

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

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

[†] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong.
Email: cskcheng@comp.polyu.edu.hk

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

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

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., nearest neighbor) with non-value-based error tolerance. 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 [2];

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. 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. The number of streams could be large and they are continuously reporting updates. A stream server could thus be crippled by the large volume of data, slowing its response to standing queries that require real-time processing [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. Examples for which controlled query errors are acceptable include wide-area resource accounting and load balancing in replicated servers. Several efforts (e.g., see [18, 9]) have been attempted to produce approximate answers to achieving better overall performance. In particular, intelligent protocols are proposed in [17, 12] to wisely control when streams should report updates. The goal 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.

Most filter-bound-based approximation techniques assume that a maximum tolerable error is specified by a *numerical value*. Consider a network of sensors

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

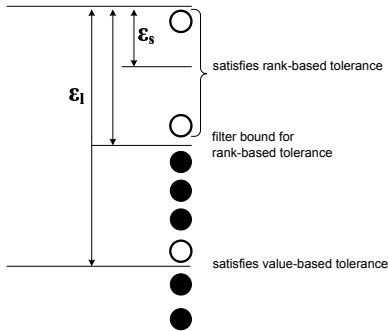


Figure 1: Value-based and rank-based tolerances.

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} .

However, the problem of the above strategy is that choosing a numerical error tolerance is unintuitive. As an example, in a typical location-based application a user can inquire about his closest neighbor. Specifying a numerical error tolerance requires some knowledge about the relative distances or spread of the objects. (Should the tolerance be one meter or 100 meters?) In a sensor network, for instance, various kinds of data like humidity, temperature, UV-index can be collected [7]. 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., images), a numerical, or *value-based error*, could be difficult to specify.

Moreover, a bad choice of the numerical error tolerance may significantly weaken the value of a query. If ϵ is set too large, the returned stream could rank far from the true maximum (see Figure 1 for $\epsilon = \epsilon_l$). To solve this problem, a user then has to be careful not to set ϵ too large. Unfortunately, unless the user has 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 a small filter bound (e.g., ϵ_s) is too small to capture the tolerance.

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 translating

a rank-based tolerance to a filter bound could also be easier. For example, 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.

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 [3]. Typical examples of entity-based queries include range queries and k -nearest neighbor queries, which are common in applications such as location monitoring [21], sensor networks [7] and computer-aided manufacturing (CAM) systems [14]. Observe that the user of an entity-based query is *not* concerned about the numerical value of the answer. Thus a *non-value-based* tolerance is more intuitive for entity-based queries.

While most previous works in filter bound applications assume *value-based* queries (e.g., aggregate queries), in this paper we study extensively two different classes of entity-based queries, namely, rank-based queries and non-rank-based queries. 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* [16]. 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 [10].) 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 filter-bound protocols that reduce communication costs between the server and stream sources, and consequently, reduce server load. As we will also see later, our fraction-based tolerance protocols would instruct some stream sources to be shut down. This can be potentially

beneficial for sensors with limited battery power. 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 can be generalized to higher dimension cases. To summarize, 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
- Examine the effectiveness of our 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

Due to the high-volume and continuous nature of data streams, the goal of a stream management system is to conserve system resources such as memory [1], computation [14, 18, 9] and communication costs [8, 17, 12]. 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. Also, they are mostly applicable to value-based queries only. Our work investigates the possibility of exploiting non-value-based tolerance for continuous entity-based queries. Figure 2 illustrates our contributions in more details.

The idea of using adaptive filters in which filter bounds are installed to reduce communication costs was first proposed in [17]. However, that paper only considers value-based tolerance over aggregate queries such as average and minimum. Babcock et al. [2] 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. [12] 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

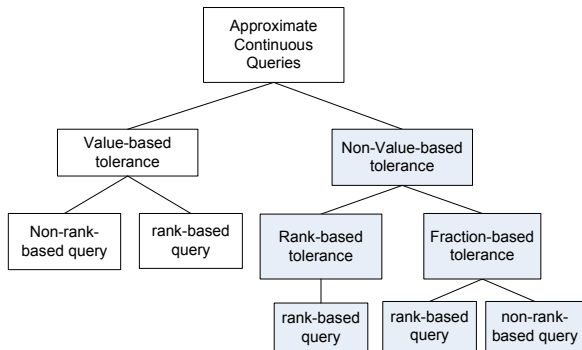


Figure 2: Our contributions (shaded).

in [6]. Our work distinguishes from theirs 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 [14]. Notice that k -NN queries are more general than top- k queries studied in [12].

The classification of queries into value-based and entity-based has been proposed in [3]. 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. [20] studied approximate answers for set-valued queries, where a query answer contains a set of objects. They propose that an exact answer E can be approximated by two sets: a *certain set* C which is a subset of E , and a *possible set* P such that $C \cup P$ is a superset of E . Khanna et al. [13] 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. [5] proposed precision and recall as a quality metric for approximating 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, which has not been addressed before.

The idea of viewing a k -NN query as a range query was proposed by Iwerks et al. [11]. 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 the system model and query types. We also formally define the concept of non-value-based tolerances.

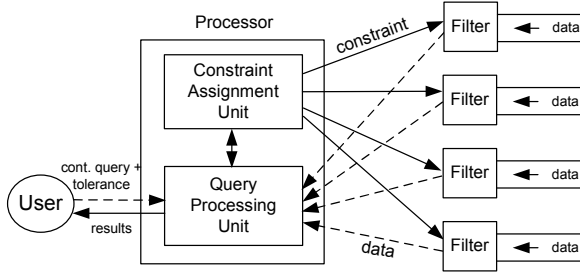


Figure 3: Stream Management System Model.

3.1 System Model

We assume a distributed stream management model similar to those described in [2, 17, 12]. The system consists of a set $S = \{S_1, S_2, \dots, S_i, \dots, S_n\}$ of n data streams with Stream S_i reporting a value $V_i \in \mathbb{R}$. 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 updates are reported to the server. A filter constraint is a closed interval $[l_i, u_i]$, where $l_i, u_i \in \mathbb{R}$. 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 reported.

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 *entity-based queries*, a broad class of queries that return names or identifiers of objects as answers [3]. We further classify entity-based queries into *rank-based queries* and *non-rank-based queries*:

(1) **Rank-based query.** Given a number $k \in \mathbb{N}$ (k is called the *rank requirement*), a rank-based query returns identifiers of objects that rank k th or above. Thus the relative ranking of data items is important to the query answer. Examples include k -nearest-neighbor (or k -NN) query where the objects with the k shortest distances to a query point q are reported [11, 14]; and top- k queries, where answers with the k -highest ranking scores are returned [2]. In this paper we use k -NN queries as an example of how fil-

ter bound protocols are applied, since it is common in streaming systems like computer-aided product-line monitoring systems, mobile environments, and network traffic monitoring [11, 14, 5]. Note that a k -NN query can be easily transformed to a k -minimum or k -maximum query, by setting 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 [21] and sensor networks [7]. A range query is specified by an interval $[l, u]$. Streams whose values fall within $[l, u]$ 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 others.

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 is 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 is the third maximum, but not anything further than that. A rank-based tolerance is important when a large error in ranking of answers is not desirable. For example, in a distributed system, requests from different users possess various priorities, and the system should process jobs with the highest priority values.

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 $rank$ is defined based on the differences from the query point.

Definition 1 Rank-based Tolerance. *Given a rank-based query Q with rank requirement k , an answer set $A(t)$ returned at time t , and a maximum rank tolerance $\epsilon_k^r = k + r$ where $r \in \mathbb{Z}^+$, 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.

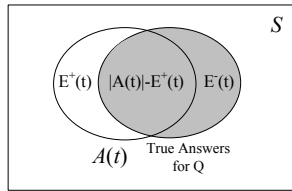


Figure 4: $A(t)$, $E^+(t)$ and $E^-(t)$.

3.4 Fraction-based Tolerance

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 who enter a danger zone, and it may be acceptable that 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 [14], and fraction-based tolerance can be used to measure the quality of results [5].

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

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

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

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 $F^-(t) \leq \epsilon^-$.

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. We make this assumption

because in most scenarios, users are not interested in results with more incorrect answers than correct ones. Also, this assumption is required for guaranteeing the correctness of our protocols.

For notational convenience, we use $E^{max+}(t)$ to denote the maximum number of streams that can 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

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

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

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

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

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,

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

This implies

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

$$|A(t)| \leq 2k. \quad (8)$$

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:

$$|A(t)| \geq k(1 - \epsilon^-) \quad (9)$$

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

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

We are now ready to describe our protocols that guarantee query correctness with specific tolerance constraints. 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*. Similar to [2], there are two 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.

Section 4 describes how to exploit rank-based-tolerance for rank-based queries. In Section 5 we exploit fraction-based-tolerance for both rank-based and non-rank-based queries.

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 cost. Its main idea is to maintain a closed 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 $(k+r+1)$ st 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 col-

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 both $A(t)$ and $X(t)$

3. **if** $A(t) \subset X(t)$ **then**

(I)insert into $A(t)$ 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) $U(t) \leftarrow \bigcup \{S_i | V_i \in R' \wedge S_i \notin A(t)\}$

(iv) **if** $|U(t)| \geq 2$ **then**

a. $A(t) \leftarrow A(t) \cup \{S_i | S_i \in U(t) \wedge |V_i - q| = \min_{S_i \in U(t)} |V_i - q|\}$

b. $X(t) \leftarrow A(t) \cup \{S_i | S_i \in U(t) \wedge |V_i - q| \in \min_{r+1, S_i \in U(t)} |V_i - q|\}$

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 into $X(t)$

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: The **RTP** algorithm (at the server side).

lects 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. It can be shown easily that Correctness 1 is enforced [4]. 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:

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.

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

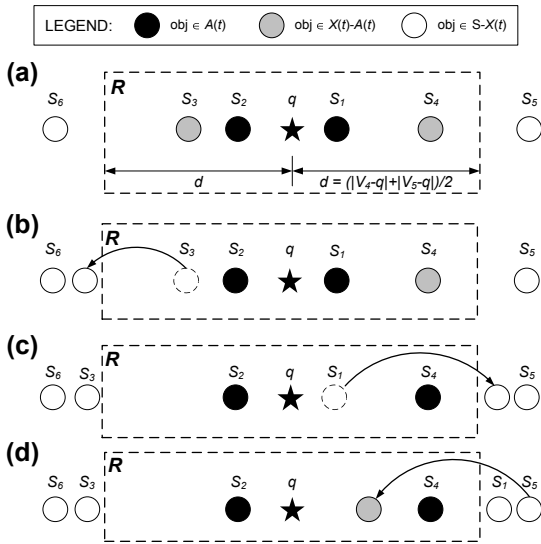


Figure 6: RTP for k -NN with $k = 2$ and $r = 2$.

from $X(t) - A(t)$ (Steps 2 and 3) where $\text{rank}(S_j, t) \leq \epsilon_k^r$. Figure 6(c) gives an example of this case. As S_1 moves out of R , it is replaced by S_4 in $A(t)$.

It is possible that the set $X(t) - A(t)$ is empty due to removals caused by repeated application of Step 1 above. We can choose to re-execute Initialization phase, but this is expensive as all streams need to be probed. 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 j th-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 not less than 2, then $A(t)$ and $X(t)$ will be fixed and the new bound is deployed (Step 4(I)(iv)) (the notation $\min_{r+1, S_i \in U(t)} |V_i - q|$ in (iv)(b) means any object in $U(t)$ ranking $(r + 1)$ st or higher in terms of distance from q). The search region expands until we reach V_n , and if still nothing is found, Initialization phase will be re-executed.

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. Initialization needs $O(n)$ messages. The running cost for the maintenance phase is $O(nr)$. In practice this number can be fewer. As illustrated in Figure 6, objects can leave R (a) or enter R (d), without incurring any maintenance costs. Details of this analysis are described in [4].

5 Fraction-based Tolerance

As discussed earlier, fraction-based tolerance can be applied to both non-rank-based and rank-based queries. In Section 5.1, we study a protocol for non-rank-based queries. We further extend that 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 one type of the 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.

First, consider a simple filter protocol where no tolerance is allowed: each stream filter is assigned the constraint $[l, u]$ 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 algorithm *zero-tolerance protocol for non-rank-based query (ZT-NRP)*.

Although ZT-NRP can reduce the amount of communication, it does not exploit any tolerance at all. Thus updates may be generated unnecessarily. The protocol described next utilizes fraction-based tolerance to achieve a better performance.

5.1.1 Exploiting Fraction-based Tolerance

Initialization Phase In the *fraction-based tolerance protocol for non-rank-based queries*, or **FT-NRP** in short (Figure 7), no more than a fraction ϵ^+ of the answer (i.e., $A(t)$) can be wrong at any time t . The server first captures the states of the streams at time t_0 (Steps 1-3). Out of the $|A(t_0)|$ answers that satisfy the range query, we assign the constraint $[-\infty, \infty]$ to $E^{\text{max}+}(t_0)$ filters, and $[l, u]$ to the remaining $|A(t_0)| - E^{\text{max}+}(t_0)$ filters (Step 4), where $E^{\text{max}+}(t_0)$ is given by $|A(t_0)| \cdot \epsilon^+$ (Equation 3). Filters assigned with $[-\infty, \infty]$ (called *false positive filters*), do not report their values. If no stream in $A(t)$ replies, the false positive requirement is met i.e., $F^+(t) \leq \epsilon^+$. Since $E^{\text{max}+}(t_0)$ streams are “shut down”, the amount of communication is reduced. This approach also potentially saves battery power in a sensor network, since the sensors can be “shut down” and consume less energy than active sensors. We use n^+ , initially equal to $E^{\text{max}+}(t_0)$, to denote the current number of false positive filters.

False negative tolerance can be exploited in a similar way. Observe that $|Y(t_0)| = |S - A(t_0)|$ streams do not satisfy Q . We assign $[\infty, \infty]$ (called *false negative filters*) to $E^{\text{max}-}(t_0)$ of them, and $[l, u]$ to the remaining ones. In essence, $E^{\text{max}-}(t_0)$ streams are “turned off”. When no data are received from $Y(t_0)$, we guarantee at any time t , $F^-(t) \leq \epsilon^-$. From Equations 2,3 and 4, $E^{\text{max}-}(t_0)$ equals $|A(t_0)| \frac{\epsilon^-(1-\epsilon^+)}{1-\epsilon^-}$. We use n^- ,

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. $A(t_0) \leftarrow \{S_i | V_i \in [l, u] \text{ at time } t_0\}$
 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 into $A(t)$
 - (II) $count \leftarrow count + 1$
2. **else**
 - (I) remove S_i from $A(t)$
 - (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 into $A(t)$
 - (III) install $[l, u]$ for the filter of S_z
 - (IV) $n^- \leftarrow n^- - 1$

Figure 7: The **FT-NRP** algorithm (at the server side).

initially set to $E^{max-}(t_0)$, to denote the current number of false negative filters.

After Initialization Phase, correctness requirement 1 is satisfied. That is, if no update is received at time t , $F^+(t) \leq \epsilon^+$ and $F^-(t) \leq \epsilon^-$.

Maintenance Phase Updates can affect the correctness of **FT-NRP**. Assume the server receives an updated value V_i from S_i at time t_u . Immediately prior to receiving V_i , according to correctness 1, the following must hold (from Equations 3, 4):

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

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

Let t be the current time instant with $t \geq t_u$. There are two different cases of updates to consider:

Case 1: $V_i \in [l, u]$. This means S_i , previously not in the result, is now an answer. We handle this by inserting S_i into $A(t_u)$ (Step 1(I)). As $E^{max+}(t)$ is 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 prove similarly that Equation 12 holds. Thus correctness 2 is guaranteed.

Observe that *both* $F^+(t)$ and $F^-(t)$ drop. This is because the answer quality is improved when more streams satisfying the query. We exploit this by using a variable, **count**, to record the number of new items inserted into the answer under this scenario (Step 1(II)). Let t_c denote the time when **count** is zero. Then, $|A(t)| = |A(t_c)| + \mathbf{count}$ for $\mathbf{count} \geq 0$. Intuitively, t_c is the time when F^+ and F^- attain their maximum values without violating correctness. At any time t , if $\mathbf{count} \geq 0$, $F^+(t) \leq F^+(t_c)$ 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 into 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 $n^+ > 0$, 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 $[l, u]$.

Case 1: $V_y \in [l, u]$. S_y is now a true positive. Since S_y was assigned a false positive filter, V_y has already been in $A(t_u)$, so $|A(t_u)|$ is unchanged (i.e., $|A(t_c)| - 1$).

We then install $[l, u]$ for S_y to ensure $V_y \in [l, u]$ when no update is received from it (Step (II)(i)). Since S_y is no longer a false positive, $E^{max+}(t_u)$ is decremented.

Thus $F(t_u)$ is now less than $\frac{E^{max+}(t_c)-1}{|A(t_c)|-1}$, which is less than $F^+(t_c)$, meeting the false positive constraint. The false negative constraint is also satisfied: by Equation 4, $F^-(t_u) \leq \frac{E^{max-}(t_u)}{|A(t_u)| - E^{max+}(t_u)}$, or

$\frac{E^{max-}(t_c)}{(|A(t_c)|-1) - (E^{max+}(t_c)-1)}$, which is less than ϵ^- .

Case 2: $V_y \notin [l, u]$. 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 $\epsilon^+ \leq 0.5$, $\frac{E^{max+}(t_c)-1}{|A(t_c)|-2}$ cannot be

larger than $\frac{E^{max+}(t_c)}{|A(t_c)|}$, and is thus 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 $V_z \in [l, u]$, 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 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 $\frac{E^{max-}(t_c)-1}{(|A(t_c)|-2)-E^{max+}(t_c)}$, which is smaller than ϵ^- because $\epsilon^- \leq 0.5$. Hence correctness 2 is also met.

The details of the correctness proofs for the special cases: (1) $n^+ = 0 \wedge n^- > 0$ and (2) $n^+ > 0 \wedge n^- = 0$ can be found in our technical report [4]. We remark that when both n^+ and n^- become zero, all false positive and negative filters are essentially replaced by the $[l, u]$ constraint. Hence the fraction-based tolerance constraints are trivially met, and the protocol reduces to **ZT-NRP**. To exploit tolerance, the Initialization Phase of **FT-NRP** may be run again.

Communication Costs. Initialization requires $O(n)$ messages, while maintenance generates at most five messages. However, since no messages are required unless count is zero, the actual cost can be lower.

5.2 Rank-based queries

We now present the fraction-based tolerance protocol for k -NN query. Our solution is to transform a k -NN query to a range query, and then adopt the fraction-based protocol we described in Section 5.1.

5.2.1 Transforming k-NN 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). We call this protocol **ZT-RP**. Its Initialization Phase computes R and then distributes R to all the stream filters. 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 still encloses the k nearest objects. In addition, the new R has to be announced to every stream.

The main drawback of this simple protocol is that it is very sensitive to the situation when an object's value crosses R . When this happens, R has to be re-

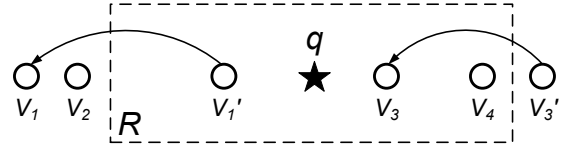


Figure 8: False positives and false negatives for k -NN.

computed and announced to every stream! Now let us investigate how this problem can be alleviated.

5.2.2 Using FT-NRP for k-NN Query

We just discussed how to view 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. To develop a fraction-based tolerance protocol for a k -NN query, apparently we can transform a k -NN query to a range query, and then directly apply **FT-NRP**. However, this is incorrect. In particular, we may not use the values of ϵ^+ and ϵ^- specified by the k -NN query to parametrize **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 k th-ranked object and thus contains k objects. Similar to the initialization phase of **FT-NRP**, for objects with values in R we assign false positive filters to $k\rho^+$ streams; and for streams with values outside R , we apply false negative filters to $k\rho^-$ streams. All other streams are assigned R as their filter bounds. The rest of this section examines how ρ^+ and ρ^- should be set.

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 8. 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 at most $|A(t)|\rho^+$ streams are assigned with false positive filters, at most $|A(t)|\rho^+$ false positives are produced in this way.

Another kind of false positive is caused by false negative filters. Suppose S_4 , being ranked the k -th and lies within R , is part of the answer. Also assume S_3 is associated with a false negative filter, whose last reported value, V_3' , does not lie within R . As illustrated in Figure 8, when the new value of S_3 , i.e., V_3 , is within R , the rank of S_3 becomes k or higher. The rank of S_4 is demoted to $k + 1$ and thus S_4 becomes a false positive. Since false negative filters can be assigned to at most $k\rho^-$ streams (Equation 5), at most $k\rho^-$ false

positives are created in this way.

The sum of the false positives generated by these two scenarios is $|A(t)|\rho^+ + k\rho^-$, where $|A(t)|$ is less than $\frac{k}{1-\epsilon^+}$ (Equation 7). Also, the user cannot tolerate more than $|A(t)|\epsilon^+$ false positives, with a minimum value of $k(1-\epsilon^-)\epsilon^+$ (Equation 9). Therefore,

$$\rho^- \leq \frac{\rho^+}{\epsilon^+ - 1} + (1 - \epsilon^-)\epsilon^+ \quad (13)$$

Meeting false negative requirement. Again there are two types of false negatives for a k NN query. As shown in Figure 8, 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. The server does not know this, and so S_3 is a false negative. The number of false negatives 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 (without notifying the server). The maximum number of this kind of false negatives is thus $|A(t)|\rho^+$, the maximum number of false positive filters. Since the maximum number of false negatives for k -NN query is given by $k\epsilon^-$, the sum of the two kinds of false negatives, $k\rho^-$ and $|A(t)|\rho^+$, must be less than $\leq k\epsilon^-$. Equation 7 simplifies this to:

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

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:

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

Essentially, when the user-defined constraints for rank-based query (i.e., ϵ^+ and ϵ^-) are implemented using **FT-NRP**, 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:

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

5.2.3 Fraction-based Tolerant k -NN Query

Once the values of ρ^+ and ρ^- are correctly 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) A requirement for the answer $A(t)$ of a rank-based query is that $k(1 - \epsilon^-) \leq |A(t)| \leq \frac{k}{1-\epsilon^+}$ (Equations 7, 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 i.e., R is “too loose”. Similarly, when $|A(t)|$ drops below $k(1 - \epsilon^-)$, there are too few objects in R , i.e., R is “too tight”. In either case, R is no longer an appropriate filter bound, and a new bound has to be found to enclose the new k -nearest neighbors.

The advantage of **FT-RP** over **ZT-RP** is easily seen – 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 $\frac{k}{1-\epsilon^+}$.

6 Experimental Results

We now present an experimental evaluation over our protocols. We use CSIM 19 [19] to simulate the environment in Figure 3. 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 query¹. We now present two sets of results, based on real and synthetic data.

6.1 TCP Data

In the first set of experiments, we test the efficiency of our protocols based on TCP traces [15]. The experiment models a remote network monitoring application, where a central console is used to monitor a network composed of 800 subnets. We assume an agent software that implements our filters is installed at each subnet router. 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 subnet. Each of the 800 subnets represents a stream source. The “number of bytes sent” field in each packet trace is used as a data value. Here a range query can be used to classify subnets with different ranges of traffic volume. A top- k query is used to report continuously the subnets with the k -highest volume of data transferred [2].

¹When no filter is used, a “maintenance message” is essentially an update message from a stream source.

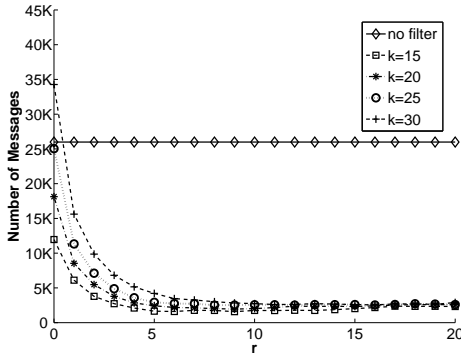


Figure 9: **RTP**: Effect of r

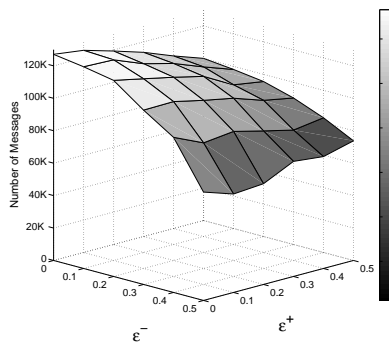


Figure 10: **FT-NRP**: Effect of ϵ^+/ϵ^-

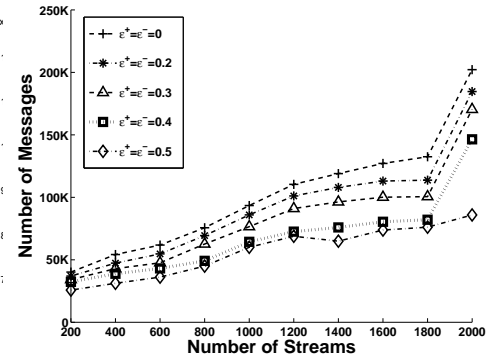


Figure 11: **FT-NRP**: Scalability

Figure 9 shows the result of varying the rank-based tolerance r for some values of k . For each value of k shown, the performance improves as r increases. Thus **RTP** is able to exploit tolerance effectively. Notice that at $k = 30$ and $r = 0$, the performance is worse than when no filter is used. 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. A range query with $[l, u] = [400, 600]$ is used. In Figure 10, the number of messages decreases as ϵ^+ and ϵ^- increase. Thus **FT-NRP** performs consistently better than **ZT-NRP**. We do not show the result when no filter is used because it has a very high cost. The scalability of **FT-NRP** is shown in Figure 11. The protocol in general scales well, and for a larger number of streams, the performance gains more by using higher tolerance values.

6.2 Synthetic Data

Next, we test the protocols with a synthetic data model. It gives us better control over data behavior. We assume 5000 data streams, 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 $[l, u] = [400, 600]$ over different values of ϵ^+ and ϵ^- . Figure 12 shows that **FT-NRP** exploits tolerance effectively. Figure 13 then 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.

We also explore how **FT-NRP** is affected by the assignment of false positive and false negative filters dur-

ing the initialization phase. We compare two heuristics: (1) *random* – streams are randomly selected to be assigned with $[-\infty, \infty]$ and $[\infty, \infty]$ constraints, and (2) *boundary-nearest* – only streams with values closest to the user-defined range $[l, u]$ are assigned with $[-\infty, \infty]$ and $[\infty, \infty]$ constraints. Figure 14 shows that *boundary-nearest* 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 reported 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* than by *random*.

The performance of **ZT-RP** and **FT-RP** over different values of k is shown in Figure 15. For k equals 60 or 100, the number of messages drops significantly with a slight increase in tolerance. This is because the bound R for enclosing the 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 note that the protocol does not perform well at $k = 20$ and $\epsilon^+ = \epsilon^- = 0.1$. At small values of k and tolerance, the number of false positive and negative filters allocated is limited. The little benefit of tolerance cannot overcome the high maintenance cost. We remark that **FT-RP** is not suitable in this situation.

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.

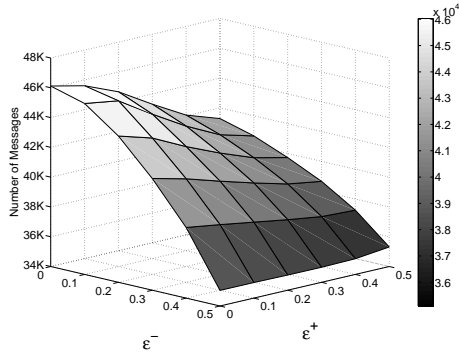


Figure 12: **FT-NRP**: Effect of ϵ^+/ϵ^-

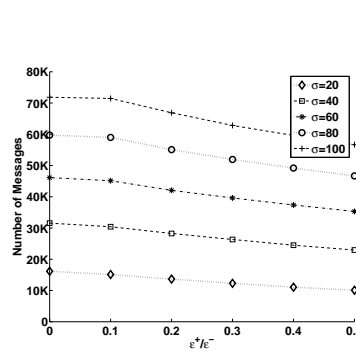


Figure 13: **FT-NRP**: Data fluctuation

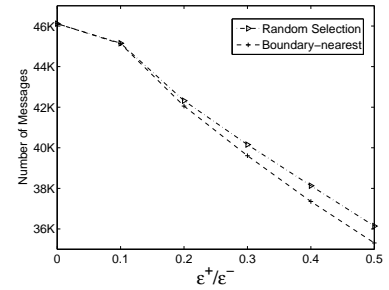


Figure 14: **FT-NRP**: Selection heuristics

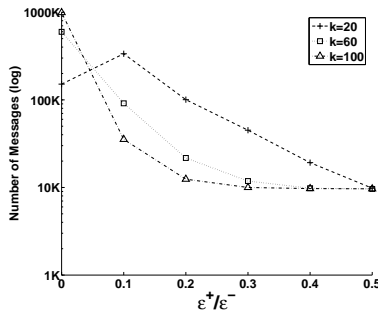


Figure 15: **ZT-RP/FT-RP**: Effect of ϵ^+/ϵ^-

We plan to extend the protocols to support multiple queries, and examine how existing data stream algorithms can be modified to support non-value tolerance.

Acknowledgments

This research was partially supported by Hong Kong Grants Council grants HKU 7040/02E. Reynold Cheng and Sunil Prabhakar were supported by NSF Grants IIS 9985019 and CCR-0010044. We thank the anonymous reviewers for their insightful comments.

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] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. ACM SIGMOD*, 2003.
- [3] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. ACM SIGMOD*, 2003.
- [4] R. Cheng, B. Kao, S. Prabhakar, A. Kwan, and Y. Tu. Adaptive stream filters for entity-based queries with non-value tolerance. Technical Report CSD TR #05-003, Dept. of CS, Purdue University, 2005.
- [5] 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.

- [6] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *Proc. EDBT*, 2004.
- [7] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. VLDB*, 2004.
- [8] D. Abadi and et al. Aurora: A data stream management system. In *Proc. ACM SIGMOD*, 2003.
- [9] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proc. ACM PODS*, 2004.
- [10] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proc. VLDB*, 2003.
- [11] G. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *Proc. VLDB*, 2003.
- [12] A. Jain, E. Chang, and Y. Wang. Adaptive stream resource management using kalman filters. In *Proc. ACM SIGMOD*, 2004.
- [13] S. Khanna and W. C. Tan. On computing functions with uncertainty. In *ACM PODS*, 2001.
- [14] N. Koudas, B. Ooi, K. Tan, and R. Zhang. Approximate NN queries on streams with guaranteed error/performance bounds. In *Proc. VLDB*, 2004.
- [15] Lawrence Berkeley National Laboratory. The Internet Traffic Archive, USA. URL <http://ita.ee.lbl.gov>.
- [16] J. Ni and C. V. Ravishankar. Probabilistic spatial database operations. In *Proc. SSTD*, 2003.
- [17] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. ACM SIGMOD*, 2003.
- [18] V. Poosala and V. Ganti. Fast approximate query answering using precomputed statistics. In *Proc. ICDE*, 1999.
- [19] Mesquite Software. CSIM 19. URL <http://www.mesquite.com>.
- [20] S. Vrbsky and J. Liu. Producing approximate answers to set- and single-valued queries. *Journal of Systems and Software*, 27(3), 1994.
- [21] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3), 1999.