Adaptive Stream Filters for Entity-based Queries with Non-Value Tolerance
Technical Report

Reynold Cheng[†] Ben Kao[§] Sunil Prabhakar[†] Alan Kwan[§] Yicheng Tu[†]

[†] Department of Computer Science, Purdue University, West Lafayette, IN 47907-1398, USA.
Email: {ckcheng,sunil,tuyc}@cs.purdue.edu

[§] Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong.
Email: {kao,klkwan}@cs.hku.hk

Abstract

We study the problem of applying adaptive filters for approximate query processing in a distributed stream environment. We propose filter bound assignment protocols with the objective of reducing communication cost. Most previous works focus on value-based queries (e.g., average) with numerical error tolerance. In this paper, we cover entity-based queries (e.g., a nearest neighbor query returns object names rather than a single value). In particular, we study non-value-based tolerance (e.g., the answer to the nearest-neighbor query should rank third or above). We investigate different non-value-based error tolerance definitions and discuss how they are applied to two classes of entity-based queries: non-rank-based and rank-based queries. Extensive experiments show that our protocols achieve significant savings in both communication overhead and server computation.

1 Introduction

Due to the rapid development of low-cost sensors and networking technologies, stream applications have attracted tremendous research interests lately. In particular, long-standing *continuous* queries are common in a stream environment for monitoring various network activities. Some examples include intrusion detection over security-sensitive regions; identification of Denial-of-Service (DOS) attacks on the Internet [6]; road traffic monitoring; network fault-detection; email spams detection; and web statistics collection.

In such systems, *streams* are installed that collect and report the states of various entities. Typically, this information is analyzed by a stream management system in real time. For example, in DoS detection, routes through which traffic is abnormally high are identified. Addresses from and to which packet frequencies rank among the top few might signal alerts. There are two characteristics that are commonly shared by such systems: (1) *Massive data volumes* — the number of streams could be large and they are continuously reporting updates. This leads to large message volumes and high computation load at the server; (2) *Reactive Systems* — a stream management system is often also a reactive, real-time system. It detects and responds to special events, typically with certain timing requirements. Timely processing of standing queries is an important requirement.

The two characteristics, unfortunately, are conflicting. A stream server could be crippled by the large volume of data, slowing its response to standing queries [1]. One possible solution is to trade query answer accuracy for speed. For example, a sensor that is reporting a temperature reading can be instructed not to transmit updates to the server if the current value does not deviate from the last reported value by a certain bound. This method could result in a significant reduction in message volume and thus the server's load. The drawback is that the server is processing queries based on inaccurate data. For many standing queries, however, a user may accept an answer with a carefully controlled error tolerance in exchange for timeliness in query processing. For example, for an aggregate query that asks for the average of some sensors' readings in a sensor network, a 1% error in the answer might be acceptable. Other examples where query errors are acceptable include stock quotes services, online auctions, wide-area resource accounting and load balancing in replicated servers. Several efforts (e.g., see [20, 28, 23, 8]) produce approximate answers to achieving better overall performance. In particular, intelligent protocols are proposed in [17, 10, 5] to wisely control when streams should report updates. The goal

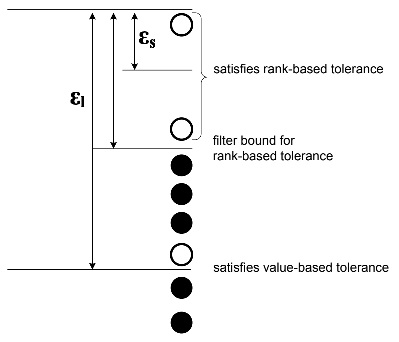
ε_s
ε_l
satisfies rank-based tolerance
filter bound for rank-based tolerance
satisfies value-based tolerance

Figure 1: Different tolerance constraints for maximum query. By using a large value ϵ_l, the user may not be aware that many streams are actually ranked above the returned answer; if the tolerance is too small (ϵ_s), the system may not benefit much from the tolerance scheme. A rank-based tolerance can be used to accurately specify that the returned answer must be ranked second or above.

of the protocols is to reduce communication overhead while at the same time user-specified error tolerances are met. These protocols make use of *filter bounds* — a system-specified range of values. A stream only reports an update if its value crosses the bound. An interesting challenge is how one could translate a user-specified error tolerance into filter bounds for streams.

Although effective, most filter-bound-based approximation techniques assume that a maximum tolerable error is specified by a *numerical value*. However, in many cases, a numerical error tolerance fails to capture the user's desired degree of accuracy. This is particularly true for queries that concern not only stream values, but also the relative *ranks* of the values as well. As an example, consider a network of sensors that report temperature readings. How would a user express an error tolerance if he is interested in the identity of the sensor with the highest temperature? One possibility is to let the user choose a numerical error tolerance, say ϵ, and the system guarantees that the answer returned to the user, say, Sensor S_1, has a value no more than ϵ smaller than that of the true highest sensor, say S_{max}. Figure 1 illustrates the idea.

However, the above strategy is not without problem. In particular, choosing an error tolerance is unintuitive. As an example, in a typical location-based application a user can inquire about his closest neighbor. Should the error tolerance specified be one meter, ten meter or one hundred meter? The user needs to have a good sense of the size of the numerical error value, which may not always be the case. In a sensor network, for instance, various kinds of data like humidity, temperature, UV-index can be collected [12]. The user may then be required to know a reasonable range of error for each data type. Moreover, if a data stream contains multi-dimensional data, (e.g., location) or multimedia data (e.g., shape), a *value-based error* can be inconvenient to specify.

Choosing a reasonable error tolerance ϵ is important, however. In particular, if ϵ is set too large, the returned stream could rank far from the true maximum (see Figure 1). To solve this problem, a user then has to be careful not to set ϵ too large. Unfortunately, unless the user has already some ideas about the data values, setting a "good" ϵ value is not easy. If ϵ is too small, then the system cannot fully benefit from the tolerance protocol. For example, in Figure 1, if the user can accept an object that ranks second or above, then the filter bound ϵ_s is too small.

An alternative approach would be to express the error tolerance in terms of a *rank* rather than an absolute value. Using our previous example again, a user could specify the error tolerance as the number of positions the returned sensor could rank below the highest one. For example, if the *rank-based* error tolerance is set to 1, then only the highest and the second highest sensors could be returned as an answer to the query. Figure 1 illustrates this example. We remark that a rank-based error tolerance could be more intuitive, particularly for queries that concern the ranking of the sensor values (e.g., top-k queries, k-NN queries, etc.) Moreover, translating a rank-based tolerance to a filter bound could also be easier. For example, after the system has taken a global snapshot of the sensors' values, it would have distinguished the sensors that are the top two from the rest of the pack. We can set a filter bound somewhere between the second highest value and the third highest value (see Figure 1). No sensors need to report updates as long as their values do not cross the bound. This is because sensors that were originally top two remain so even as the sensors' values change. Thus, either one is a good answer with respect to the rank-based error tolerance.

The above example illustrates how a *rank-based tolerance* can be used instead of *value-based* tolerance in a ranking query. A rank-based tolerance is just an example of *non-value-based* tolerance. This kind of tolerance is particularly suitable for *entity-based* queries – those that return sets of entities, rather than numerical values as answers [21]. Typical examples of entity-based queries include range queries and k-nearest neighbor queries, which are common in applications such as location monitoring [29], sensor networks [12] and computer-aided manufacturing (CAM) systems [19]. Observe that the user of an entity-based query is *not* concerned about the numerical value of the answer. Ideally, he should not need to hassle about setting a reasonable numerical error. Thus a *non-value-based* tolerance is a good match to this class of queries.

While most previous works in filter bound applications assume *value-based* queries (e.g., aggregate queries), in this paper we study extensively different kinds of entity-based queries. In particular, We study two types of entity-based queries, namely, rank-based and non-rank-based. Rank-based queries are those that concern a partial order of the stream values. Examples include top-k queries and k-nearest neighbors queries.Non-rank-based queries only concern the values of individual streams. An example is a range query.

Another dimension of our study deals with how an error tolerance is specified. Again, we are interested in error tolerance measures that are *non-value-based.* We have already discussed an example in which *rank* is used as a measure. Another possibility is to express the degree of inaccuracy through *false positive* and *false negative* [15]. Recall that the answer of an entity-based query Q is a set. Let X_Q be the correct answer set and Y_Q be the answer set returned by the system. A false positive a is an element in $Y_Q - X_Q$, i.e., a is not a correct answer but is returned as one. A false negative b is an element in $X_Q - Y_Q$, i.e., b is a correct answer not returned. (The concepts are similar to *precision* and *recall* in the IR literature [27].) A user of an entity-based query can specify the error tolerance by the maximum fraction of returned answers that are false positives, and the maximum fraction of correct answers that are false negatives. We call this kind of tolerance specification *fraction-based tolerance.*

In this paper we study how rank-based and fraction-based tolerance constraints can be exploited in a stream management system. We develop protocols that reduce communication costs between the server and stream sources, and consequently, reduce server load. Specifically, we assume each stream is equipped with an *adaptive filter* [6, 10]. A stream reports updates to the server only if the filter condition is met (e.g., "do not send an update unless the temperature value is outside the range $[20^oF, 30^oF]$). The filter constraint is usually set based on the maximum error tolerance. Since streams are refrained from sending updates, communication between stream sources and the server is reduced. Interestingly, as we will also see, our fraction-based tolerance protocols requires some stream sources to be shut down completely. This can be potentially beneficial for sensors with limited battery power since they can be operating in "sleep mode".

Another important component of our filter bound protocols is how one could map a non-value-based tolerance (either rank-based or fraction-based) to the adaptive filter constraints of the streams. As we will see later, the mapping depends on the type of the entity-based queries. In this paper we derive different protocols for rank-based queries and range queries. We will also discuss the issue of *constraint resolution*, i.e., how the adaptive filters are updated as stream values change so that the query correctness is maintained.

Although the protocols and examples presented in this paper are one-dimensional, our techniques are general and can be applied to higher dimensions.

As a summary, our contributions are:

- Motivate the need for non-value-based tolerance;
- Propose the definitions of rank-based and fraction-based tolerance for entity-based queries;
- Present protocols to exploit non-value tolerances for rank-based and non-rank-based queries; and
- Perform experimental results to test the effectiveness of the protocols on both real and synthetic data sets.

The rest of this paper is organized as follows. We discuss related work in Section 2. In Section 3, we present the assumptions of our model, and formally define the semantics of non-value-based error constraints. Section 4 presents protocols for maintaining filter constraints for rank-based tolerance, while Section 5 discusses how to do so for fraction-based tolerance. Section 6 presents our experimental results. Section 7 concludes the paper.

2 Related Work

Research in data streams has received significant interest in recent years. Issues of data streams have been surveyed in as [7].Due to the high-volume and continuous nature of data streams, systems such as STREAM [2], AURORA [11] and COUGAR [30] have been recently developed to manage them more efficiently. The goal of these systems is to conserve system resources such as memory [1], computation [19, 23, 8, 28, 20, 18] and communication costs [17, 10, 5, 22]. Most of these works reduce resource consumption by relaxing correctness requirements. Typically, a user specifies a maximum error tolerance, and the tolerance is exploited by various techniques such as approximate data structures, load shedding, filters etc. The error tolerance is often assumed to be in the form of a numerical value, and usually only value-based queries Our work investigates the possibility of exploiting non-value-based tolerance for continuous entity-based queries. Figure 2 illustrates our contribution in more details.

The idea of using adaptive filters in which filter bounds are installed to reduce communication costs was first proposed in [10]. However, that paper only considers value-based tolerance over aggregate queries such as average and minimum. Babcock et al. [6] applied a similar idea to answer top-k queries for distributed stream sources, but again the tolerance is value-based. More recently, Jain et al. [5] used Kalman Filters to exploit value-based tolerance. The Kalman Filter is installed at every stream, and with its prediction techniques it is shown to be more effective in conserving communication costs. The extension of adaptive filters in a sensor network is studied in [4]. Our

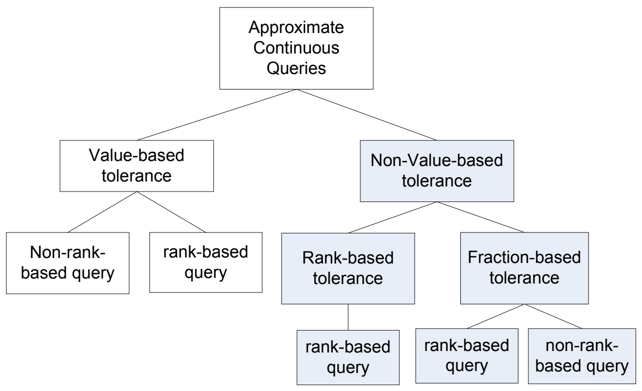

Figure 2: Our contributions (shaded).

works distinguishes from this work in that we use adaptive filters to exploit non-value-based tolerance. In addition, we study continuous k-NN queries that are used in applications such as computer aided manufacturing and traffic monitoring [19]. Notice that k-NN queries are more general than top-k queries studied in [5].

The classification of queries into value-based and entity-based has been proposed in [21]. To the best of our knowledge, the use of non-value-based tolerance for entity-based queries has only been addressed by a few researchers. Vrbsky et al. [26] studied approximate answers for set-valued queries, where a query answer contains a set of objects. They propose an exact answer E can be approximated by two sets: a *certain set* C which is the subset of E, and a *possible set* P such that $C \cup P$ is a superset of E. Khanna et al. [24] proposed a rank precision model: an answer a is called α-precise if the true rank of a lies in the interval $[r-\alpha, r+\alpha]$, where r is the rank of a informed to the user. Cui et al. [9] proposed precision and recall as a quality metric for approximate k-NN queries. We generalize the definitions of non-value-based tolerance to include rank-based and fraction-based tolerance, and we study how to exploit them in stream systems that has not been addressed before.

The idea of viewing a k-NN query as a range query was proposed by Iwerks et al. [13]. They propose the use of a closed bound which encloses at least k objects so that continuous k-NN queries can be answered more efficiently. We use a similar idea to convert a continuous k-NN query to a range query, but our focus is on how to use this technique to design filter bounds in a distributed stream environment.

3 Problem Description

In this section we describe our stream management system model, and our query model for both rank-based and non-rank-based queries. We also give formal definitions of two types of non-value-based error constraints, namely, rank-based tolerance and fraction-based tolerance.

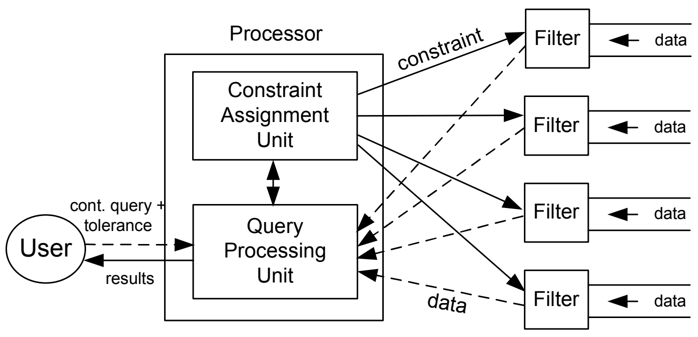

Figure 3: Stream Management System Model.

3.1 System Model

We assume a distributed stream management model similar to those described in [6, 10, 17]. The system consists of a set $S = \{S_1, S_2, \ldots, S_i, \ldots, S_n\}$ of n data streams with Stream S_i reporting a value $V_i \in \Re$. We assume that stream values are updated at discrete time instants. Each stream may be associated with an *adaptive filter* that specifies a *constraint.* With the filter mechanism, not all the updates are reported to the server. A filter constraint is a closed interval $[l_i, u_i]$, where $l_i, u_i \in \Re$. Let V_i' be the last reported value from Stream S_i. When the stream's value (V_i) changes, the filter constraint is *violated* if either (1) $V_i' \in [l_i, u_i] \wedge V_i \notin [l_i, u_i]$ or (2) $V_i' \notin [l_i, u_i] \wedge V_i \in [l_i, u_i]$. Only when the constraint is violated will the updated value be sent to the server. If no filter is installed at a stream, all updates from the stream are sent to the server.

Figure 3 shows a general architecture of such systems. Each stream source is equipped with a filter that is *adaptive* whose parameters can be changed at any time by the processor. A user submits her queries and tolerance requirements to the central processor. The constraint assignment unit then determines the relevant filter constraints to be installed in each stream. The query processing unit processes user queries and updates their results if necessary. It also receives updates from the stream sources. It communicates with the constraint assignment unit, which decides if constraints need to be revised for relevant filters.

3.2 Query Model

We are interested in a broad class of queries, called *entity-based queries*, which are those that return names or identifiers of objects as answers [21]. We further classify entity-based queries into *rank-based* queries and *non-rank-based* queries:

(1) For a **rank-based query**, the relative ranking of data items is important to the query answer. Typical examples include k-nearest-neighbor (or k-NN) query where the objects with the k shortest distances to a query point q are reported [14, 19]; and top-

k queries, where answers with the k-highest ranking scores are returned [19, 6]. In this paper we use k-NN queries as an example of how filter bound protocols are applied, since it is common in streaming systems like similarity matching in computer-aided manufacturing, mobile environments, and network traffic monitoring [19, 14, 9]. Note that a k-NN query can be easily transformed to a k-minimum or k-maximum query, by setting the query point q to $-\infty$ or $+\infty$, respectively.

(2) A **non-rank-based query** is any query that is not rank-based. In this paper we study *range queries* as an example, which are useful in stream management systems like moving-object databases [29] and sensor networks [12]. A range query is specified by an interval $[l, u]$. Streams whose values fall within the interval should be returned to the user. It is apparent that a range query is non-rank-based since the decision of whether a stream is part of the answer is independent of other streams.

For notational convenience, we use Q to denote an entity-based standing query and $A(t)$ to denote the answer set returned at time t. We use $|A(t)|$ to denote the cardinality of $A(t)$.

A standing query Q is associated with a *tolerance constraint*. We study two kinds of non-value-based tolerance constraints, namely, *rank-based tolerance* and *fraction-based tolerance*. The rest of this section examines the tolerance constraints in more detail.

3.3 Rank-based Tolerance

For a rank-based query, a user may be interested in whether the rank of an answer returned by the system matches the true rank, and if not, how *close* it is to the correct answer. For example, for a maximum query, the user may be satisfied with an answer which carries the third maximum value, but not anything further than that. A rank-based tolerance is important in situations where a large error in ranking of answers is not desirable. For example, in a distributed system, requests from different users possess various priority values, and the system should process jobs with the highest priorities. As another example, in an online game, if rewards are given to the players with highest scores, it may be unfair to give the reward to the player ranked 20th, instead of to the one ranked third.

Here, we formally define rank-based tolerance for rank-based queries. Let $rank(S_i, t)$ be the true rank of Stream S_i w.r.t. a rank-based query Q at time t. For example, if Q is a maximum query, and the system returns S_8 as the answer at time t_1 whose value actually is the third largest among all the streams, then $rank(S_8, t_1) = 3$. We note that the function *rank* depends on the query. For example, if the query is a k-NN query, then *rank* will be defined based on the differences from the query point.

Definition 1 Rank-based Tolerance. *Given a rank-based query Q, an answer set $A(t)$ returned at time t, and a maximum rank tolerance $\epsilon_k^r = k + r$, the answer set $A(t)$ is said to be correct w.r.t. ϵ_k^r if and only if $|A(t)| = k$, and $\forall S_i \in A(t), rank(S_i, t) \leq \epsilon_k^r$.*

As an example of the above definition, consider a k-NN query with $k = 3$ and $r = 2$. Then an answer set $A(t)$ is correct w.r.t. $\epsilon_3^2 = 5$ if it contains exactly three streams all of which rank fifth or above.

3.4 Fraction-based Tolerance

As we have discussed, another way to express an error tolerance is to use the concept of false positives and false negatives. The advantage of this tolerance definition is that it applies to all entity-based queries, i.e., both rank-based and non-rank-based queries. An example of fraction-based tolerance for non-rank-based queries is the sending of warning messages to soldiers that enter a danger region, in which case it is acceptable that the messages are sent to a fraction of soldiers who are not in the region (or *false positive*). For rank-based queries, k-NN queries are often used to mine multimedia data streams (e.g., images) for unknown patterns in computer-aided manufacturing [19], and fraction-based tolerance can be used to measure the quality of results [9].

Definition 2 False Positive and False Negative. *Given a query Q and an answer set $A(t)$, let $E^+(t)$ denote the number of streams in $A(t)$ that do not satisfy Q, and $E^-(t)$ denote the number of streams that satisfy Q but are not in $A(t)$. The* **fraction of false positives** *and the* **fraction of false negatives** *of Q at time t, denoted by $F^+(t)$ and $F^-(t)$, respectively, are defined as*

$$F^+(t) = \frac{E^+(t)}{|A(t)|} \tag{1}$$

$$F^-(t) = \frac{E^-(t)}{|A(t)| - E^+(t) + E^-(t)} \tag{2}$$

Note that the total number of streams that satisfy Q is given by $|A(t)| - E^+(t) + E^-(t)$. Hence $F^+(t)$ gives the fraction of the returned answers that are not correct, while $F^-(t)$ gives the fraction of the correct answers that are not returned. Figure 4 illustrates the relationship between these quantities. With those notations, we now define fraction-based error tolerance.

Definition 3 Fraction-based Tolerance. *Given a query Q, an answer set $A(t)$, a maximum false positive tolerance ϵ^+, and a maximum false negative tolerance ϵ^-, the answer set $A(t)$ is correct w.r.t. ϵ^+ and ϵ^- if and only if $F^+(t) \leq \epsilon^+$ and $F^-(t) \leq \epsilon^-$.*

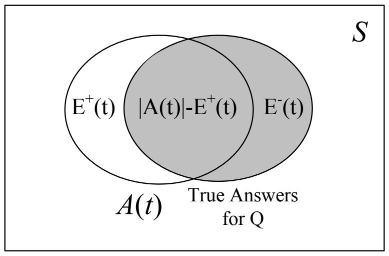

Figure 4: Definition of False Positives ($E^+(t)$) and False Negatives ($E^-(t)$). $A(t)$ is the answer set returned to the user.

The parameters ϵ^+ and ϵ^- are user-specified. The system has to guarantee that the fraction-based tolerances are met. In this paper we assume that ϵ^+ and ϵ^- are both smaller than 0.5. The rationale for this assumption is that we suppose users are not interested in results with too many incorrect answers. Another reason is that this value is required for guaranteeing the correctness of our protocols.

Again for notational convenience, we use $E^{max+}(t)$ to denote the maximum number of streams that are allowed to be incorrect in $A(t)$ and $E^{max-}(t)$ to denote the maximum number of streams that satisfy the query but are excluded from $A(t)$. From Equations 1 and 2, we have

$$F^+(t) \leq \frac{E^{max+}(t)}{|A(t)|} = \epsilon^+, \tag{3}$$

$$F^-(t) \leq \frac{E^{max-}(t)}{|A(t)| - E^{max+}(t)} = \epsilon^-. \tag{4}$$

3.4.1 Fraction-based Tolerant k-NN queries

In this section we discuss an interesting property when fraction-based tolerance is applied to k-NN queries. For a k-NN query, the number of correct answers is k. Therefore, Equation 2 becomes

$$F^-(t) = \frac{E^-(t)}{k}, \tag{5}$$

which means that at any time t, the number of false negatives ($E^-(t)$) cannot exceed k. Moreover, the number of correct answers returned in the answer set, i.e., $|A(t)| - E^+(t)$, must not exceed k. Hence,

$$1 - \epsilon^+ \leq 1 - \frac{E^+(t)}{|A(t)|} \leq \frac{k}{|A(t)|}. \tag{6}$$

This implies

$$|A(t)| \leq \frac{k}{1-\epsilon^+}, \tag{7}$$

$$|A(t)| \leq 2k. \tag{8}$$

Equation 8 is obtained by the assumption that $\epsilon^+ < 0.5$. In other words, with fraction-based tolerance, the size of the answer set returned to the user does not necessarily have to be k. For example, if the 10 nearest neighbors are queried with a fraction-based tolerance $\epsilon^+ = 0.1$, the system could return 11 streams with a guarantee that at most one of them is not correct. (That is, all correct ones are returned.) In fact, the answer set size can be controlled by ϵ^+, and is upper-bounded by $2k$. Finally, since the true answer size is always k, the following must hold:

$$|A(t)| \geq k(1-\epsilon^-) \tag{9}$$

$$|A(t)| \geq \frac{k}{2} \tag{10}$$

when ϵ^- is less than 0.5. As we can see, fraction-based tolerance limits the answer of a k-NN query to within $\frac{k}{2}$ and $2k$. This property affects the design of filter bound maintenance protocols.

3.5 Maintaining Query Correctness

With the system model and tolerance constraints defined, we are ready to describe the protocols in a stream management system that guarantee query correctness with specific tolerance constraints. All of these protocols translate tolerance constraints into filter constraints installed in the data streams. As long as the data value of a stream does not violate the filter constraint, no update is sent from the stream source to the server. When it is necessary that an update be sent to the server, the server may need to reconfigure the filter constraints. We call such reconfiguration *constraint resolution.* There are two fundamental correctness requirements for our protocols:

Correctness Requirement 1: At every point in time, if no resolution is required, then the results of all running continuous queries remain valid within their tolerance constraints.

Correctness Requirement 2: Immediately after a filter resolution process is completed, the tolerance constraint of a query is satisfied assuming that stream values do not change during resolution.

In the next section, we describe how to exploit rank-based-tolerance for rank-based queries. Section 5 develops two protocols for exploiting fraction-based-tolerance. The first protocol maintains correctness for range queries (which are non-rank-based), and the second deals with k-NN query (which are rank-based).

4 Rank-based Tolerance

Assume that q is the query point for a k-NN query. The goal of the query can then be formulated as finding the k objects whose distances from q, i.e., $|V_i - q|$, are the shortest. We use $|V_i - q|$ to decide the value of $rank(S_i, t)$. According to Definition 1, a query answer

$A(t)$ is correct at time t if its size is k and it consists of stream identifiers S_i such that $rank(S_i, t) \leq \epsilon_k^r$.

The rank-based tolerance protocol (**RTP**) described here maintains the correctness mentioned above, and at the same time exploits tolerance to reduce communication effort. Its main idea is to maintain a close interval R that encloses at least the $(k+r)$th objects closest to q. The position of R is halfway between the $(k+r)$th and the $(k+r+1)$th object. We use R as the basis for assigning filter constraints. As long as no object crosses the boundary of R, the tolerance requirements are fulfilled. An example is shown in Figure 6(a), where R lies in between the positions of the fourth-nearest object, S_4 and the fifth-nearest object, S_5.

RTP consists of two phases: **Initialization** and **Maintenance**, which are responsible for assigning and maintaining filter constraints respectively. The server maintains a set of objects, X, where each object in X lies within R. Let $X(t)$ represent the set X at any given time t. The answer set returned to the user, $A(t)$, is extracted from $X(t)$, i.e., $A(t) \subseteq X(t)$. Figure 5 illustrates these two phases.

The task of the **Initialization Phase** is to distribute the constraint R to filters. At time t_0, it collects information from all sensors and assigns appropriate values to $A(t_0)$ and $X(t_0)$. Then it executes `Deploy_bound(`t_0`)`, which calculates the constraint R and sends it to all streams. The phase enforces Correctness 1 since at any time t after the Initialization phase, if no updates are received, the server immediately knows that no object crosses the boundaries of R. This means any object $S_j \in A(t)$ satisfies $rank(S_j, t) \leq \epsilon_k^r$. Also, the size of $A(t)$ is still k, and thus the requirements of Definition 1 are met. As an illustration, Figure 6(a) shows the position of query point q, the initial state of the objects, and the constraint R after the Initialization phase.

After initialization, an update from S_i indicates its value has either left or entered R. Answer correctness can be violated, and the **Maintenance Phase** corrects errors by considering three cases:

1. **Case 1:** When an update from $S_i \in X(t) - A(t)$ is received, V_i is no longer within R. Thus S_i is removed from $X(t)$ (Step 1). Correctness 2 is ensured, since any $S_j \in A(t)$ still satisfies $rank(S_j, t) \leq \epsilon_k^r$, and $|A(t)| = k$. Figure 6(b) illustrates this scenario when $S_3 \in X(t) - A(t)$ sends its update to the server.

2. **Case 2:** An update from $A(t)$ indicates that S_i should not be in the answer anymore, since V_i is outside R and there is no longer any guarantee that $rank(S_j, t) \leq \epsilon_k^r$. To ensure correctness, we replace S_i with an item S_j from $X(t) - A(t)$ (Steps 2 and 3) where $rank(S_j, t) \leq \epsilon_k^r$. Figure 6(c) gives an example of this case. As S_1 moves out of R, it is replaced by S_4 in the result set $A(t)$.

   it is possible that the set $X(t) - A(t)$ is empty due to removals caused by repeated application of Step 1 above. In this situation, we can re-execute Initialization phase, but this is expensive as all streams need to be probed. since we have to probe all stream values. Note that R now only contains the objects in $A(t)$. Step 4 looks for candidates to judiciously replace S_i: it expands its search region based on the old ranking scores kept by the server. The search region, R', is formed based on the jth-ranked object from q, where j ranges from $k+r+1$ to n (Step 4(I)(i)-(ii)). The server then queries the clients if their values are within the expanded region R' (Step 4(I)(iii)). If the number of responses, $|U(t)|$, is greater than $r+2$, then $A(t)$ and $X(t)$ will be fixed and the new bound is deployed (Step 4(I)(iv)) (the notation $\min_{r+1, S_l \in U(t)} |V_l - q|$ in (iv)(b) means any object in $U(t)$ ranking $(r+1)th$ or higher in terms of distance from q). The search region expands until we reach V_n, and if still nothing is found, the Initialization phase will be evaluated.

3. **Case 3:** S_i signals that its value is now within R. If the size of $X(t)$ is less than ϵ_k^r, we add S_i to $X(t)$ and the correctness is maintained, since $|X(t)|$ is not larger than ϵ_k^r (Step 6(I)), which is illustrated in Figure 6(d). When $|X(t)| > \epsilon_k^r$, we have to evaluate R so that it contains ϵ_k^r or less objects. To do this, we only need to probe the objects in $X(t)$ (Step 7).

Communication Costs. We state without proof the communication cost in terms of the number of messages between the server and the streams. The Initialization phase needs takes $O(n)$ messages. In the maintenance phase, the running cost is $O(nr)$. In practice the number of messages will be much fewer, because we do not often run into costly situations such as Steps 4 and 7 in the Maintenance phase. As illustrated in Figure 6, in many cases, objects can leave R (a) or enter R (d), without incurring any maintenance cost (corresponding to Steps 3 and 6). As long as the number of objects in $X(t) - A(t)$ is between 0 and r, no maintenance is required.

5 Fraction-based Tolerance

As mentioned earlier, fraction-based tolerance is a different type of "non-value-based" error, and it can be used for both classes of entity-based queries: non-rank-based and rank-based. In Section 5.1, we study a protocol for exploiting fraction-based tolerance for non-rank-based queries. We further extend this protocol to support rank-based queries in Section 5.2.

5.1 Non-rank-based Queries

We now present a protocol for exploiting fraction-based tolerance for range queries, which are non-rank-based queries. Recall that a range query is characterized by a close interval $[l, u]$, where streams with values within this interval are to be reported.

To get more insight into our tolerance protocol, consider a simpler protocol where no tolerance is allowed. This protocol assigns the constraint $[l, u]$ to each stream filter at the beginning. Any violation in a filter has to be reported to the server, so that query answers can be updated correspondingly. Correctness is guaranteed, since essentially each filter evaluates the range query on the stream it is responsible for. We call this protocol *zero-tolerance protocol for non-rank-based query* (**ZT-NRP**).

ZT-NRP reduces the amount of communication since a filter will only emit items when its associated constraint is violated. Consequently, the workload imposed on the processor is reduced. However, this protocol does not exploit any fraction-based tolerance at all – it may generate more updates than necessary. The protocol we describe next utilizes the maximum false positives and false negatives allowed to achieve a better performance.

5.1.1 Exploiting Fraction-based Tolerance

Figure 7 shows the *fraction-based tolerance protocol for non-rank-based queries* (**FT-NRP**). Similar to **RTP**, it has two phases: initialization and maintenance.

Initialization Phase Fraction-based tolerance requires that no more than a fraction ϵ^+ of the query answer (i.e., $A(t)$) should be wrong at any time t. To use this tolerance, the server first captures the states of the streams (Steps 1-3). Out of the $|A(t_0)|$ answers that satisfy the range query, we assign the constraint $[-\infty, \infty]$ to $E^{max+}(t_0)$ filters, and $[l, u]$ to the remaining $|A(t_0)| - E^{max+}(t_0)$ filters (Step 4), where $E^{max+}(t_0)$ is given by $|A(t_0)| \cdot \epsilon^+$ (Equation 3). Filters assigned with $[-\infty, \infty]$ (we call them *false positive filters*), send no updates to the server. If no stream in $A(t)$ replies, we can guarantee that $F^+(t) \leq \epsilon^+$ i.e., the false positive requirement is met. Since $E^{max+}(t_0)$ streams are "shut down", the amount of communication can be reduced significantly. In a sensor network, this also implies savings in battery power since the sensors that are "shut down" are essentially running in the sleep mode. We use n^+, initially equal to $E^{max+}(t_0)$, to denote the current number of false positive filters.

A similar idea can be used to exploit false negative tolerance. Observe that $|Y(t_0)| = |S - A(t_0)|$ streams do not satisfy Q. Among these streams, we select $E^{max-}(t_0)$ of them to install the constraint $[-\infty, \infty]$ (called *false negative filters*), and for the remaining $|Y(t_0)| - E^{max-}(t_0)$ streams, a filter bound of $[l, u]$ is deployed (Step 5). In essence, the false negative filters "turn off" $E^{max-}(t_0)$ streams. When no data are received from any stream in $Y(t_0)$, we guarantee that at any time t, $F^-(t) \leq \epsilon^-$. Using Equations 2,3 and 4, we can derive $E^{max-}(t_0)$ to be $|A(t_0)| \frac{\epsilon^-(1-\epsilon^+)}{1-\epsilon^-}$. We use n^-, initially equal to $E^{max-}(t_0)$, to denote the current number of false negative filters.

After the Initialization Phase is complete, we can guarantee that correctness requirement 1 for fraction-based tolerance is satisfied. That is, if no update is received at time t, $F^+(t) \leq \epsilon^+$ and $F^-(t) \leq \epsilon^-$.

Maintenance Phase Similar to **RTP**, updates can affect the correctness of **FT-NRP**. Assume the server receives an update from stream S_i at time t_u. The new value reported is V_i. Immediately prior to receiving S_i, according to correctness criterion 1, the following must hold (from Equations 3 and 4 respectively):

$$F^+(t) \leq \frac{E^{max+}(t_u)}{|A(t_u)|} \leq \epsilon^+ \tag{11}$$

$$F^-(t) \leq \frac{E^{max-}(t_u)}{|A(t_u)| - E^{max+}(t_u)} \leq \epsilon^- \tag{12}$$

Now assume t is the current time instant where $t \geq t_u$. Depending on the nature of the update, there are two cases to consider.

Case 1: $V_i \in [l, u]$. This means that S_i which was not in the result earlier is now in the result. We handle this update by inserting S_i to $A(t_u)$ (Step 1(I)). As $E^{max+}(t)$ remains unchanged, and $|A(t)|$ becomes $|A(t_u)| + 1$, $F^+(t)$ is at most $\frac{E^{max+}(t_u)}{|A(t_u)|+1}$ (Equation 3), and is thus less than ϵ^+ (Equation 11). Since $E^{max-}(t)$ is also unchanged, we can establish with similar arguments that Equation 12 also holds. Thus correctness 2 is guaranteed.

An important observation is that the values of *both* $F^+(t)$ and $F^-(t)$ drop. This is because the answer quality is improved due to more streams satisfying the query answer. We exploit this fact by using a variable called `count` to keep track of the number of new items inserted to the answer under this scenario (Step 1(II)). Let t_c denote the time when `count` is zero. Then, $|A(t)| = |A(t_c)| +$ `count` for `count`≥ 0. Intuitively, t_c is the time when F^+ and F^- attain their maximum values without violating the correctness requirements. At any time t, if `count`≥ 0, $F^+(t) \leq F^+(t_c)$ and $F^-(t) \leq F^-(t_c)$, a result that we will use next.

Case 2: $V_i \notin [l, u]$. This means S_i satisfied Q immediately after $[l, u]$ was installed to its filter, but it is no longer the answer to Q at time t_u. Step 2(I) removes this "bad answer" from $A(t_u)$. We also decrement `count` by one (Step 2(II)). As explained in case 1, as long as `count` is greater than zero, $F^+(t) \leq F^+(t_c)$ and $F^-(t) \leq F^-(t_c)$. Since $F^+(t_c) \leq \epsilon^+$ and $F^-(t_c) \leq \epsilon^-$, the correctness requirements are met.

When `count` becomes 0, correctness is no longer guaranteed: $|A(t_u)|$, becomes $|A(t_c)| - 1$, and thus $F^+(t_u)$ and $F^-(t_u)$ can be respectively larger than $F^+(t_c)$ and $F^-(t_c)$. Intuitively, there are more items removed from the answer due to **Case 2** than the number of items inserted to the answer due to **Case 1**. To ensure that $F^+(t_u)$ and $F^-(t_u)$ are restored to a "normal level", `Fix_Error` is executed in Step 2(III).

`Fix_Error` improves the degree of answer correctness by consulting streams associated with false positive and false negative filters to update the answer, so as to "compensate" the loss of correctness due to the removal of an answer in Step 2(I). We now discuss how `Fix_Error` works, assuming that both false positive and false negative filters are available (i.e., n^+ and $n-$ are greater than zero).

When n^+ is greater than 0, a stream S_y with a false positive filter is requested to send its value (Step 1(I)). There are two cases, depending on whether V_y is inside $[l, u]$.

1. $\boldsymbol{V_y \in [l, u]}$. S_y is now a true positive. Since S_y was assigned a false positive filter, V_y has already been in $A(t_u)$, and so $|A(t_u)|$ remains unchanged (i.e., $|A(t_c)| - 1$). We then install the $[l, u]$ constraint for S_y to make sure $V_y \in [l, u]$ when no update is received from (Step (II)(i)). In doing so, S_y is no longer a false positive, and so $E^{max+}(t_u)$ is decremented. Thus $F(t_u)$ is now less than $\frac{E^{max+}(t_c)-1}{|A(t_c)|-1}$, which is smaller than $F^+(t_c)$, and the false positive constraint is met. The false negative tolerance constraint is also satisfied, since

$$\begin{aligned} F^-(t_u) &\leq \frac{E^{max-}(t_u)}{|A(t_u)| - E^{max+}(t_u)} \quad \text{(Eqn 4)} \\ &= \frac{E^{max-}(t_c)}{(|A(t_c)| - 1) - (E^{max+}(t_c) - 1)} \\ &\leq \epsilon^- \end{aligned}$$

   Thus both false positive and false negative constraints are satisfied.

2. $\boldsymbol{V_y \notin [l, u]}$. S_y is now a true negative. Since S_y no longer satisfies Q, We remove S_y from $A(t_u)$ (Step 1(III)), and $|A(t_u)|$ becomes $|A(t_c)| - 2$. Since $E^+(t_u)$ is also decremented, $F^+(t_u)$ is now less than $\frac{E^{max+}(t_c)-1}{|A(t_c)|-2}$. Since we have assumed that ϵ^+ cannot be larger than 0.5, $\frac{E^{max+}(t_c)-1}{|A(t_c)|-2}$ cannot be larger than $\frac{E^{max+}(t_c)}{|A(t_c)|}$, and is therefore less than ϵ^+.

   However, $F^{max-}(t_u)$ is now at most $\frac{E^{max-}(t_c)}{(|A(t_c)|-2)-(E^{max+}(t_c)-1)}$, which can be more than ϵ^-. To remedy this, we pick one stream associated with a false negative filter (Step 2(I)). If $V_z \in [l, u]$, then we include S_z into the answer (Step 2(II)). We also install $[l, u]$ to the filter of S_z (Step 2 (III)). Now $|A(t_u)|$ is increased to $|A(t_c)| - 1$, and $F^-(t_u)$ is at most $\frac{E^{max-}(t_c)-1}{(|A(t_c)|-1)-(E^{max+}(t_c)-1)}$, which is smaller than ϵ^-. Further, $F^+(t_u)$ is now at most $\frac{E^{max+}(t_c)-1}{|A(t_c)|-1}$, which is still less than ϵ^+. Thus correctness 2 is met.

   On the other hand, if $V_z \notin [l, u]$, $|A(t_u)|$ and $E^{max+}(t_u)$ remain unchanged and thus the false positive constraint is still satisfied. Since $E^-(t_u)$ is at most $E^{max-}(t_c) - 1$, $F^-(t_u)$ is at most $\frac{E^{max-}(t_c)-1}{(|A(t_c)|-2)-E^{max+}(t_c)}$, which is smaller than ϵ^- because $\epsilon^- \leq 0.5$. Hence correctness 2 is also met.

We skip the correctness proofs for special cases: (1) $n^+ = 0 \wedge n^- > 0$ and (2) $n^+ > 0 \wedge n^- = 0$. We also remark that when both n^+ and n^- become zero, it implies both the false positive and negative filters are replaced by the $[l, u]$ constraint. Hence the false positive and negative constraints are met, and this protocol reduces to **ZT-NRP**. It may then be necessary to re-execute the Initialization Phase in order to have the tolerance exploited.

Communication Costs. The Initialization Phase requires $O(n)$ messages, while the maintenance Phase generates at most five messages when both false positive and false negative filters have to be consulted by `Fix_Error`. However, since no messages are required as long as `count` is zero, the actual maintainence cost is low as verified by our experiments.

5.2 Rank-based queries

We now present the fraction-based tolerance protocol for k-NN query, a typical rank-based query. Our solution is based on the work in Section 5.1. In particular, we transform a k-NN query to a range query, and then adopt the fraction-based protocol designed for range queries. Let us see how this is done in detail.

5.2.1 Transforming k-NN Query to Range Query

A k-NN query can be viewed as a range query: if we know the bound R that encloses the k-th nearest neighbor of the query point q, then any objects with values located within R will be an answer to the k-NN query.

We can use this idea to design a filter scheme for k-NN query (with zero-tolerance). The protocol, called **ZT-RP**, is illustrated in Figure 8. The Initialization Phase computes R which tightly encloses k nearest neighbor, and then distributes R to all the stream filters. Then if no responses are received from the streams, the server is assured that all k objects are within R, and they are still the k nearest neighbors of q. Since no error is allowed, if any object enters or leaves R, we have to recompute R so that R encloses the k nearest objects. The Maintenance Phase in Figure 8 illustrates how R is maintained.

The main drawback of this simple protocol is that it is very sensitive to updates when an object's value crosses R. Each time R is crossed, it has to be recomputed, and its new value is announced to every stream!

Next, we describe how this situation can be improved by exploiting fraction-based tolerance.

5.2.2 Using FT-NRP for k-NN Query

In the last section, we discussed how to model a k-NN query as a range query for the purpose of constraint deployment. Recall that the definition of fraction-based tolerance is the same for k-NN query and range query. Therefore, to develop a fraction-based tolerance protocol for a k-NN query, it seems that we can simply transform a k-NN query to a range query, and then directly apply the protocol developed for range query (**FT-NRP**). Unfortunately, this does not work without some minor changes. In particular, we may not use the values of ϵ^+ and ϵ^- specified by the k-NN query to parametize **FT-NRP** directly.

To understand why, let ρ^+ and ρ^- be the maximum false positive tolerance and maximum false negative tolerance used by **FT-NRP** to answer a k-NN query (with tolerance ϵ^+ and ϵ^-). Let R be the smallest region that initially bounds the kth-ranked object and thus contains k objects. Similar to the initialization phase of **FT-NRP**, for objects lying in R we apply false positive filters to $k \cdot \rho^+$ streams, and R to the remaining $k \cdot (1-\rho^+)$ filters. For streams with values outside R, we apply false negative filters to $k \cdot \rho^-$ streams, and R as the constraint to the remaining filters. Let us now examine how to set ρ^+ and ρ^- appropriately.

Meeting false positive requirement. Suppose R encloses the k nearest objects of q. Let S_1 be part of the answer set, and $V_1' \in R$ is the value of S_1 last reported to the server. Hence S_1 is one of the k nearest neighbors. If S_1 is associated with a false positive filter, the new value of S_1, i.e., V_1, may not be located within R. Consider the situation in Figure 9. Suppose there exists a stream S_2 such that $V_1 < V_2$. Then S_1 is no longer a correct answer, since S_2 now ranks higher and pushes the rank of S_1 to $k+1$. Therefore S_1 becomes a false positive. Since we can have at most $|A(t)| \cdot \rho^+$ streams assigned with false positive filters, in the worst case $|A(t)| \cdot \rho^+$ false positives are generated in this manner.

Another kind of false positive is inflicted by false negative filters. Suppose S_4, being ranked the k-th and lies within R, is part of the answer set. Also assume that S_3 is associated with a false negative filter, whose last reported value, V_3', does not lie within R. As illustrated in Figure 9, when the new value of S_3, i.e., V_3, is within R, the rank of S_3 becomes k or higher. During this process, the rank of S_4 is dropped to $k+1$ and hence S_4 becomes a false positive. Since we can assign false negative filters to at most $E^{max-}(t) = k \cdot \rho^-$ streams (c.f. Equations 4 and 5), in the worst case $k \cdot \rho^-$ false positives are created in this way.

Combining these two types of false positives, the total number of false positives is $|A(t)| \cdot \rho^+ + k \cdot \rho^-$, where $|A(t)|$ is less than $\frac{k}{1-\epsilon^+}$ (Equation 7). Also, the user cannot tolerate more than $|A(t)| \cdot \epsilon^+$ false positives, which has a minimum value of $k \cdot (1-\epsilon^-) \cdot \epsilon^+$ (Equation 9). Therefore,

$$\begin{aligned} |A(t)| \cdot \rho^+ + k \cdot \rho^- &\leq |A(t)| \cdot \epsilon^+ \\ \Rightarrow \rho^- &\leq \frac{\rho^+}{\epsilon^+ - 1} + (1-\epsilon^-)\epsilon^+ \quad (13) \end{aligned}$$

Equation 13 dictates the possible values of ρ^+ and ρ^- for satisfying the false positive requirement.

Meeting false negative requirement. We have two types of false negatives for a kNN query. Again assume R encloses the k nearest objects. As shown in Figure 9, the first type of false negatives is caused by streams like S_3, whose last reported value V_3' is not within R, and is assigned with false negative filters. Later its new value V_3 is within R and its rank is raised to k or higher. Unfortunately the server does not know about this, and so S_3 becomes a false negative. The number of false negatives generated this way is at most $k\rho^-$, the maximum number of false negative filters. The second type is caused by streams with false positive filters like S_1. Again S_1 was among the top-k objects since its last reported value V_1' is within R. However its new value V_1 is less than V_2, so S_2 ranks k or higher. The server does not know about this, since S_1 does not update its position. The maximum number of false negatives generated this way is thus $|A(t)| \cdot \rho^+$, the maximum number of false positive filters assigned. Since the maximum number of false negatives for k-NN query is given by $k \cdot \epsilon^-$, the sum of the two kinds of false negatives, $k\rho^-$ and $|A(t)| \cdot \rho^+$, must be less than $\leq k\epsilon^-$. Equation 7 simplifies this to:

$$\rho^- \leq \frac{\rho^+}{\epsilon^+ - 1} + \epsilon^- \qquad (14)$$

Guaranteeing correctness. To make sure both false positives and false negatives are met, we combine Equations 13 and 14 so that the following is achieved:

$$\rho^- \leq \frac{\rho^+}{\epsilon^+ - 1} + \min((1-\epsilon^-)\epsilon^+, \epsilon^-) \qquad (15)$$

Essentially, when the user-defined constraints for rank-based query (i.e., ϵ^+ and ϵ^-) are implemented using the non-rank-based error protocol, the values of ρ^+ and ρ^- so set must satisfy Equation 15. To maximize the degree of tolerance exploited, the values of ρ^+ and ρ^- should be maximized according to the following equation:

$$\rho^- = \frac{\rho^+}{\epsilon^+ - 1} + \min((1-\epsilon^-)\epsilon^+, \epsilon^-) \qquad (16)$$

In our experiments, the values of ρ^+ and ρ^- are derived from Equation 16.

5.2.3 Fraction-based Tolerant k-NN Query

Once the values of ρ^+ and ρ^- are rightly set, we can extend **FT-NRP** to exploit the fraction-based tolerance of k-NN queries. The corresponding protocol, called **FT-RP**, differs from **FT-NRP** in two aspects:

1. Unlike a range query with a fixed bound $[l, u]$, the "range" of k-NN query is defined by R – the tightest bound that contains the k-th nearest neighbor. Thus, **FT-RP** first finds R before running the initialization phase of **FT-NRP**. Notice that the filter constraint R so calculated will not be changed even when R contains more or less than k objects – except when the conditions described next are met. Essentially, we use R only as an estimate of the k nearest neighbors.

2. An additional requirement for the answer $A(t)$ of rank-based query is that $k(1-\epsilon^-) \leq |A(t)| \leq \frac{k}{1-\epsilon^+}$ (Equations 7 and 9). Initially $|A(t)|$ is equal to k, but as time goes by, the number of items in $A(t)$ will be increased (decreased) when an object enters (exits) R. Intuitively, when $|A(t)|$ exceeds $\frac{k}{1-\epsilon^+}$, there are too many objects in R – that is, R is "too loose". Similarly, when $|A(t)|$ drops below $k(1-\epsilon^-)$, there are not enough objects in R, implying that R is "too tight". In either case, R is no longer an appropriate filter for the k-NN query. Thus similar to the maintenance phase of **ZT-RP**, a new bound has to be found to enclose the k-nearest neighbors.

The advantage of **FT-RP** over **ZT-RP** is easily explained – it does not have to recompute and broadcast R each time an object enters or leaves R, but only when $A(t)$ drops below $k(1-\epsilon^-)$ or exceeds $\frac{k}{1-\epsilon^+}$. This is because **FT-RP** exploits tolerance, which is not allowed by **ZT-RP**.

6 Experimental Results

We have implemented the non-value-based tolerance protocols. In this section we present our experimental results.

We use CSIM 18 [25] and Tcl scripting tools [3] to develop our simulation programs. We model the environment in Figure 3, where we simulate data streams as well as a continuous query being executed in the system for a certain period of time. We study the performance of the tolerance-based protocols over various degrees of tolerance, and compare with (1) the case when no filter is used at all, and (2) filter protocols with no tolerance allowed (i.e., **ZT-NRP** and **ZT-RP**). The performance metric for measuring communication costs is the number of maintenance messages required during the lifetime of the query[1]. In the rest of this section, we will present two sets of results, based on real and synthetic data.

[1]When no filter is used, a "maintenance message" is essentially an update message from a stream source

6.1 TCP Data

In the first set of experiments, we test the efficiency of our protocols based on TCP traces [16]. The experiment models the scenario where an Internet host monitors network traffic from 800 ISPs. We assume a software is installed at each ISP that implements our filter bound protocols. The dataset contains 30 days of wide-area traces of TCP connections, capturing 606,497 connections. We model the connections whose IP addresses share the same 16-bit prefix as data from the same ISP. We assume 800 data streams, and use the "number of bytes sent" field in each trace as a data value. .

Figure 10 shows the result of varying the rank-based tolerance r for some values of k. We observe that for different values of k, the performance improves as r increases. This indicates **RTP** is able to exploit tolerance effectively. Also notice that at $k = 30$ and $r = 0$, the performance is worse than when no filter is used at all. This is because at $r = 0$, the bound R needs to be recomputed frequently and many maintenance messages are generated as a result.

Next, we examine how well **FT-NRP** exploits fraction-based tolerance for range queries. In Figure 11, we observe that the number of messages decrease as ϵ^+ and ϵ^- increase. We do not show the result when no filter is used because it has a very high cost. We also examine the scalability of **FT-NRP** in Figure 12, where we can see that the protocol in general scales well. We also observe that for a larger number of streams, the performance gains more by using higher tolerance values.

6.2 Synthetic Data

In the second set of results, we verify the effectiveness of protocols by using a synthetic data model, which gives us better control over data behavior. We assume 5000 data streams in this model, and data values are initially uniformly distributed in the range $[0, 1000]$. The time between each data item is generated follows an exponential distribution with a mean of 20 time units. When a new data value is generated, its difference from the previous value follows a normal distribution with a mean of 0 and standard deviation (σ) of 20.

We first examine the performance of **FT-NRP** for a range query with $l = 400$ and $u = 600$, over a wide range of ϵ^+ and ϵ^- values. As shown in Figure 13, **FT-NRP** is able to exploit tolerance effectively.

Now let us look at Figure 14 that illustrates the effect of data fluctuation (i.e., the amount of difference between two adjacent values in a data stream) on **FT-NRP**. As σ increases, **FT-NRP** generates more messages. When a data value changes abruptly, it has a higher chance of violating the filter bound constraint and generate an update.

In another experiment, we explore how the performance of **FT-NRP** is affected by the assignment of false positive and false negative filters during Steps 4 and 5 of the initialization phase. We compare two heuristics: (1) *random* – streams are randomly selected to assign $[-\infty, \infty]$ and $[\infty, \infty]$ constraints, and (2) *boundary-nearest* – only streams whose values are closest to the user-defined range $[l, u]$ are assigned the $[-\infty, \infty]$ and $[\infty, \infty]$ constraints. Figure 15 shows the result. The *boundary-nearest* heuristic outperforms *random* because streams with values close to $[l, u]$ are likely to cross the boundary of $[l, u]$, and so by assigning false positive/negative filters to them, the number of updates generated can be reduced. As the amount of tolerance increases, the difference is more pronounced, because more false positive/negative filters are available, and they are more appropriately placed by *boundary-nearest* heuristic than by *random* heuristic.

The final result presented is the performance of **ZT-RP** and **FT-RP** over different values of k. As shown in Figure 16, for k equals to 60 or 100, the number of messages drops significantly with a slight increase in tolerance. This is because the bound R for enclosing k nearest objects is not "tight", and objects can cross R without requiring R to be recomputed and assigned as a new constraint to the streams. With zero tolerance, however, R virtually changes everytime an object crosses it. We also note that the protocol does not perform well at k of 20 and tolerance of 0.1. With a small values of k and tolerance, the number of false positive and negative filters allocated is very limited. This little benefit of tolerance cannot overcome the maintenance overhead. We remark that **FT-RP** is not suitable for small values of k and tolerance.

7 Conclusions

The performance of data stream management systems can often be improved by allowing some tolerance. In this paper we studied how non-value tolerance can be exploited for entity-based queries. We presented simple protocols to incorporate rank-based and fraction-based tolerance into both rank-based and non-rank-based queries. Through testing with real and simulation data, we showed that our protocols are effective in reducing communication costs. The concepts of our protocols can be extended to multiple dimensions.

There are a number of interesting avenues for future work. One issue is to see how other entity-based queries, such as skyline and join queries, can be handled in adaptive filters. It is worthy to examine how existing data stream algorithms can be modified to support non-value tolerance. We also plan to extend these protocols for multiple queries.

References

[1] A.Arasu, B.Babcock, S.Babu, J.McAlister, and J.Widom. Characterizing memory requirements for queries over continuous data streams. *ACM Trans.Database Syst.*, 29(1), 2004.

[2] A.Arasu, B.Babcock, S.Babu, M.Datar, K.Ito, R.Motwani, I.Nishizawa, U.Srivastava, D.Thomas, R.Varma, and J.Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26, March 2003.

[3] ActiveState. Tcl. http://www.tcl.tk.

[4] A.Deligiannakis, Y.Kotidis, and N.Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *Proc. EDBT*, 2004.

[5] A.Jain, E.Chang, and Y.Wang. Adaptive stream resource management using kalman filters. In *Proc. ACM SIGMOD*, 2004.

[6] B.Babcock and C.Olston. Distributed top-k monitoring. In *Proc. ACM SIGMOD*, 2003.

[7] B.Babcock, S.Babu, M.Data, R.Motwani, and J.Widom. Models and issues in data stream systems. In *Proc. ACM PODS*, 2002.

[8] R.Motwani B.Babcock, M.Datar and L.O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proc. PODS*, 2003.

[9] B.Cui, H.Shen, J.Shen, and K.Tan. Exploring bit-difference for approximate knn search in high-dimensional databases. In *Australasian Database Conference*, 2005.

[10] C.Olston, J.Jiang, and J.Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. of ACM SIGMOD*, 2003.

[11] D.Abadiand, D.Carneyand, U.Cetintemel, M.Cherniack, C.Convey, C.Erwin, E.Galvez, M.Hatoun, J.Hwang, A.Maskey, A.Rasin, A.Singer, M.Stonebraker, N.Tatbul, Y.Xing, R.Yan, and S.Zdonik. Aurora: A data stream management system. In *Proc. ACM SIGMOD*, 2003.

[12] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. VLDB*, 2004.

[13] G.Iwerks, H.Samet, and K.Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *Proc. VLDB*, 2003.

[14] G.Kollios, D.Gunopulos, and V.Tsotras. Nearest neighbor queries in a mobile environment. In *Proc. STDBM*, 1999.

[15] J.Ni and C.V.Ravishankar. Probabilistic spatial database operations. In *Proc. SSTD*, 2003.

[16] Lawrance Berkeley National Laboratory. The Internet Traffic Archive, USA. http://ita.ee.lbl.gov.

[17] M.Cherniack, H.Balakrishnan, M.Balazinska, D.Carney, U.Cetintemel, Y.Xing, and S.Zdonik. Scalable distributed stream processing. In *Proc. CIDR*, 2003.

[18] M.Greenwald and S.Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proc. ACM PODS*, 2004.

[19] N.Koudas, B.Ooi, K.Tan, and R.Zhang. Approximate NN queries on streams with guaranteed error/performance bounds. In *Proc. VLDB*, 2004.

[20] P.Gibbons and Y.Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD*, 1998.

[21] R.Cheng, D.Kalashnikov, and S.Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. ACM SIGMOD*, 2003.

[22] S.Babu, R.Motwani, K.Munagala, I.Nishizawa, and J.Widom. Adaptive ordering of pipelined stream filters. In *Proc. ACM SIGMOD*, 2004.

[23] S.Ganguly, M.Garofalakis, and R.Rastogi. Processing set expressions over continuous update streams. In *Proc. ACM SIGMOD*, 2003.

[24] S.Khanna and W.C.Tan. On computing functions with uncertainty. In *ACM PODS*, 2001.

[25] Mesquite Software. Csim 19. http://www.mesquite.com.

[26] S.Vrbsky and J.Liu. Producing approximate answers to set- and single-valued queries. *Journal of Systems and Software*, 27(3), 1994.

[27] V.Hristidis, L.Gravano, and Y.Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proc. VLDB*, 2003.

[28] V.Poosala and V.Ganti. Fast approximate query answering using precomputed statistics. In *Proc. ICDE*, 1999.

[29] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3), 1999.

[30] Y.Yao and J.Gehrke. Query processing for sensor networks. In *Proc. CIDR*, 2003.

```
Initialization (at time $t_0$)
1. request all streams to send their values
2. $A(t_0) \leftarrow \{S_i | rank(S_i, t_0) \leq k\}$
3. $X(t_0) \leftarrow \{S_i | rank(S_i, t_0) \leq \epsilon_k^r\}$
4. execute Deploy_bound($t_0$)

Maintenance
Upon receiving a new update $V_i$ from stream $S_i$ at time $t$ ,
Case 1:  $S_i \in X(t) - A(t)$ /* $V_i$ “leaves” $R$ */
 1. remove $S_i$ from $X(t)$

Case 2:  $S_i \in A(t)$ /* $V_i$ “leaves”$R$ */
 2. remove $S_i$ from $A(t)$
 3. if $|X(t)| > k$ then
    (I)insert to $A(t)$ an object in $X(t) - A(t)$ with highest rank
 4. else /* $R$ only contains $|A(t)| = k - 1$ objects */
    (I)for $j = k + r + 1$ to $n$ do
       (i) Let $d'$ be $|V_j - q|$ s.t. $rank(S_j, t_0) = j$
       (ii) $R' \leftarrow [q - d', q + d']$
       (iii)$U(t) \leftarrow \bigcup\{S_l | V_l \in R' \wedge S_l \notin A(t)\}$
       (iv) if $|U(t)| > r + 2$ then
          a.$A(t) \leftarrow A(t) \cup \{S_l | S_l \in U(t) \wedge |V_l - q| = \min_{S_l \in U(t)} |V_l - q|\}$
          b.$X(t) \leftarrow A(t) \cup \{S_l | S_l \in U(t) \wedge |V_l - q| \in \min_{r+1, S_l \in U(t)} |V_l - q|$
          c.execute Deploy_bound($t$)
          d.quit
 5. execute Initialization

Case 3:  $S_i \in S - X(t)$ /* $V_i$ “enters” $R$ */
 6. if $|X(t)| < \epsilon_k^r$ then
    (I) insert $S_i$ to $X(t)$
 7. else /* Evaluate new $R$ */
    (I) $\forall S_i \in X(t)$, request for current values $S_i$
    (II) $A(t) \leftarrow \{S_i | rank(S_i, t) \leq k\}$
    (III)$X(t) \leftarrow \{S_i | rank(S_i, t) \leq \epsilon_k^r\}$
    (IV) execute Deploy_bound($t$)

Deploy_bound($t$)
 1. $S_x \leftarrow S_i$ where $rank(S_i, t) = \epsilon_k^r$
 2. $S_y \leftarrow S_i$ where $rank(S_i, t) = \epsilon_k^r + 1$
 3. $d \leftarrow \frac{|V_x - q| + |V_y - q|}{2}$
 4. $\forall S_i \in S$, deploy constraint $[q - d, q + d]$
```

Figure 5: Maintaining rank-based-tolerance at the server by **RTP**.

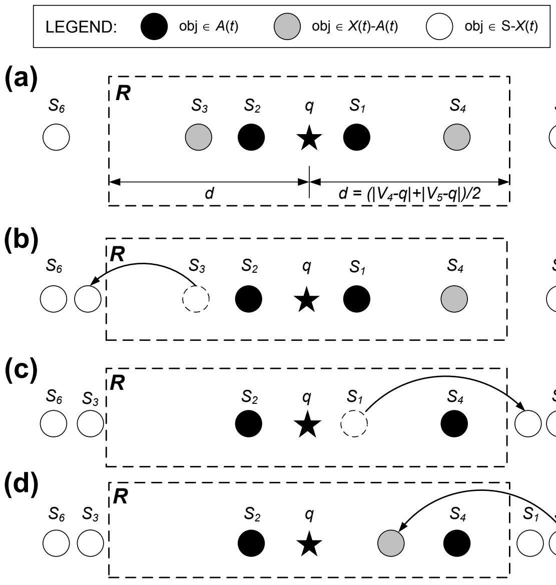

Figure 6: Illustrating the rank-based protocol for a k-NN query with $k = 2$ and $r = 2$.

```
**Initialization** (at time $t_0$)
    Let $count = 0$, $n^+ = |A(t_0)|\epsilon^+$, $n^- = |A(t_0)|\frac{\epsilon^-(1-\epsilon^+)}{1-\epsilon^-}$
    1. request all streams to send their values
    2. $A(t_0) \leftarrow \{S_i | V_i \in [l,u]$ at time $t_0\}$
    3. $Y(t_0) \leftarrow S - A(t_0)$
    4. For streams in $A(t_0)$,
        (I) install $[-\infty,\infty]$ to $n^+$ filters
        (II) install $[l,u]$ to remaining $|A(t_0)| - n^+$ filters
    5. For streams in $Y(t_0)$
        (I) install $[\infty,\infty]$ to $n^-$ filters
        (II) install $[l,u]$ to remaining $|Y(t_0)| - n^-$ filters

**Maintenance**
    Upon receiving a new update, $V_i$ from stream $S_i$,
    1. **if** $V_i \in [l,u]$ **then**
        (I) insert $S_i$ to $A(t)$
        (II)count $\leftarrow$ count $+ 1$
    2. **else**
        (I) remove $S_i$ from $A$
        (II) **if** count $> 0$ **then** count $\leftarrow$ count $- 1$
        (III)**else execute** Fix_Error

Fix_Error
    1. **if** $n^+ > 0$ **then**
        (I) request value from $S_y$ with $[-\infty,\infty]$ constraint
        (II) **if** $V_y \in [l,u]$ **then**
            (i) install $[l,u]$ for the filter of $S_y$
            (ii)$n^+ \leftarrow n^+ - 1$
            (iii)**quit**
        (III)remove $S_y$ from $A(t)$
    2. **if** $n^- > 0$ **then**
        (I) request value from $S_z$ with $[\infty,\infty]$ constraint
        (II) **if** $V_z \in [l,u]$ **then** insert $S_z$ to $A(t)$
        (III)install $[l,u]$ for the filter of $S_z$
        (IV) $n^- \leftarrow n^- - 1$
```

Figure 7: Maintaining fraction-based tolerance of range query at the server by **FT-NRP**.

Initialization

```
    1. request from the clients for their latest values
    2. compute the region $R$ that includes $k$ neighbors of $q$
    3. return the $k$ neighbors to the user
    4. broadcast $R$ to all streams as the new constraint
```

Maintenance

```
    1. Upon receiving an update that indicates $V_i \in R$,
        (I) request all streams inside $R$ to send their values
        (II) re-evaluate the new $k$-NN
        (III) return the new answer to the user
        (IV) broadcast the new bounds to all streams
    2. Upon receiving an update that indicates $V_i \notin R$,
        (I) enlarge $R$ to $R'$ where $V_i \in R'$
        (II) request all streams inside $R'$ to send their values
        (III)compute region $R''$ to include $k$ neighbors of $q$
        (IV) return the new answer to the user
        (V) broadcast $R''$ to all streams as the new constraint
```

Figure 8: Maintaining correctness of rank-based query with zero tolerance (**ZT-RP**).

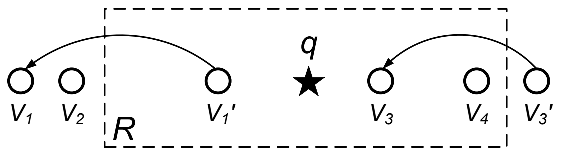

Figure 9: Illustrating how false positives and false negatives are generated for a k-NN query.

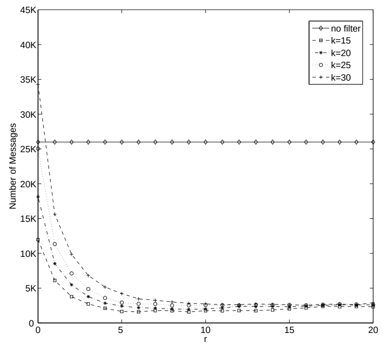

Figure 10: Effect of r on **RTP**.

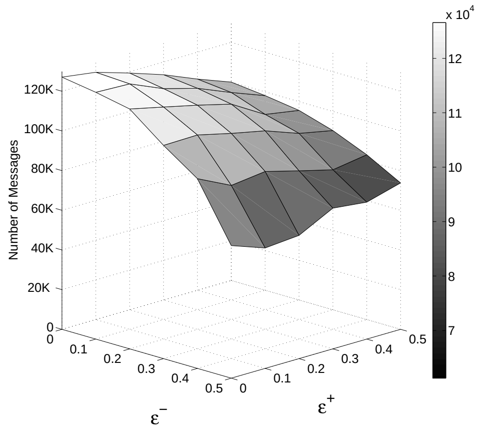

Figure 11: Effect of ϵ^+ and ϵ^- on **FT-NRP**.

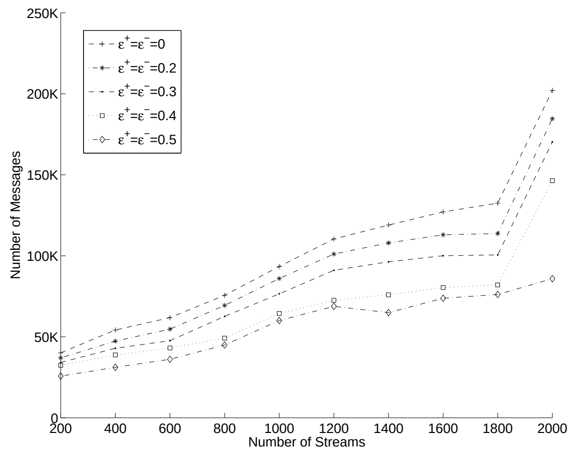

Figure 12: Scalability of **FT-NRP.**

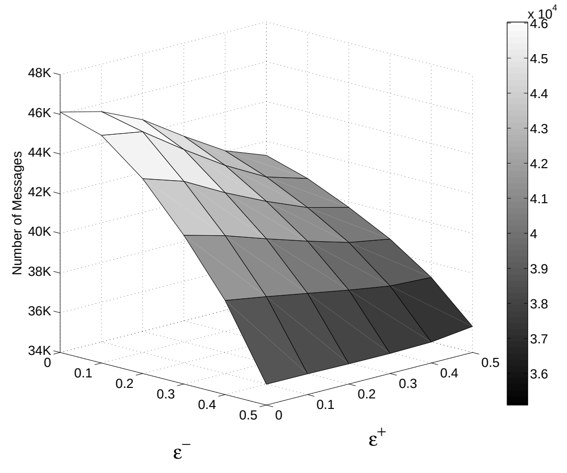

Figure 13: Effect of ϵ^+ and ϵ^- on **FT-NRP**.

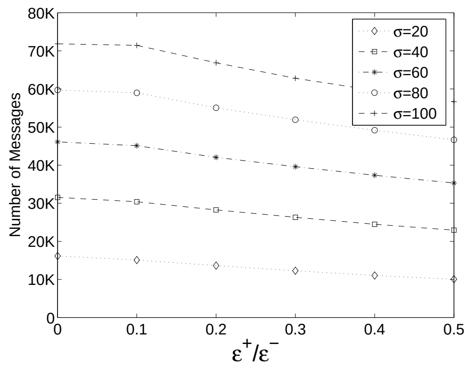

Figure 14: Effect of data fluctuation on **FT-NRP.**

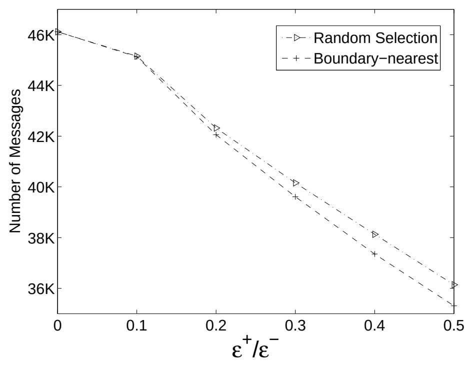

Figure 15: Selection heuristics for **FT-NRP.**

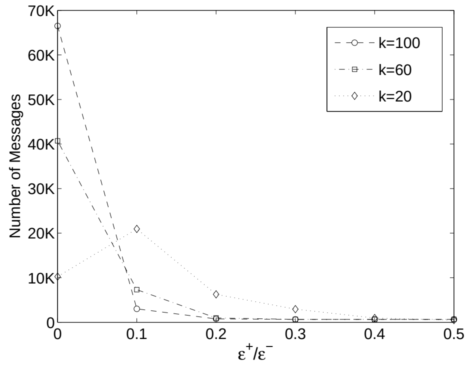

Figure 16: Effect of ϵ^+ and ϵ^- on **FT-RP.**