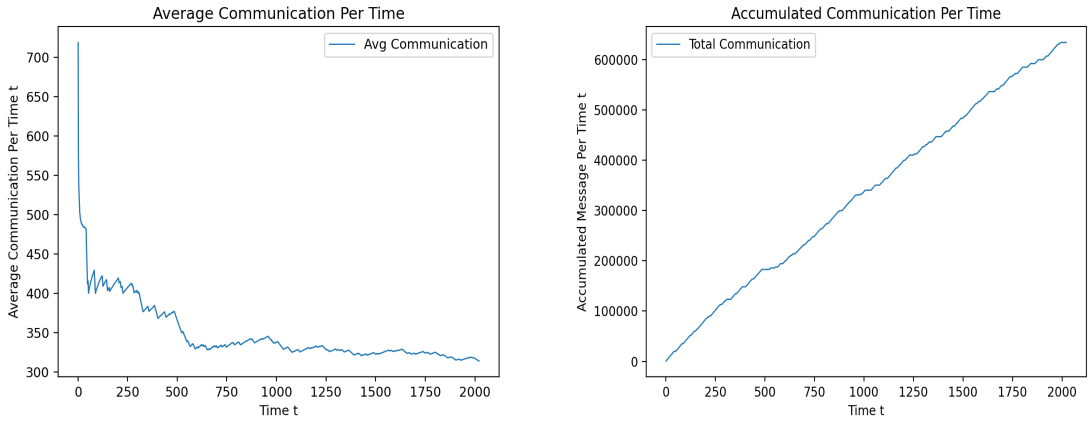

Figure 5.5: Online Position Top-k Algorithm

The next algorithm to be simulated is the online position top-k algorithm. The results for this simulation are presented in Figure 5.5. In this simulation, values are only transmitted to the coordinator when there is a violation as described in section 3. The first graph in the Figure 5.5 illustrates the average number of values communicated per each time step. It is seen from the graph that, the average number of communicated values falls steeply after time t_0 until t_{50}, after which it slowly fluctuates until the end of the simulation which is attributed to a number of messages being filtered out. This explains why the slope for the total accumulated communication is less steep.

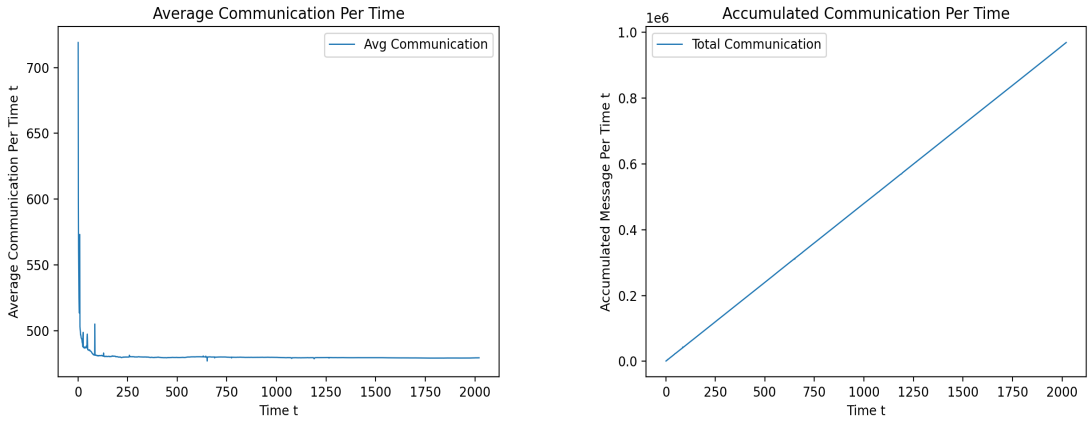

Figure 5.6: Simple Online Position Top-k Algorithm

We introduced in this work, simple online position top-k algorithm, an adaptation of online position top-k and the simple ε-approximate algorithm. In this algorithm, an ε error bound of 1.5 is allowed on the mid point as determined in the online position top-k algorithm. The simulation results are presented in Figure 5.6. In the first graph which represents the average number of communication per time step, it is evident that the number of messages transmitted in each round drops significantly after the first t_0. Compared to the simple ε-approximate algorithm or the online position top-k, a relatively larger volume of communication was witnessed during the simulation of this algorithm. This explains why the slope for the online position top-k algorithm is much steeper than the simple ε-approximate algorithm and the online position top-k algorithm.

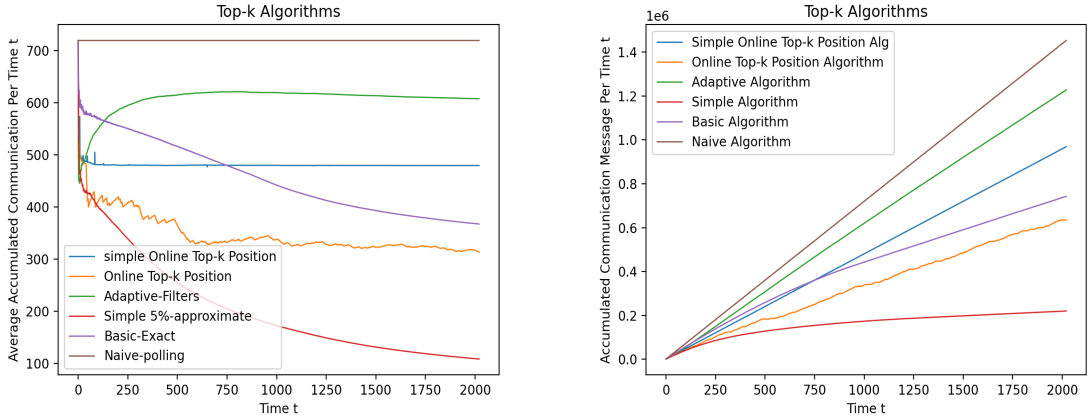

Figure 5.7: Comparison Of All Algorithms

To better compare the performance of all the above algorithms in terms of communication efficiency, the communication per each algorithm is displayed in Figure 5.7. The second graph corresponds to the total accumulated number of communi-

Algorithm	**Naive**	**Basic**	**Simple** - 1.5%	**Adaptive -0.05**	**Online top-k**	**Simple Online**
Time Steps	2021	2021	2021	2021	2021	2021
Communication	1455120	742456	219710	1227922	634306	969130
Rate (%)	100	51.02	15.09	84.39	43.91	66.6

Table 5.1: Total Accumulated Communication

cation recorded by the various algorithms during their respective simulations. It must be noted that, the lesser the communication, the better the communication efficiency. From this graph, it is clearly seen that the simple 1.5%-approximate algorithm has the best communication efficiency. This is followed by online top-k position, basic-exact, simple online top-k position, adaptive-.05 precision filters and lastly, naive-polling algorithms in that succession.

We also wanted to find out whether an increment or decrement in the error bound for the simple ε-approximate algorithm and the expanding or shrinking of precision control for the adaptive filters-δ precision algorithm will have an impact on their communication efficiency. Several simulations were then run with 5% and 7.5% approximations and precision controls of 3 and 5 for the simple ε-approximate and adaptive filters-δ precision algorithms respectively. The results are presented in Figure 5.8. This really had a substantial impact on the communication complexity for the adaptive filter-δ precision unlike the simple algorithm as seen in both graphs. This can be attributed to the insignificant difference between values in the dataset.

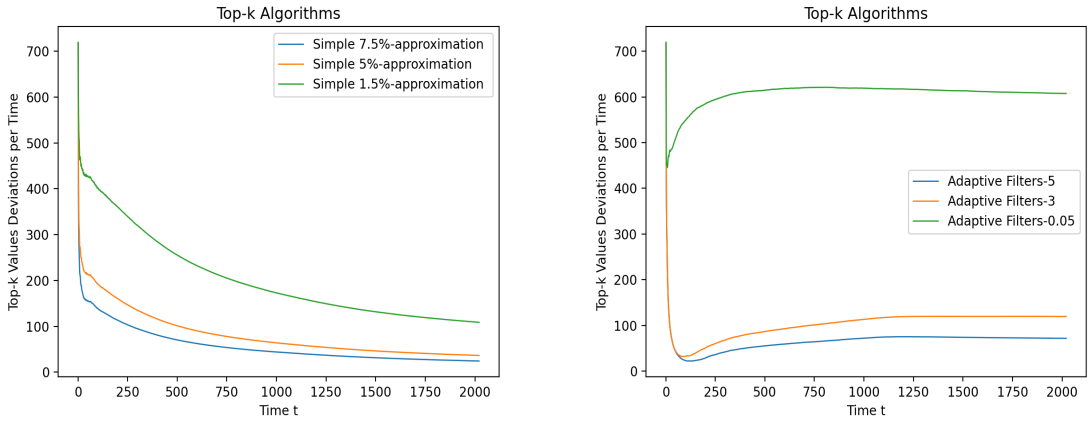

Figure 5.8: Effects Of Error Bound On Communication

The exact communication performance of these algorithms are presented figuratively in Table 5.1.

To illustrate the distribution of communication by the respective algorithms, a box plot is employed. With the constant volume of communication recorded in the naive-polling, it is justifiable of a mean communication value of 720 without any

outliers. Mean distribution of approximately 340 was recorded by the basic-exact algorithm with only one outlier of 720 which was recorded at the initialisation stage. Due to the lower volumes of communication achieved with the simple ε-approximate algorithm, less than 50 mean distribution is achieved. On the contrary, the large volumes of communication recorded for the adaptive-filters-δ precision, a mean of approximately 650 with some amount of outliers.

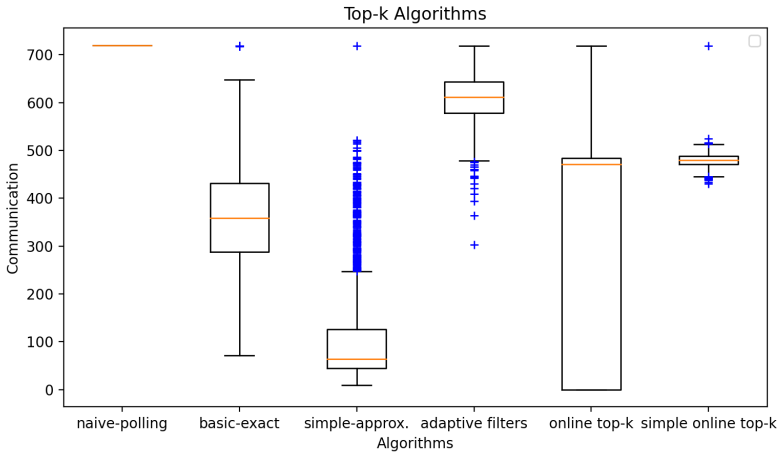

Figure 5.9: Distribution Of Communication

5.1.1.2 Top-k Accuracy Rate / Correctness

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple** 5%	**Adapt 0.05**	**Online Top-k**	**Simple Online**
Time Steps	2021	2021	2021	2021	2021	2021
Accurate Values	2021	2021	485	1758	1391	2021
Accuracy Rate	100%	100%	24.00%	86.99%	68.83%	100%
Avg Deviation	0.00%	0.00%	1.67%	0.42%	1.54%	0.00%

Table 5.2: K Computation Accuracy Rate

In as much as we are interested in finding out the communication efficiency, we are also interested in determining the accuracy with which each of these algorithms computes the top-k value. The results are presented in Table 5.2. It can be observed from the table that, the naive-polling, basic-exact and simple online top-k position algorithms accurately computed the top-k values at each time step. Also, adaptive filters-δ precision, simple ε-approximate and online top-k position algorithms recorded an accuracy rate of 86.99, 24.00 and 68.83 respectively. The margin of error or deviation of these algorithms from the accurate naive-polling top-k value(s) have been presented in Figure 5.10

One important observation made from this analysis is that, the higher the communication efficiency and higher top-k determination accuracy rate. It can therefore

be said that, there is a positive or direct correlation between the communication complexities and rate of accuracy in determining the top-k values.

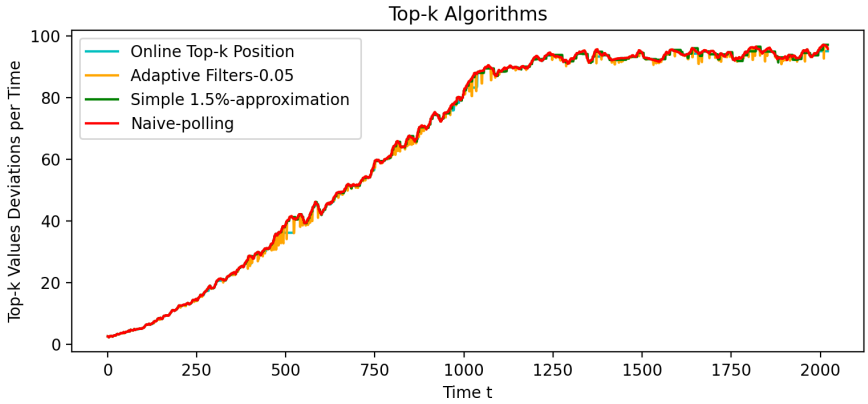

Figure 5.10: Top-k Value Computed Per Time Step

We also sort to examine the accuracy with which the coordinator computes the sensor node with the k value(s). In case of a tier break in the top-k value(s), sensors with the least ID numbers are considered in the implementation of the algorithms. For example, if $S_i > S_n$ but S_i , $S_n = v^t$, then S_i will be selected. In the approximation algorithms, it is possible for the coordinator to accurately compute the right k sensor value(s) but inaccurately compute the right sensor node with the top-k values and vice versa. The simulation results for this experiment are presented in Figure 5.11.

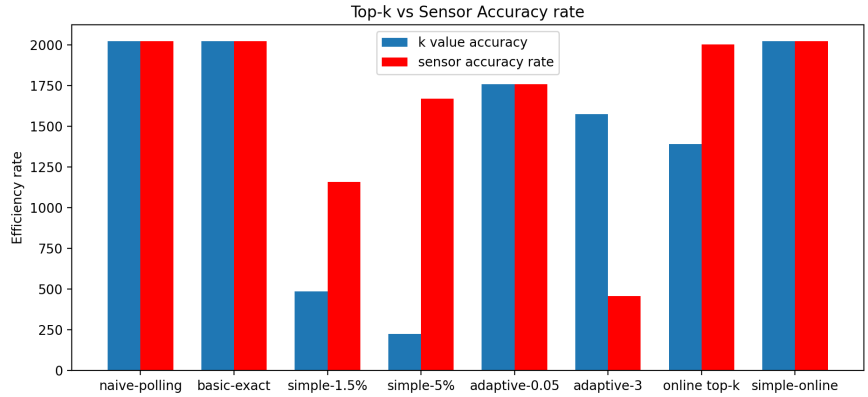

Figure 5.11: Top-k Values Vs Sensor Accuracy Rate

5.1.1.3 Computation Time

One area of interest is the processing time with which the algorithms compute the top-k values. This is necessary for real-time applications where urgency in computation is of the essence. Therefore, the duration of each simulation was recorded and

the results are presented in Table 5.3. The best algorithm in terms of processing time is the naive-polling, followed by the basic-exact algorithm. The worse performing algorithms are adaptive filters-0.05 precision and adaptive filters-2.0 precision.

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple 1.5%**	**Simple 5%**	**Adapt** 0.05	**Adapt** 2.0	**Online Top-k**	**Simple Online**
TimeStep(ms)	21.5	115.1	266.1	236.5	344.2	377.9	240.2	190.7
Coord(ms)	19.01	65.9	18.38	17.9	60.8	66.7	21.36	23.8
Total (sec)	43.5	232.6	537.8	477.9	695.7	763.7	485.277	385.488

Table 5.3: K Computation Time

5.2 Packet Processing Rates Experiment

The second experiment was conducted on a dataset that constitutes packet processing rates obtained from Ericsson. This also involves 720 distributed CPS or IoT sensors ($n = 720$) and a single coordinator C. There is a maximum of 2021 rounds or time steps of reading data. In the comparison of this dataset and the dataset used in experiment 1, this dataset has relatively larger differences between the values. Similarly, these simulation results are also essential to monitor the distribution of packets across a mobile network architecture.

5.2.1 Comparison Of All Tested Algorithms

As done in the previous experiment, this section will focus on analysing the simulation results. This will go a long way to further buttress our findings to improve the answers to our research questions.

5.2.1.1 Number Of Communication

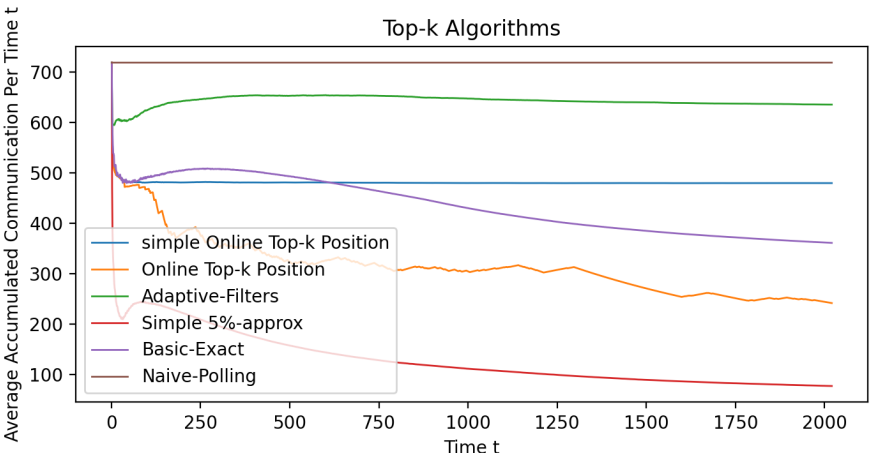

Figure 5.12: Average Number Of Communication Per Time

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple** - 1.5%	**Adapt** 0.05	**Online Top-k**	**Simple Online**
Time Steps	2021	2021	2021	2021	2021	2021
Communication	1455120	730079	329645	1449894	488495	969914
Rate (%)	100	1.20	0.02	85.97	67.23	0.81

Table 5.4: Total Accumulated Communication

Figure 5.12 is a representation of the average number of communication at each time step after the simulation of the respective algorithms using the packet size dataset. It can be seen from this figure that, the slope for naive-polling algorithm is not different from the one in the first experiment. The reason again being that, all values are transmitted to the central coordinator without any form of filtering. Hence, the plateau nature of the slope.
It is also realised that, there is 720 communications during initialisation round t_0 for the basic algorithm. The number of communication begins to decrease sharply after the initialisation stage until time step t_{200} and then stabilises until the end of the simulation. This could be attributed to the distinct nature of the data in the initial stages and how repetitive it becomes during the later stages.
An error guarantee bound of 1.5 percent was again allowed here, that $\varepsilon = 1.5\%$ for the simple approximation. From the graph, it is realised from the slope for simple ε-approximation that, the number of communication decreases significantly after the initial stages of the simulation. This means large volumes of values are filtered out after the initial stage.
The next slope to consider is the adaptive filters-δ precision algorithm. The precision control used in this experiment is 0.05. A larger amount of communication was recorded by this algorithm since most of the values read by the distributed sensors were in violation of the filters.
The online top-k position slope also dips sharply until t_{230}. It then fluctuates after the drop, until t_{2021}. This happens to be one of the most efficient algorithms in terms of communication in this experiment.
The last algorithm that was simulated in experiment 2 is the simple online top-k position algorithm and the results are depicted by the green slope. Similarly, it could be witnessed that, the slope falls sharply until it reaches approximately 480 at step t_{200}. It then maintains a relatively flat slope to the end which means the average communication is stable during those time periods.

To monitor how this dataset also reacts to an increment or decrement in error guarantee bound and precision control for simple ε- approximation and the adaptive filter-δ precision respectively, other simulations were setup. In these simulations, ε = 5 and $\varepsilon = 7.5$ whilst precision control is 3 and 5. This leads to a reduction in the number of communication from 329645 to 156200 for a simple 5%- approximation and from 1449894 to 1284801 for the adaptive filters-δ precision. Figure 5.13 is a presentation of the results after adjusting the level of approximation and filters.

Figure 5.14 elaborates on the distribution of communication for all the algorithms. It must be emphasised that, the lower the mean, the better the communication

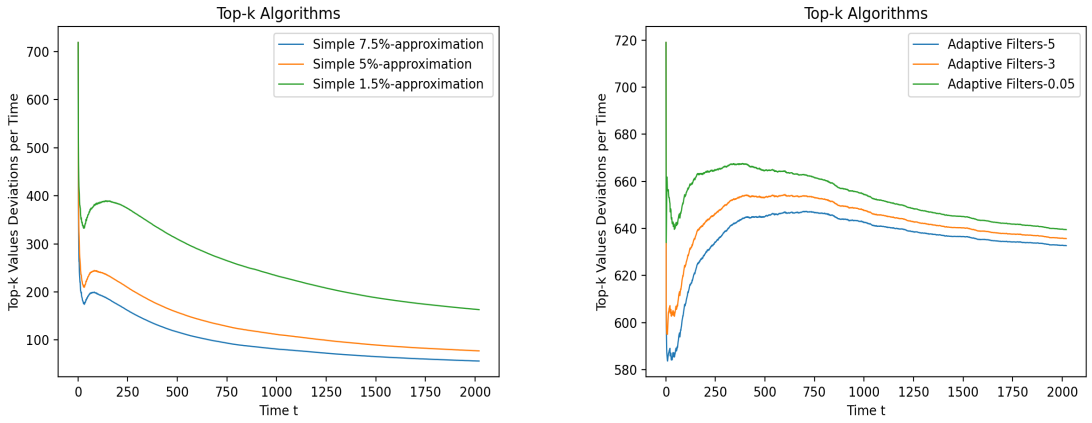

Figure 5.13: Average Communication Per Time Step

performance.

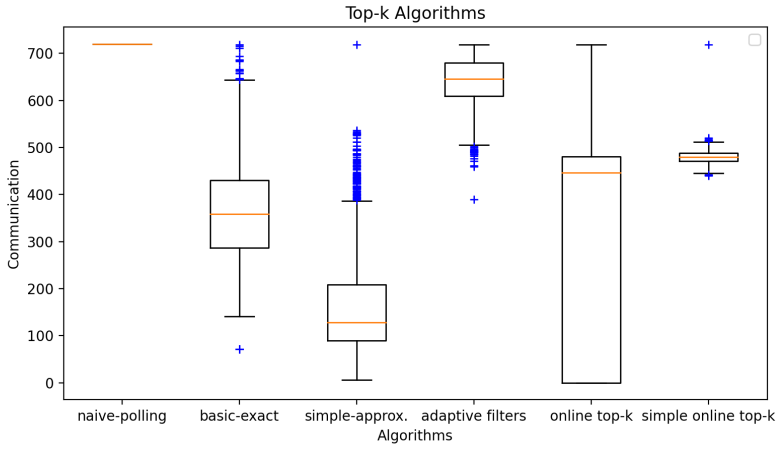

Figure 5.14: Distribution Of Communication

5.2.1.2 Top-k Accuracy Rate

At this point, we analyse the results to see the accuracy with which the various algorithms determined the top-k values after the entire simulation. The results are presented in the table below. In this table, it can be noticed that naive, basic and simple online top-k position algorithms obtained a 100 percent accuracy rate in determining the top-k values.

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple 1.5%**	**Adapt. 0.05**	**Online Top-k**	**Simple Online**
Time Steps	12500	12500	12500	12500	12500	12500
Accurate values	12500	12500	378	9429	1217	12500
Accuracy Rate	100%	100%	3.02%	75.43%	9.73%	100%
Avg Deviation	0.00%	0.00%	1.39%	2.82%	0.34%	0.00%

Table 5.5: Top-k Accuracy Rate

It was again observed that, there is a direct correlation between a number of communication and a number of accurately computed k values but an inverse relationship between the communication efficiency and the accuracy rate for computing the k value(s). That is, the higher the communication efficiency, the lower the accuracy rate and vice versa. Hence, although the simple ε-approximate algorithm has the best communication complexity, it obtained the worst accuracy rate in computing the k value. Similarly, although the naive algorithm achieved the highest communication complexity, it also achieved the best accuracy rate. After widening the error bound and filters, we also determined how it affects the accuracy rate. The results are presented in Figure 5.11. Figure 5.15 also illustrates how the tested algorithms deviated from the accurate top-k values computed by the naive-polling algorithm

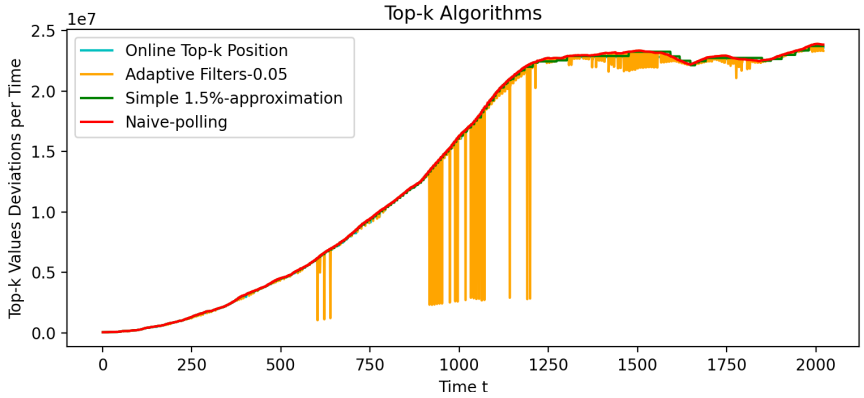

Figure 5.15: Deviations Of Top-k Values Per Time

We also examined to see the accuracy with which the algorithms select the sensor or node with the top-k value. As said earlier, the approximate algorithms have a tendency for the coordinator to compute the accurate k values but not the sensors in cases where multiple sensors read the same data. This is depicted in Figure 5.16.

5.2.2 Computation Time

The various simulations of the algorithms were monitored to compare the computation or processing time used by these algorithms to compute the top-k value(s). The table below presents the average time per time step per sensor, total time by all sensors and by the coordinator and the total time used by both the sensor and the coordinator in the simulation of the algorithms.

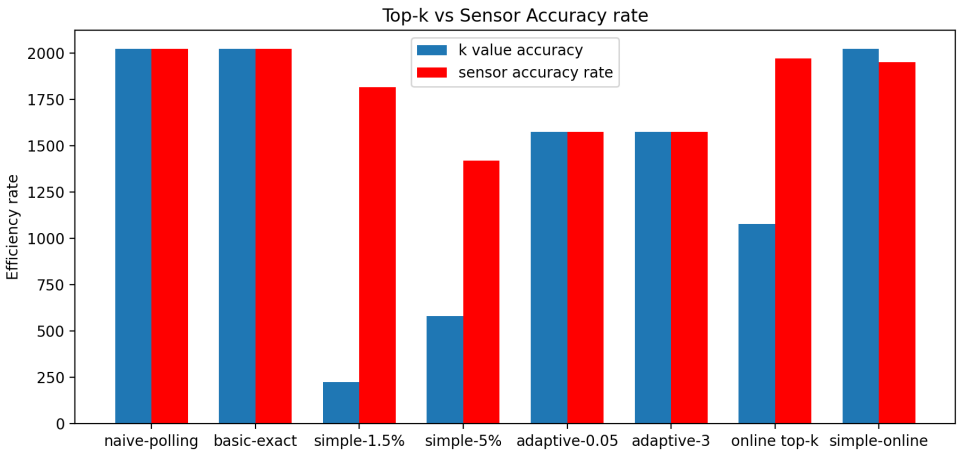

Figure 5.16: Top-k Computation Accuracy Rate

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple 1.5%**	**Simple 5%**	**Adapt 0.05**	**Adapt 3.0**	**Online Top-k**	**Simple Online**
TimeStep (ms)	24	141.54	231.09	260.27	586.44	622.51	431.05	176.97
Total (sec)	49.436	286.06	467.041	526.189	1185.2	1258.4	871.148	357.664

Table 5.6: Processing/ Computation Time

5.3 Speed Measurement Experiment

Experiment 3 was conducted on speed measurements extracted from GPS traces as presented in Figure 4.3. This also involves 9762 distributed monitoring sensors ($n = 9762$) reading the positions and communicating them to a single coordinator C. There is a maximum of 1348 rounds or time steps of reading data.

5.3.1 Analysis Of Results

This section will dwell on analysing the simulation results for the experiments. The analysis will be centered on the communication efficiency and accuracy rate for computing valid $k = 1$ value for each time step.

5.3.1.1 Number Of Communication

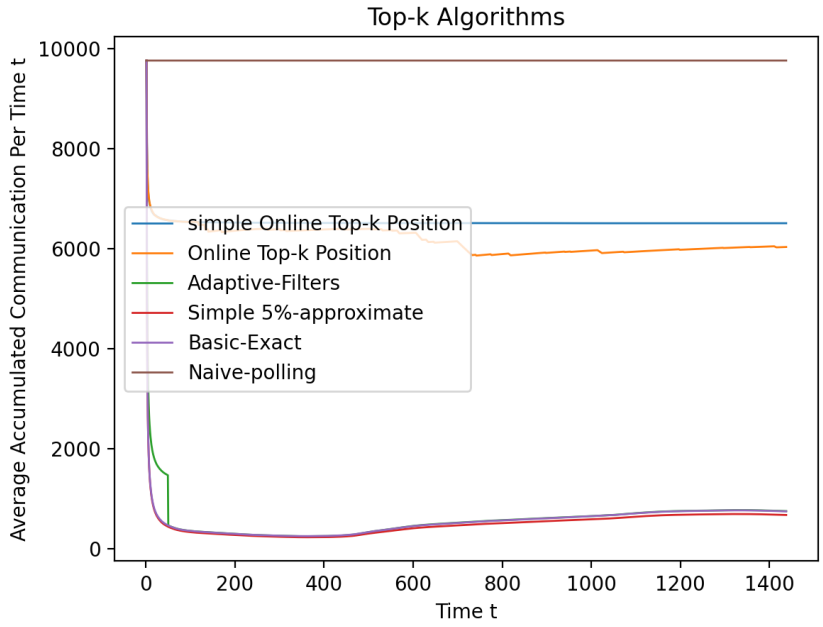

Figure 5.17: Average Number Of Communication Per Time

The figure above is a representation of the average number of communication at each time step after the simulation for the respective algorithms using the read values. It can be seen from this figure that, the slope for adaptive filters-δ precision ($\delta = 1.0$), simple-5% approximate algorithms are almost the same. This can be explained by the insignificant differences between the number of communication recorded in each time step.

It is also realised that, the average number of communication recorded for the simple online top-k position and the online top-k position was quite large, that is around 6000. The horizontal slope for the naive-polling algorithm can again be explained by the 9762 values being communicated by the distributed sensors during each round.

Algorithm	**Naive Polling**	**Basic Exact**	**Simple**	**Adapt. 1.0**	**Online Top-k**	**Simple Online**
Time Steps	1438	1438	1438	1438	1438	1438
Communication	14037756	1078218	973985	1084672	8678388	9361394
Rate (%)	100	7.68	6.93	7.73	61.82	66.69

Table 5.7: Total Accumulated Communication

Based on Figure 5.17, we are able to see the performance of the algorithms in terms of total average communication after the experiment. Simple ε-approximate, adaptive filters and basic-exact algorithms recorded similar values. For this reason, the slopes appear clustered together as the best performing algorithms in this experiment. Online top-k emerged as the next best algorithm, followed by the simple online top-k position and naive-polling being the least performing algorithm. Table 5.8 presents a summary of the communication efficiency of all the algorithms.

Figure 5.18 elaborates on the distribution of communication for all the algorithms. It must be emphasised that, the lower the mean, the better the communication performance. Similarly, the graph again depicts simple ε-approximate, basic-exact and adaptive filters-δ precision as having the lowest mean with a single outlier being the values read during time step t_0. As usual, the naive polling algorithm recorded the highest mean of 9762.

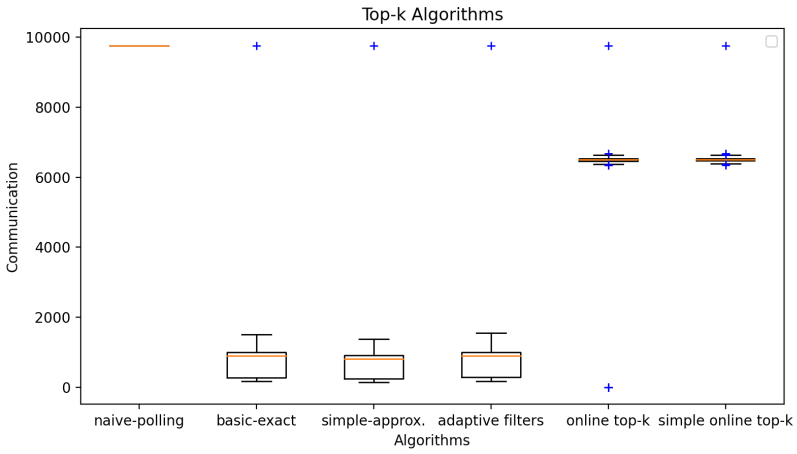

Figure 5.18: Distribution Of Communication Per Algorithm

5.3.1.2 Top-k Accuracy Rate

At this point, we analyse the results to see the accuracy with which the various algorithms determined the top-k values after the entire simulation. The results are presented in Table 5.8. In this table, it can be noticed that naive, basic and simple online top-k position algorithms obtained a 100 percent accuracy rate in determining the top-k values.

Algorithm	Naive-Polling	Basic-Exact	Simple 5%	Adapt 1.0	Online Top-k	Simple Online
Time Steps	1438	1438	1438	1438	1438	1438
Accurate values	1438	1438	718	1314	1331	1438
Accuracy Rate	100%	100%	49.79%	91.77%	92.56%	100%
Avg Deviation	0.00%	0.00%	0.51%	0.08%	0.07%	0.00%

Table 5.8: Total Accumulated Communication

Just like in experiment1, it has again been observed that, there is a direct correlation between number of communication and top-k accuracy rate but an inverse correlation between the communication efficiency and the accuracy rate for computing the k value. That is, the better the communication efficiency, the lower the accuracy rate. Hence, although the simple ε-approximation algorithm has the best communication complexity, it obtained the worst accuracy rate in computing the k value. Similarly, although the naive algorithm achieved the highest communication complexity, it also achieved the best accuracy rate.

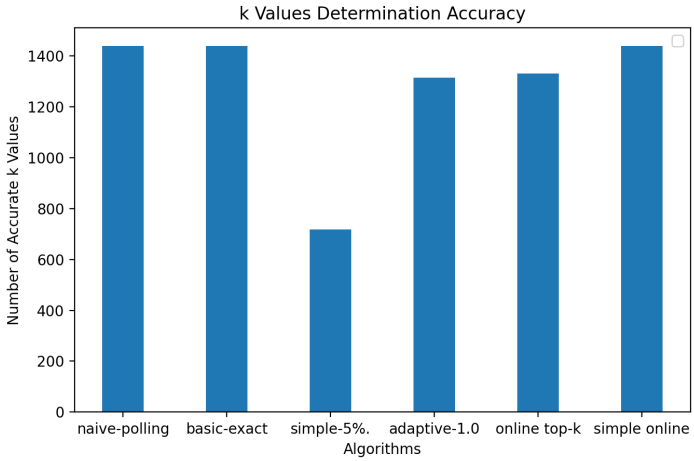

Figure 5.19: Top-k Computation Accuracy Rate

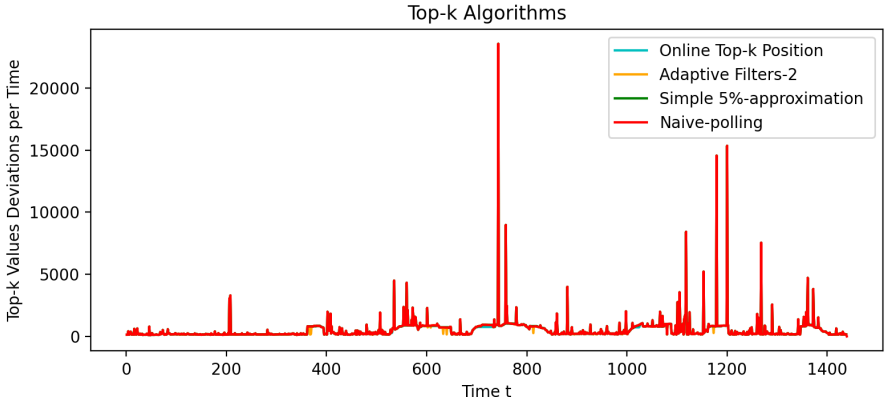

Figure 5.20: Deviation From Top-k Values Per Time

5.4 Temperature Values Experiment

A final experiment was conducted on temperature values from Inter Lap. The number of distributed tracking sensors is 58 and a single coordinator C. The simulation constituted 12500 rounds or time steps of data.

5.4.1 Number Of Communication

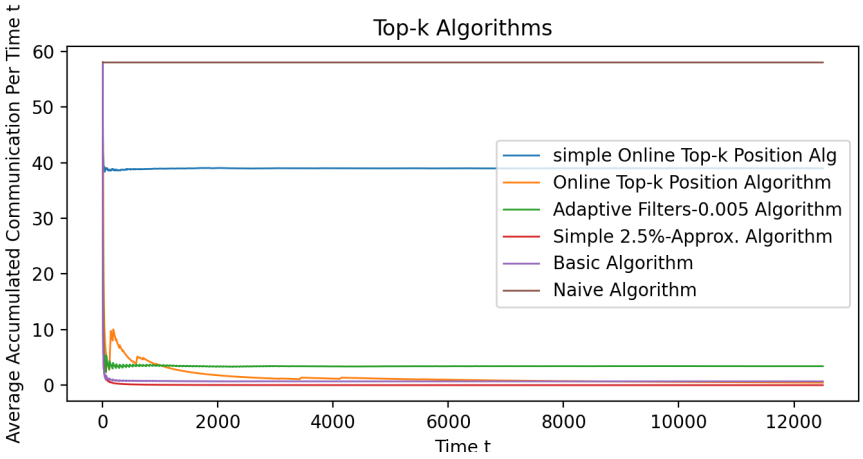

Figure 5.21: Average Number Of Communication Per Time

The graph Figure 5.21 represents the average number of communication per time by all the respective algorithms. After the simulation, the simple 2.5%-approximation algorithm emerged as the most efficient. The algorithm is begun by communicating all its readings to the coordinator but decrease to a minimum around t_{500} after which they maintain a constant average until the end. This could be attributed to the repetitive and insignificant difference between the values in the datasets. The basic-exact algorithm is the second efficient in terms of the average number of accumulated communication. This is also attributed to a large number of repetitive values in the datasets. The online top-k position algorithm is next in succession with respect to efficiency in the average number of communication per time. This algorithm also begins by communicating all its values in the initialisation stage but gradually reduces to a minimum until the time period 6000. This is followed by the simple online top-k position. At the initialisation stage, it sends all its readings to the coordinator but gradually reduces until around time 700. From this time period on wards, it maintains an average accumulated communication of around 38. The simple online top-k position is succeeded by the Adaptive filters-δ precision algorithm. A precision control of 0.005 was maintained in this experiment for the adaptive filters-δ precision algorithm. Similarly, all its values are communicated to the coordinator at the initialisation stage. It then drops steadily in the average number of accumulated communication around the time period 800 and rises in again gradually until the end of the simulation. It is also depicted in this graph, that the average number of accumulated communications for the naive polling algorithm is always the same at each time period, that is, 58 per every time period.

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple 2.5%**	**Adapt. 0.005**	**Online Top-k**	**Simple Online**
Time Steps	12500	12500	12500	12500	12500	12500
Communication	725000	8669	152	623276	5904	487427
Rate (%)	100	69.35	0.02	5.88	0.81	67.23

Table 5.9: Total Accumulated Communication

Based on Table 5.9, it can be concluded from this experiment that, simple ε-approximation algorithm has the least amount of communication. This is followed by the online top-k position and basic-exact, simple online top-k position, adaptive filters-δ precision and the naive-polling algorithms.
Figure5.22 is a box plot of the communication per round of the various algorithms to illustrate the general distribution of communication during the entire simulation.

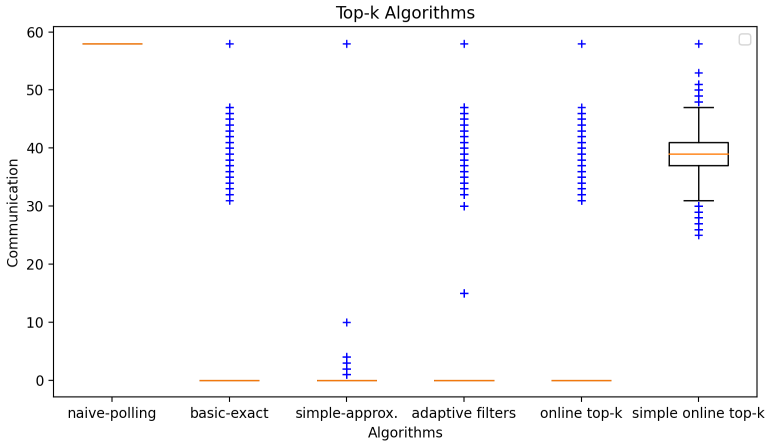

Figure 5.22: Communication Per Round

This box plot depicts a mean distribution of communication for the naive-polling algorithm to be 58 without any outliers. The mean communication for the basic-exact algorithm is also approximately 0 with some outliers from above 30 to 50 and one at 58 which explains the communication at the initialisation stage. Simple ε-approximation algorithm similarly has a mean communication of approximately 0 and a few outliers between 1 and 10 whilst there is a single outlier at 58 too which can again be attributed to the communication at the initialisation stage. Also, the mean communication for the adaptive filter-δ precision can be observed to be approximately 50 and the 25^{th} quantile around 48. Some outliers are also observed between 0 and 48 with a single one at 58. Approximately 0 mean communication is also recorded for the online top-k position algorithm with some outlier between 30 and 50. Again, there is a single outlier of 58 recorded during the initialisation stage. The last algorithm represented on the diagram is the simple online top-k position algorithm which obtains a mean communication of about 37 a few outliers between 25 and 30. It also has few outliers between 45 and 52 and similarly, 58 outliers representing the communication during the time step t_0.

5.4.2 Top-k Accuracy Rate

A breakdown for the accuracy of which the respective algorithms determined the k values in this experiment is presented in the table below.

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple 2.5%**	**Adapt. 0.005**	**Online Top-k**	**Simple Online**
Time Steps	12500	12500	12500	12500	12500	12500
Accurate values	12500	12500	378	858	1217	12500
Accuracy Rate	100%	100%	3.02%	6.94%	9.74%	100%
Avg Deviation	0.00%	0.00%	1.67%	1.60%	1.54%	0.00%

Table 5.10: K Computation Accuracy Rate

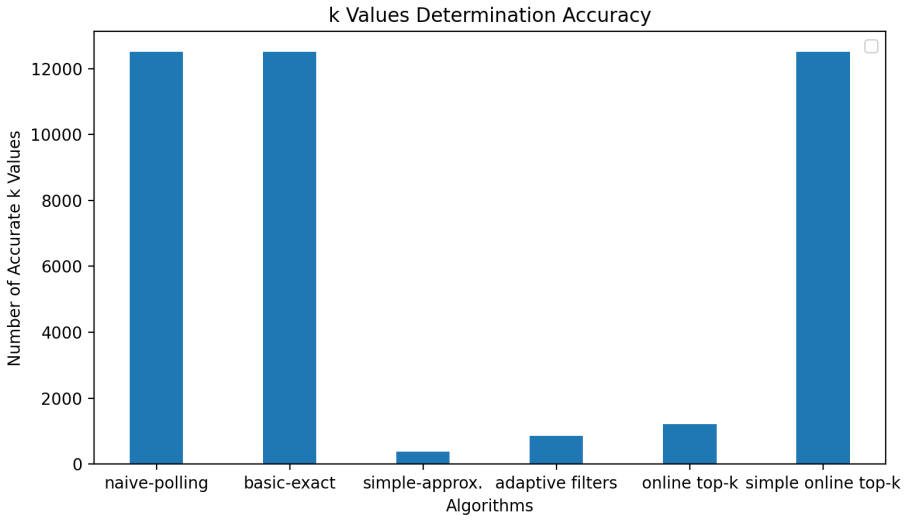

Figure 5.23: K Computation Accuracy Rate

In this experiment, which constituted a simulation of temperature values from Inter Lab, naive-polling, basic-exact and simple online top-k position algorithms determined the k values with 100 percent accuracy as depicted graphically in Figure 5.23. With precision control of 0.005 used by the adaptive filters-δ precision algorithm, it emerged as the 4^{th} best accuracy rate of 75.43 percent. It was determined that, this algorithm has a 0.42 percent average rate of deviation from the top-k value. This is succeeded by an online position algorithm with an accuracy rate of 3.02 percent and an average deviation of 1.54 percent. The algorithm with the worst accuracy in determining the k value in this experiment is the simple ε-approximate algorithm which also had an average deviation rate of 1.67 from the top-k value. Figure 5.24 represents how the margin of errors with which the non-exact algorithms deviated from the valid values which were computed in the naive-polling algorithm.

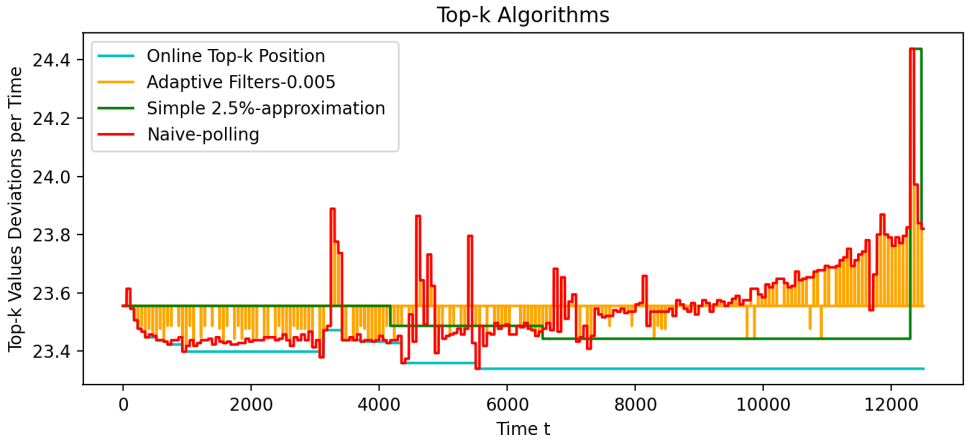

Figure 5.24: K Deviation From Top-k Values Per Time

5.5 Discussion

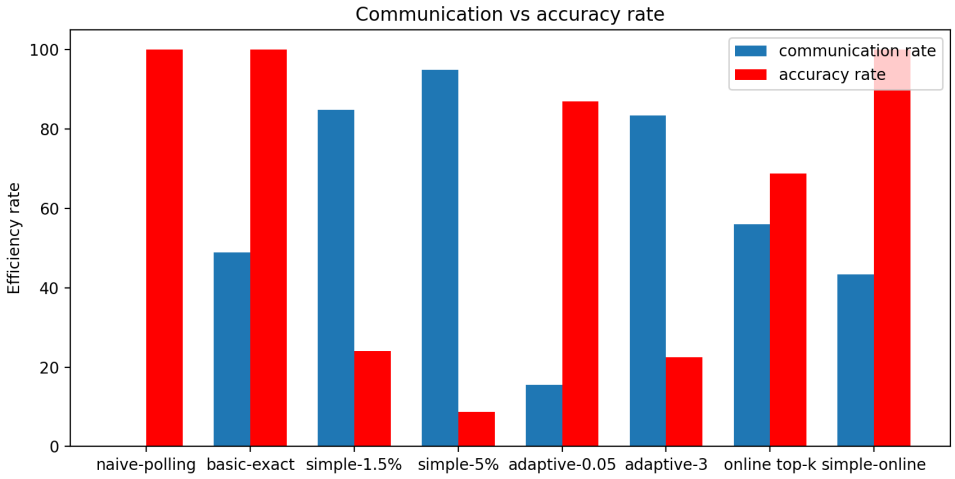

Figure 5.25: Experiment 1

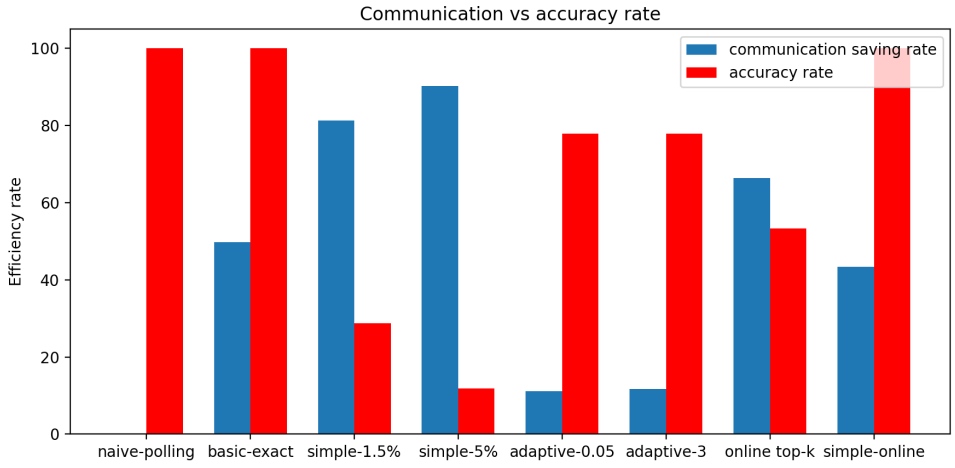

Figure 5.26: Experiment 2

After a rigorous analysis of the simulation results, some patterns were observed. These are discussed in the paragraphs below.

First, the higher the rate of communication, the higher the accuracy rate for computing the k values. This means a direct correlation between the rate of communication and accuracy rate. On the contrary, the better the communication efficiency, the less efficient the accuracy rate in computing the k values. And thus, there is an inverse correlation between the efficiency of communication and accuracy rate. It was discovered that, while exact algorithms scored in top-k accuracy computation, approximation algorithms scored high in communication efficiency.

Second, the widening or shrinking of the precision control for the adaptive filters-δ precision or the error guarantee bound for the simple ε-approximate algorithm could affect the communication rate and hence the accuracy for computing the top-k value(s) depending on the nature of datasets. Datasets with insignificant margins between successive rounds are much suited for smaller precision control value or error bound approximation. And the closer the precision control value or error bound is to 1, the higher the top-k computation accuracy rate.

Third, although the simple online top-k position is not an exact algorithm, it accurately computed the top-k values in all the experiments. This could be attributed the nature of dataset. Therefore, it is likely to deviate at some point when presented with a different dataset other than the ones used in this report.

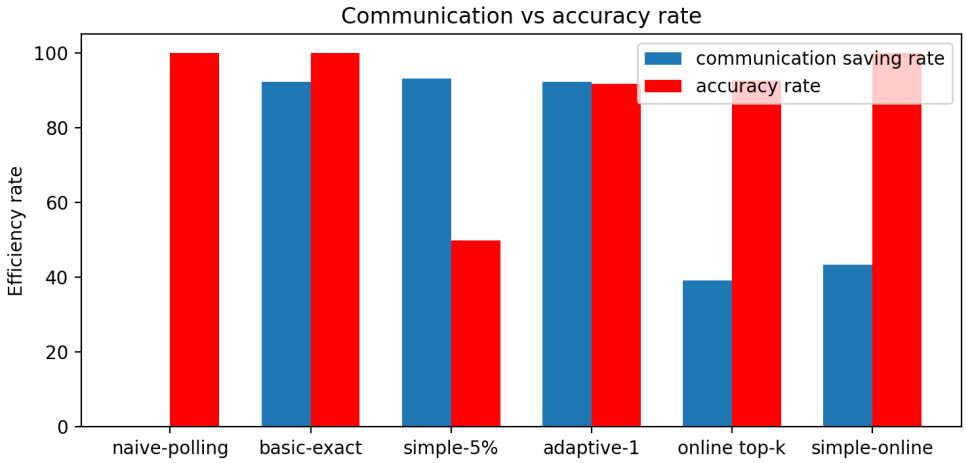

Figure 5.27: Experiment 3

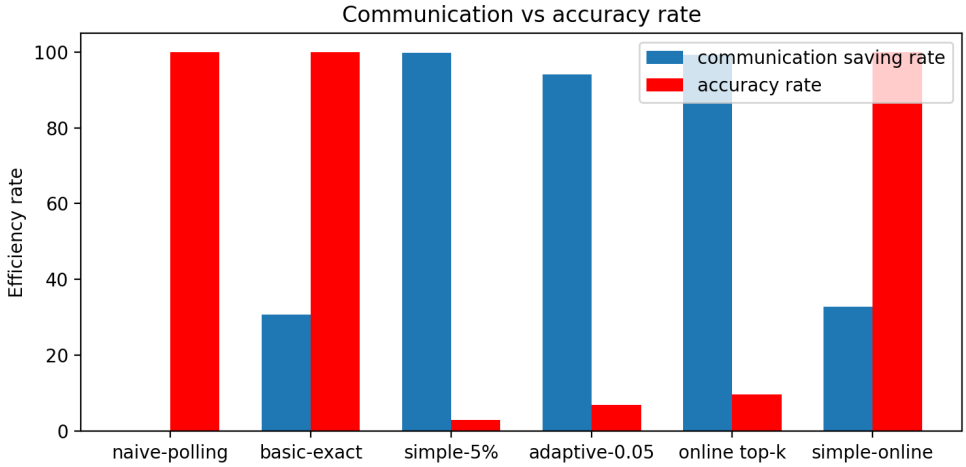

Figure 5.28: Experiment 4

With reference to Figure 5.30 which is based on the average communication efficiency of the tested algorithms from all the experiments, we can now make the assertion

that, in terms of communication efficiency, simple ε-approximate is the most efficient, followed by adaptive filters-δ precision, online top-k position, basic-exact, simple naive-polling, basic-exact, simple online top-k position and naive-polling algorithms in that order. With respect to the accuracy in computing valid top-k value(s), naive, basic and simple online top-k position algorithms are the most efficient.

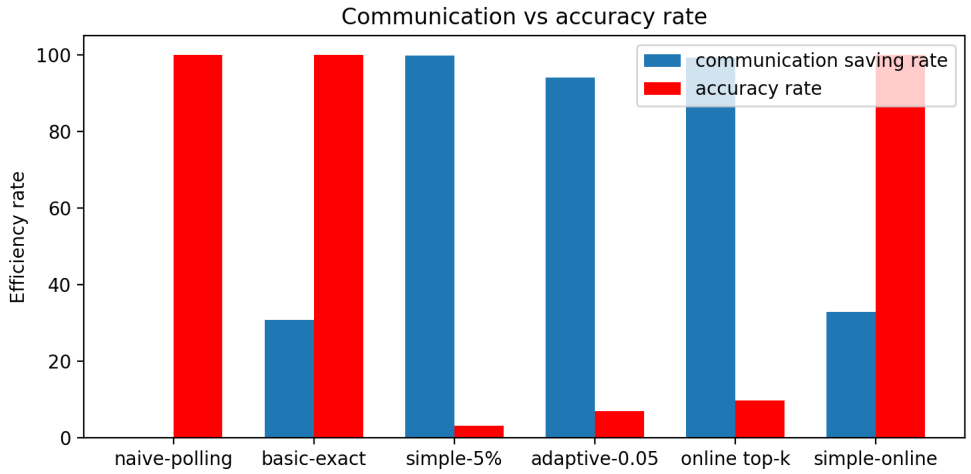

Figure 5.29: Average Number Of Communication Per Experiment

In an attempt to also determine computation time for each algorithm, we measured the processing time of each simulation. The results are presented in Table 5.11.

Algorithm	**Naive-Polling**	**Basic-Exact**	**Simple 1.5%**	**Simple 5%-**	**Adapt 0.05**	**Adapt 3.0**	**Online Top-k**	**Simple Online**
CPU (ms)	24	141.54	231.09	260.27	586.44	622.51	431.05	176.97
Packet (ms)	21.533	115.07	266.096	236.463	344.223	377.896	240.236	190.741

Table 5.11: K Computation Time Per Round (ms)

From the table, the naive-polling algorithm has the best computation time whilst Adaptive filters-3.0 precision has the worse. One pattern that can be observed from this table is that, the algorithms with worse communication performance actually recorded the least processing time. An average of the processing times recorded from simulations are presented in Figure 5.30

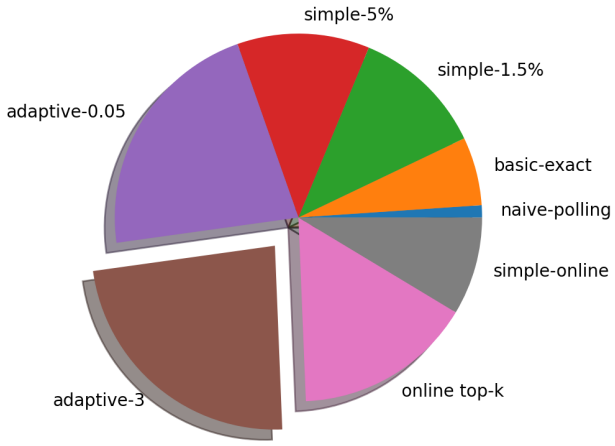

Figure 5.30: Average Computation Time Per Round (ms)

6

Conclusion

Efficient CDT Algorithms with less communication overhead and computation time is highly sort after to drive today's modern technologies that evolve around load balancing, fleet management, anomaly detection, etc. For this reason, this thesis was conducted to subject a number of existing CDT algorithms notably naive-polling, basic-exact, simple ε-approximation, adaptive filters-δ precision, online top-k position and our adaptation, simple online top-k position to a comparable analysis. The basis of our analysis was their respective communication performance, the accuracy with which they compute top-k value(s) and their processing times. The paragraphs below elaborates on the findings of our analysis.

All the investigated algorithms emerged as deterministic. Depending on the requirements of the application, one can choose between exact or approximation algorithms. Approximation algorithms were observed to generally generate less communication compared to the exact algorithms. Therefore, in a system where a margin of error on the top-k value(s) can be tolerated, one might opt for an approximation algorithm to save overhead cost. On the other hand, if accuracy in the top-k value(s) is a high requirement, then one must opt for an exact algorithm that will most likely generate higher communication volumes than an approximation algorithm.

There is therefore a trade off between communication cost and accuracy in computing top-k value(s) when selecting an algorithm for a system design or application. This is a vital choice to make since there is an inverse or negative correlation between communication efficiency and the top-k accuracy computation rate. This trade off is further complicated when processing time is to be factored into the analysis. From our analysis, it was observed that, although the exact algorithms generates relatively higher communication in comparison to the approximation algorithms, they recorded substantially lower processing or computation times.

Basic-exact algorithm scored high in all the compelling sides of the trade off. It comes next to simple ε-approximation, online top-k position and adaptive filters-δ precision in terms of communication efficiency. It is also second after basic-polling algorithm in terms of processing time. Lastly, it emerged as the best with respect to accuracy in computing the k value(s), alongside naive-polling and simple online top-k position algorithms. Therefore in a distributed real-time system that seeks to save communication overhead and also in which top-k computation accuracy is of a higher preference, then basic-exact algorithm will be the best fit.

Future works can be directed at incorporating fault tolerance among the coordinator and the distributed sensors or CPS to enhance availability. It could also further in-

vestigate the data structures and libraries to find out its impact on the processing or computation time. Another interesting are to investigate is communication latency of the various algorithms.

Bibliography

[1] Approximate frequency counts over data streams. *Proceedings of the 28th International Conference on Very Large Data Bases (Hong Kong, China), 364-357.*, 2002.

[2] Adaptive filters for continuous queries over distributed data streams. *Copyright 2003 ACM 1-58113-634-X/03/06*, 2003.

[3] Finding (recently) frequent items in distributed data streams. *Proceedings of the 2 rt InternationalConference on Data Engineering*, pages 767–778., 2005.

[4] Online mining (recently) maximal frequent itemsets over data streams. *Proceedings of Research Issues in Data Engineering: Stream Data Mining and Applications 11-18.*, 2005.

[5] Continuous monitoring of distributed data streams over a time-based sliding window. *Symposium on Theoretical Aspects of Computer Science*, 2010.

[6] Multicriterion decision in management: principles and practice, volume 25. *Springer Science Business Media*, 2012.

[7] So who won?: dynamic max discovery with the crowd. *In SIGMOD, pages 385–396*, 2012.

[8] R. M. Adelsman and A. B. Whinston. Sophisticated voting with information for two voting functions. *Journal of Economic Theory, 15(1):145–159*, 1977.

[9] Bharambe A Reiter M. Akella, A and S. Seshan. Detecting ddos attacks on isp networks. *Proceedings of the 2003 PODS Workshop on Management and Processing of Data Streams.*, 2003.

[10] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 28–39, 2003.

[11] Ho-Leung Chan, Tak-Wah Lam, Lap-Kei Lee, and Hing-Fung Ting. Continuous monitoring of distributed data streams over a time-based sliding window. *Algorithmica*, 62(3-4):1088–1111, 2012.

[12] Graham Cormode. The continuous distributed monitoring model. *ACM SIGMOD Record*, 42(1):5–14, 2013.

[13] Graham Cormode, Minos Garofalakis, Shanmugavelayutham Muthukrishnan, and Rajeev Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 25–36, 2005.

[14] Luca De Vito, Vincenzo Cocca, Maria Riccio, and Ioan Tudosa. Wireless active guardrail system for environmental measurements. In *2012 IEEE Workshop on Environmental Energy and Structural Monitoring Systems (EESMS)*, pages 50–57. IEEE, 2012.

[15] Romaric Duvignau, Bastian Havers, Vincenzo Gulisano, and Marina Papatriantafilou. Querying large vehicular networks: How to balance on-board workload and queries response timef. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2604–2611. IEEE, 2019.

[16] Romaric Duvignau, Marina Papatriantafilou, Konstantinos Peratinos, Eric Nordström, and Patrik Nyman. Continuous distributed monitoring in the evolved packet core. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, pages 187–192, 2019.

[17] B. Eriksson.

[18] Michael Fenlon, Anastasios Makris, and Paul LaFrance. Method and system for monitoring distributed systems, February 26 2004. US Patent App. 10/647,193.

[19] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. Sketch-based geometric monitoring of distributed stream queries. *Proceedings of the VLDB Endowment*, 6(10):937–948, 2013.

[20] S. Muthukrishnan K. Y. Q. Z. Graham Cormode. Optimal sampling from distributed streams. *PODS '10 Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 77–86*, 2010.

[21] H. Kim and B. Karp. Autograph: Toward automated distributed worm signature detection. *Proceedings of the 13th USENIX Security Symposium (San Diego, California)*, pages 271–286., 2004.

[22] K A. Kumar V. K S D. Kumar, V. and B. Maruti. Prototype design and continuous monitoring for bridge safety by using iot. https://www.ijert.org/research/prototype-design-and-continuous-monitoring-for-bridge-safety-by-using-iot-IJERTCONV8IS11020.pdf.

[23] L.K. Lee and Ting H.F. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. *Proceedings of the 25th Symposium on Principles of Database Systems (Chicago, Illinois)*, pages 290–297.

[24] Alexander Mäcker, Manuel Malatyali, and Friedhelm Meyer auf der Heide. Online top-k-position monitoring of distributed data streams. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 357–364. IEEE, 2015.

[25] Bakiras Spiridon Mouratidis Kyriakos and Papadias Dimitris. Continuous monitoring of top-k queries over sliding windows. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2006/01/01.

[26] L. Gravano N. Bruno and A. Marian. Evaluating top-k queries over web-accessible databases. *In Proc.ICDE*, 2002.

[27] Chang Y. Smith J. R. LI C. Natsev, A. and J. S. Vitter. Supporting incremental join queries on ranked inputs. *In Proceedings of the 27th International Conference on Very Large Data Bases. 281–290.*, 2001.

[28] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 563–574, 2003.

[29] A. Gionis P. I. R. M. Mayur Datar. "maintaining stream statistics over sliding windows,". page 1794–1813.

[30] Konstantinos Peratinos and Sarkhan Ibayev. Epgtop: A tool for continuous monitoring of a distributed system. *Master's thesis. Chalmers University of Technology, Gothenburg, Sweden*, 2019.

[31] A. Lotem R. Fagin and M. Naor. Optimal aggregation algorithms for middleware. *In Proc. PODS*, 2001.

[32] Duffield N. Spatscheck 0. van der Merwe J. Sekar, V. and H. Zhang. Lads: Large-scale automated ddos detection system. *Proceedings of USENIX Annual Technical Conference (Boston, Massachusetts)*, pages 171–184., 2006.

[33] Izchak Sharfman, Assaf Schuster, and Daniel Keren. A geometric approach to monitoring threshold functions over distributed data streams. *ACM Transactions on Database Systems (TODS)*, 32(4):23–es, 2007.

[34] R. Stanojevic. Scalable heavy-hitter indentification. *Retrieved from: http://www.hamilton.ie/person/rade/ScalableHH.pdf*, 2007.

[35] Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, and Marina Papatriantafilou. Geometric monitoring in action: a systems perspective for the internet of things. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pages 433–436. IEEE, 2018.

[36] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the 32nd international conference on Very large data bases*, pages 187–198, 2006.

[37] Mingwang Tang, Feifei Li, and Yufei Tao. Distributed online tracking. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 2047–2061, 2015.

[38] G. theobald M., Weikum and R. Schenkel. Top-k query evaluation with probabilistic guarantees. *In Proceedings of the 30th International Conference on Very Large Data Bases. 648–659.*, 2004.

[39] Nørvag K Vazirgiannis M Vlachou A, Doulkeridis C. On efficient top-k query processing in highly distributed environments. *Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD*, page pp 753–764, 2008.

[40] Ke Yi and Qin Zhang. Multidimensional online tracking. *ACM Transactions on Algorithms (TALG)*, 8(2):1–16, 2012.

[41] Ke Yi and Qin Zhang. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 65(1):206–223, 2013.

[42] Zhenjie Zhang, Reynold Cheng, Dimitris Papadias, and Anthony KH Tung. Minimizing the communication cost for continuous skyline maintenance. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 495–508, 2009.

[43] Li G Zhang X and Feng J. Crowsourced top-k algorithms: An experimental evaluation. *Proceedings of the VLDB Endowment*, 2016.

[44] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. proceedings of the 28th international. *Conference on Very Large Databases (Hong Kong, China)*, pages 358–369., 2002.

A

Appendix 1

Source Code:
CLICK HERE to direct you to source of code of the algorithms and datasets