The Price of Validity in Dynamic Networks

Mayank Bawa, Aristides Gionis, Hector Garcia-Molina, Rajeev Motwani
Computer Science Department
Stanford University
Stanford, CA 94305
{bawa,gionis,hector,rajeev}@db.stanford.edu

ABSTRACT

Massive-scale self-administered networks like Peer-to-Peer and Sensor Networks have data distributed across thousands of participant hosts. These networks are highly dynamic with short-lived hosts being the norm rather than an exception. In recent years, researchers have investigated *best-effort* algorithms to efficiently process aggregate queries (e.g., sum, count, average, minimum and maximum) [6, 13, 21, 34, 35, 37] on these networks. Unfortunately, query semantics for best-effort algorithms are ill-defined, making it hard to reason about guarantees associated with the result returned. In this paper, we specify a correctness condition, *single-site validity*, with respect to which the above algorithms are best-effort. We present a class of algorithms that guarantee validity in dynamic networks. Experiments on real-life and synthetic network topologies validate performance of our algorithms, revealing the hitherto unknown *price of validity*.

1. INTRODUCTION

Consider an aggregate query that has to be computed over data hosted in a network of n hosts. A simple approach is to ship data from the n hosts and store it in a central database where query processing can take place (the *warehousing* approach [5]). The database can take steps (e.g., concurrency control) to ensure valid semantics for the query: the aggregate reflects data for some snapshot of the network. Although simple, such data shipping from all hosts in the network incurs a high communication cost both in the network and at the central database host.

The alternative approach is to leverage the computational capabilities of hosts in the network by shipping the query and processing it using a distributed query plan (the *in-network processing* approach [5]). An efficient query plan enables only relevant data to be shipped thereby reducing communication cost.

The emergence of large-scale self-administered networks over the last decade has forced us to think about scenarios where n is of the order of thousands or millions of short-lived hosts. Peer-to-Peer (P2P) and Sensor Networks are prime examples of such massively distributed data-centric networks. P2P Networks like Gnutella, KaZaa and Freenet have been used for file-sharing by millions of hosts that have short lifetimes.[1] A sensor network [11, 17, 29] consists of thousands of sensors that monitor their environment in real-time, communicate over a wireless network and have lifetimes dictated by their internal battery unit.

SIGMOD 2004 June 13-18, 2004, Paris, France.

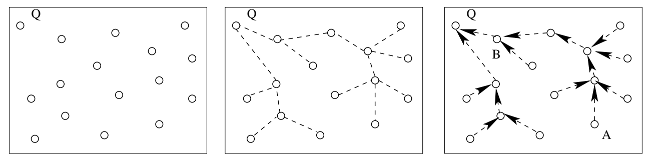

Figure 1: *Ill-Defined Semantics* (a) Sensor Network with 16 sensors, (b) Broadcast, and (c) Convergecast. Failure of sensors A and B *after* Broadcast leads to counts of 15 and 6 respectively. Which of these is correct and why?

In-network processing has been the approach of choice for large-scale networks in several projects [6, 13, 21, 34, 35, 37]. A distributed query plan however requires us to deal with dynamism in the network, and the semantics of the final answer returned. In traditional distributed database systems, a host failure leads to an abort of the query. However, in large-scale networks, we want query processing to continue even if a "few" hosts fail. Unfortunately, the algorithms proposed have been *best-effort* and the semantics of the aggregate thus computed are rather ill-defined in the presence of host failures.

EXAMPLE 1.1. *Consider the Sensor Network shown in Figure 1-a on which a user wishes to* count *the number of active sensors. Proposed algorithms [21, 35, 37] process such aggregate queries in two phases. In the first (Broadcast) phase, the query floods the network and a spanning tree is built on the sensors (Figure 1-b). In the second (Convergecast) phase, sub-tree counts are propagated up from the leaves to the root (Figure 1-c). Each interior host adds together its sub-tree counts with a* 1 *for itself, and propagates the result to its parent.*

A failure-free execution of Convergecast returns count=16 *as the answer. However, even if there is a* single *failure in*

[1] The median session duration for a Gnutella host was measured as 60 minutes in 2001 [32].

the network, the answer returned could vary from count=16 *to* count=6 *depending on which sensor failed and when. The user is thus unable to associate a meaning to the* count *value returned: What is the correct answer and how does it relate to the value returned?* ∎

In large-scale self-administered networks, host failures are the norm rather than being an exception, and it is necessary to give a meaning to possible interleaving of host failures with query processing. We propose defining query semantics for dynamic networks, and by extension revisiting the query processing algorithm itself. In this paper we present motivation, methodology and performance results on designing algorithms that ensure valid semantics.

EXAMPLE 1.2. *What is the meaning of the answer to a query? The desire to associate semantics with a query result is certainly not new and this question has been answered in a variety of domains by researchers.*

For instance, Hellerstein et. al. [14] suggest performing aggregation online to allow users to observe the progress of their queries. A running aggregate is displayed to the user as an estimate of the final result based on the tuples retrieved so far. By itself, the running aggregate is meaningless to the user. The semantics of the running aggregate are fixed by associating properties of "Confidence" and "Interval Bounds". The twin properties together give a probabilistic estimate of the running aggregate's proximity to the final result. ∎

Our Contribution (Validity Semantics): In this paper, we develop a simple and intuitively appealing correctness condition for queries on dynamic networks. We say that a query result is *single-site valid* if it is "equivalent", in a sense formally defined in Section 4, to a legal failure-free computation *as observed from the querying host.* We extend this definition to the continuous and approximate query domains. We propose an in-network processing scheme, WILDFIRE, that computes single-site valid results for aggregate (min, max, count, sum and avg) queries (Section 5). We compare its performance against efficient best-effort algorithms on real-life and synthetic networks (Section 6).

Our results show that while existing best-effort algorithms perform poorly, WILDFIRE returns valid answers *even under high dynamism.* WILDFIRE incurs similar costs as best-effort algorithms for min and max queries, but has to pay 5× communication cost for count and sum queries. Single-site validity and WILDFIRE represent a first step in a larger effort to understand query processing in dynamic networks. In this paper we focus on developing and achieving validity semantics; we leave fine tuning to a future exercise.

2. RELATED WORK

Aggregate queries have been a topic of much research in recent years as aggregates are a useful means of summarizing state for large-scale networks. For example, aggregate queries can be used to deduce usage trends in P2P networks [34, 36] (e.g., average load on hosts in the network, average lifetime of hosts) and are of primary interest in sensor networks [11] (e.g., average temperature readings recorded in an area). Moreover, dynamic networks have been associated with environments where communication is a premium (e.g., low bandwidth hosts in P2P Networks, high cost of transmitting wireless messages in Sensor Networks) and aggregation schemes have been proposed as a means of reducing communication costs [16, 21, 35].

The issues explored in this paper are akin to the questions raised by Imielinski and Badrinath [15] on querying transient data in mobile distributed networks: "Information changes so fast in our application that it may change *during* query evaluation. This creates an interesting research problem on the very basic, semantic level. What is the meaning of the answer to the query? How to compute it and how to augment it?" We briefly review such work as has been done in the context of dynamic P2P and Sensor Networks.

Append-only Continuous Query Semantics: Bonnet et. al. [5] propose a data model and continuous query semantics for sensor networks that mixes stored relational data at a central host with data extracted from Sensor Networks represented as time series. Each continuous query defines a persistent view which is maintained during a given time interval. The model allows append-only inserts to the sensor data sequence and does not specify how to incorporate changes in data sequence due to host failures. It also does not consider effects of failures of hosts that are participating in the query plan for view maintenance.

Eventual Consistency: Gossip-based or epidemic algorithms [1, 8] are known to be tolerant of random failures of hosts in a network. Epidemic algorithms for aggregate computation [13, 18, 34] operate over multiple rounds. In each round, each host exchanges information with one or a few hosts chosen at random. The guarantees of the final result obtained from such "gossiping" are usually probabilistic in nature and afford weak semantics of *eventual consistency.* In the presence of updates and host failures, epidemic algorithms can only guarantee that a correct answer reflecting the changes will be returned *eventually* (e.g., when updates or host failures cease and the network becomes stable).

Best-Effort Semantics: Best-effort algorithms [6, 13, 21, 35, 37] opt for a "lossy" query processing and declare answers that can be arbitrarily bad in the presence of host failures. The algorithms use a combination of Broadcast and Convergecast as a mechanism for query processing. One or more spanning trees are built on the relevant hosts using Broadcast; data is aggregated along the trees using Convergecast. As we discuss in Section 4.4, the final results can be arbitrarily bad if hosts fail during Convergecast.

Validity Metrics: *Completeness* [13] or *Relative Error* [6, 37] have been used to measure the validity of query results. Completeness is the percentage of hosts in the network whose data contributed to the final query result. Relative Error is $|\frac{\widehat{x}}{x} - 1|$ where $\widehat{x}$ is the reported result and x is the "true" result. These are essentially validity metrics that can only be computed by an Oracle (with a perfect view of the dynamic network) *post* processing, and not properties that can be associated with the query processing algorithm.

The aggregation operators proposed in Section 5 use the probabilistic counting schemes of Flajolet and Martin [12]. Similar techniques were used by Palmer et. al. [25] for approximating the neighbor function of a disk-based graph, and in a concurrent work, by Considine et. al. [6] for use in aggregation with multiple spanning trees.

3. PROBLEM SETTING

In this section, we define the problem of computing an aggregate query over data distributed across hosts in a network. The input to the problem is a set of *hosts* $H = \{h_1, h_2, \ldots, h_n\}$ and a *querying host* h_q that issues an *aggregate query.* Each host is said to *possess* attribute-value pairs which are appropriately named and typed. The desired output is a value $v := q(H)$ which is the aggregate computed over all attribute values shared by hosts in the network.[2]

Symbol	Definition
$G := (H, E)$	Network G with hosts H and edges E
h_q	Host that initiates query processing
δ	Maximum time delay between $(h, h') \in E$
H_t	Hosts in G at time t
D_t	Diameter of G at time t
$\widehat{D}$	Overestimate of D_t over interval $[0, T]$

Table 1: Notation

Network Model

The hosts communicate over a network that can be represented as an undirected graph $G := (H, E)$, where H is the set of hosts and E is a set of pairs (h, h') that describe symmetric neighbor relations between hosts $h, h' \in H$. Messages can only be sent in G from a host to its neighbor.

We work with a *relaxed asynchronous* model of distributed systems, i.e., there are known upper bounds on process execution speeds, message transmission delays, and clock drift rates. All of the above bounds can be integrated in a known universal maximum delay δ between $(h, h') \in E$. Thus a message generated at time t at a source h will be received by a live destination $h' \in N(h)$ by time $t + \delta$.

In this setting, hosts can monitor a neighboring host for failures using heartbeats sent periodically at intervals of time T_{hb}. If a host h does not receive a heartbeat from its neighbor h' within $T_{hb} + \delta$ time of the last heartbeat, then h can deduce that there must have been a failure at h'.

EXAMPLE 3.1. *A P2P Network is an overlay on the Internet. Each host in the P2P network maintains a list of its neighbor hosts' IP-addresses. Neighbors have a symmetric TCP connection established between them over which messages can be exchanged. Hosts in a P2P Network are on the Internet and usually synchronize their clocks using NTP [24] attaining accuracies within a few milliseconds.*

Each sensor in a Sensor Network has a unique medium access control (MAC) address [29]. Each sensor is equipped with a wireless radio which is used to transmit and receive messages. The messages are transmitted short-range to reduce power consumption. Each message specifies the recipient sensor's MAC address; all other sensors within range drop messages not intended for them. Asymmetric links, where sensor h can receive messages from its neighbor h' but not vice-versa are detected and ignored by common routing protocols [27]. Hosts in a Sensor Network have their clocks locally synchronized with their neighbors [10]. ∎

Dynamism Model

In a dynamic network, hosts can join or leave the network at will. A host that leaves the network is called a *failed* host and does not participate in the network protocol anymore. The graph G is updated on join or leave of a host, and the neighborhood sets are updated dynamically. We use H_t to denote the set of hosts, and D_t the diameter of G at time t. We will assume that it is possible to "guess" a constant $\widehat{D} \geq max_{t=0}^{T}\{D_t\}$ over a small interval of time $[0, T]$.

We assume that a message sent from a host to a live neighbor is reliably delivered. However, we will be concerned about *overlay network partitions*, where the network G becomes disconnected due to host failures, making it impossible for some hosts to communicate with each other when using multi-hop message routes in G.

EXAMPLE 3.2. *Hosts in a P2P Network join the network when the end-user activates the P2P application. Hosts can leave the network when the end-user deactivates the application, terminates network connectivity, or switches the computer off. Hosts communicate using TCP/IP which ensures reliable in-order delivery of messages.*

Hosts in a Sensor Network join the network when the inbuilt timer sets on. Hosts can leave the network when they "die" due to battery-failure, software exceptions, hardware faults, etc [11]. Although link failures are much more common in Sensor Networks resulting in message losses [7], poor quality links can be detected and ignored while routing multi-hop messages [37].

Information networks have been shown to exhibit the small world phenomenon *[20] where D grows extremely slowly with $|H|$. The social network had $D = 6$ in 1967 [23], World Wide Web had $D = 19$ in 1999 [2] and Gnutella had $D = 12$ in 2001 [30]. A crude overestimate $\widehat{D}$ is thus feasible.* ∎

4. AGGREGATE QUERY SEMANTICS

An in-network query processing algorithm has to contend with dynamism as hosts join and leave the network unpredictably. In this section, we define a model for query processing that associates semantics with query results even in the event of changes in the host set H during processing. We analyze its computability characteristics and extend it to the continuous query and approximate query domains.

4.1 One-time Query Semantics

In the in-network query processing schemes, each host manages its own attribute values, and participates in query processing to send only relevant data over the network. A user issues an aggregate query at one of the hosts h_q. The query is processed in two phases. In the first (Broadcast) phase, h_q floods the query in G as each host forwards the query to its neighbors.[3] The hosts on receiving the query initiate the second (Convergecast) phase, when the query is processed and a result is returned to h_q.

> *Suppose that a user issued an aggregate query from h_q at time 0 for which an in-network algorithm declared the result to be a value v at time T. What semantics would be desirable on such a value v?*

We will start our discussion by requiring the *strictest* semantics. We will analyze such a requirement to understand the reasons for its infeasibility. We will then chip away at the requirement until we arrive at one that is indeed computable. We will see that *dynamism* and *distribution* combine to cause fundamental uncertainty in both the *time* and *operands* over which a query is performed.

---

[2] For ease of exposition, we will use h to refer to both the unique *identifier* of the host and the attribute *value* being aggregated that is possessed by the host. The particular use will be clear from context.

[3] The query may first be routed to a relevant sub-network and then flood to only hosts in the sub-network.

> **Snapshot Validity:** An algorithm that computes an aggregate query must guarantee that $v = q(H_t)$ for some time t in $[0, T]$.

The strictest semantics that we can require is to have v correspond to an evaluation of the query for some snapshot of the network. Formally, we desire that the result returned be the *exact* value of the aggregate query at some time instant t in the interval $[0, T]$. The following theorem shows that it is *impossible* to devise an algorithm that can guarantee such semantics.

THEOREM 4.1. *There is no algorithm in the relaxed asynchronous model with reliable ordered communication that satisfies Snapshot Validity.* ∎

PROOF. Assume for contradiction that there is an algorithm A^* that achieves Snapshot Validity for some t in $[0, T]$. Consider a run of the algorithm A^* on G constructed as follows. Arrange $k+1$ hosts labeled $h_0, h_1, \ldots, h_k$ in a chain. Host h_0 initiates A^* at time 0 by flooding the query in G. For any such t, a chain of new hosts labeled $h'_2, h'_3, \ldots, h'_k$ can join G at h_1 at instant $t-1$. Then, $H_t = \{h_0, h_1, \ldots, h_k, h'_2, h'_3, \ldots, h'_k\}$. However, h'_k becomes aware of the query only by $t+k-1$ by when the value at h'_k may have changed. Thus, h'_k is unable to contribute its value of time t to A^*, resulting in a contradiction. □

The negative result for Snapshot Validity forces us to think about properties that can be associated with state of the network $\{H_0, H_1, \ldots, H_T\}$ over the interval in which query was being processed. Let $H_I = \cap_{t=0}^{T} H_t$ (Intersect) denote the set of hosts that were alive at *all* instants and $H_U = \cup_{t=0}^{T} H_t$ (Union) denote the set of hosts that were alive at *some* instant during the query processing interval.

> **Interval Validity:** An algorithm that computes an aggregate query must guarantee that $v = q(H)$ for some set H such that $H_I \subseteq H \subseteq H_U$.

Interval Validity requires that v reflect data values that were in the network *during* query processing. The set H_I consists of all hosts that were in G during the entire query processing interval, while H_U additionally includes hosts that may have joined or left G while the query was being processed. We now require that the result be $v = q(H)$ for some $H_I \subseteq H \subseteq H_U$. The following theorem shows that, once again, *no* algorithm can guarantee such semantics.

THEOREM 4.2. *There is no algorithm in the relaxed asynchronous model with reliable ordered communication that satisfies Interval Validity.* ∎

PROOF. Consider a network G with a host h which is 1-connected to h_q implying that there exists a host h' whose failure disconnects h from h_q. Notice that h_q cannot communicate with h along G if h' fails. If h' fails at instant 1 in the Broadcast phase before q has reached h', the alive host h will never receive the query, and hence will be unable to contribute its attribute value to the final query result v.

Assume for contradiction that there is an algorithm A^* that achieves Interval Validity. Consider a network G' that has all hosts and edges in G except h and its edges. Notice that $H'_I = H_I - \{h\}$ and $H'_U = H_U - \{h\}$ where H'_I and H'_U are defined for G'. Let A^* be executed on both networks G and G' and in both cases h' fails as discussed earlier. It is easy to see that A^* will return the same answer $v = q(H)$ for both G and G'. However, $H = H'_I \subset H_I$ implying $H \subset H_I \subset H_U$, a contradiction. □

```
PROTOCOL ALLREPORT (q, D̂)
    ▷ On issue of query q at host h_q
    Send Broadcast message [h_q, q] to all neighbors
    M := {h_q}
    Set Timer to expire after 2D̂δ
    while (Timer not expired) {
        ▷ On receipt of attribute value a from h
        M := M ∪ {a}
    }
    Output v = q(M)
    Terminate

    ▷ On receipt of query q at host h ≠ h_q
    Send Broadcast message [h_q, q] to all neighbors
    Send attribute value to h_q
    Terminate
```

Figure 2: ALLREPORT **achieves Single-Site Validity**

The contradiction for Interval Validity motivates us to think about network connectivity. The Broadcast phase starts from the host h_q and disseminates the query across the network. The Convergecast phase collects attribute values in the network and returns an answer to h_q. We want v to reflect the data values possessed by hosts that were reachable from h_q during both phases. As before, we can define a lower bounding host set H_C and an upper bounding set H_U and require that $v = q(H)$ where $H_C \subseteq H \subseteq H_U$. Let $H_U = \cup_{t=0}^{T} H_t$, while H_C consist of hosts such that for each host $h \in H_C$, there is at least one path in G from h_q to h that is stable during $[0, T]$.

> **Single-Site Validity:** An algorithm that computes an aggregate query must guarantee that $v = q(H)$ for some set H such that $H_C \subseteq H \subseteq H_U$.

The following theorem assures us that an algorithm that guarantees Single-Site Validity can be devised. In fact, the proof of the theorem is constructive and presents a simple, though inefficient, algorithm to achieve the above semantics.

THEOREM 4.3. *There is an algorithm in the relaxed asynchronous model with reliable ordered communication that satisfies Single-Site Validity.* ∎

PROOF. Consider the ALLREPORT algorithm presented in Figure 2 which is initiated by h_q flooding the query in G. Each host that receives the query sends its attribute value to h_q. Host h_q collects value reports into a set M until $T = 2\widehat{D}\delta$ time, when it declares the result to be $v = q(M)$.

To prove the upper bound on M, we note that a host sends its value to h_q only upon receiving the Broadcast message. Such a host must clearly have been alive in $[0, T]$ implying the upper bound. To prove the lower bound on M, we need to show that *all* hosts in H_C must have sent their values to h_q by time T. By definition, each host $h \in H_C$ has a path P_h in G from h_q that exists during the query processing interval. Note that $length(P_h) \leq max_{i=0}^{n}\{D_i\} \leq \widehat{D}$. Thus, h will receive the Broadcast message and its value will be received at h_q by time T. $\square$

The AllReport algorithm explores the network starting from h_q. The querying host progressively builds its own "view" of the network G over time using which the query result is computed. As observed from h_q, all joins and leaves of hosts can be viewed as occurring *before* query processing started or *after* query processing ended, with H being alive in G during the entire query processing interval. Since the result is based on the view from a single host, we name such a guarantee *Single-Site Validity.*

4.2 Continuous Query Semantics

Until now, we have assumed that the aggregate query is a one-time query. However, aggregate queries are also a key component of online monitoring applications. Queries in such applications are long-running and periodic, allowing users to receive results at h_q *continuously* [5].

A user registers a continuous query by contacting one of the hosts h_q at time 0, requesting its processing until time T. In this setting, what semantics would be desirable on the result v_t received at h_q at time t in $[0, T]$? It is easy to see that the impossibility results of Snapshot and Interval Validity that hold for one-time query evaluation carry over to the continuous domain. Single-Site Validity is still feasible but, as we discuss next, a naive adaptation results in extremely weak semantics. The long-running characteristic of the query allows for $0 \ll t \leq T$. The resulting H_C considered over a long interval $[0, t]$ could easily become empty in a dynamic network. In such a case, Single-Site Validity degenerates to a trivial requirement: the answer received at t must be $v_t = q(H)$ for some set H such that $\phi \subseteq H \subseteq H_U$.

Keeping the computability results derived for one-time queries in mind, we require that v reflects the attribute values possessed by hosts reachable from h_q during a recent interval of width W. For the interval $[t - W, t]$, let $H_U = \cup_{i=t-W}^{t} H_i$ denote the set of hosts that were active at some instant during this interval. Let H_C be comprised of hosts that have at least one path to h_q stable during the above interval. We now insist that $v_t = q(H)$ for $H_C \subseteq H \subseteq H_U$. It is easy to see that no such algorithm can be devised for $W < max_{i=t-W}^{t}\{D_i\delta\}$. Then, for a sufficiently large and explicitly stated W, an algorithm must satisfy the following requirement.

Continuous Single-Site Validity: An algorithm that computes a continuous aggregate query must guarantee that $v_t = q(H)$ for some set H where $H_C \subseteq H \subseteq H_U$ defined over the interval $[t - W, t]$.

4.3 Approximate Query Semantics

The previous subsections have considered *precise* semantics for query results. There can be scenarios where a "dirty but quick" answer is acceptable for which efficient *approximate* algorithms can be designed. Nevertheless, we believe that the semantics should be fixed and the approximation quantified and conveyed to the user. Conversely, the user may insist on specifying the level of imprecision that can be tolerated. The following requirement can then be imposed on such approximate algorithms.

Approximate Single-Site Validity: An algorithm that computes an aggregate query must guarantee that the answer returned v satisfies $(1 - \varepsilon)q(H) \leq v \leq (1 + \varepsilon)q(H)$ with probability at least $1 - \zeta$ for some $H_C \subseteq H \subseteq H_U$, $0 < \varepsilon < 1$ and $0 < \zeta < 1$.

A simple modification of AllReport demonstrates that Approximate Single-Site Validity can be achieved in specific scenarios. Consider the problem of estimating the size $|H|$ of network G. In the modified algorithm (RandomizedReport), given parameters ε and ζ, h_q floods a message containing an additional parameter $p \geq \frac{4}{\varepsilon^2 n} ln \frac{2}{\zeta}$ during Broadcast. Each host that receives the message sends a 1 to h_q with probability p. Host h_q collects reports into a set M until time $T = 2\widehat{D}\delta$, and then declares the result to be $v = |M|/p$. The result can be shown to satisfy Approximate Single-Site Validity, and was obtained using $(1-p)|H|$ fewer messages than AllReport.

4.4 Analysis of Current Solutions

AllReport can be used to achieve one-time and continuous, and RandomizedReport to achieve approximate, Single-Site Validity. However, the two algorithms perform the "least" in-network processing possible. AllReport was studied as a naive solution (Direct Delivery) for query processing on Sensor Networks by Yao and Gehrke [35]. Experimental results showed that the number of messages needed to route attribute values to h_q across G and the bandwidth requirements imposed on hosts in the neighborhood of h_q are quite large. Direct Delivery thus pays a high price in terms of communication costs.

Can we do better than AllReport while achieving Single-Site Validity? Previous work suggests several algorithms that increase the "degree" of in-network processing to reduce communication costs. In such works, hosts are organized into an edge-subset network during Broadcast. The two most popular examples are SpanningTree [13, 21, 35, 36, 37] and DirectedAcyclicGraph [6, 21]. In a SpanningTree protocol, hosts are organized into a spanning tree rooted at h_q during Broadcast. In the Convergecast phase, relevant attribute values are selected at hosts, and partial aggregates are propagated up from the leaf hosts to h_q along the tree. Researchers observed that SpanningTree is sensitive to failures as there is a unique path along which the attribute value from each host is propagated to h_q. A DirectedAcyclicGraph protocol remedies this by providing each host with upto k parents, thus organizing the hosts into a directed acyclic graph.

Let us consider the communication costs of the above *best-effort* algorithms. Communication between hosts in both protocols requires small fixed-size messages only. The Broadcast phase requires $|E|$ messages to flood the query in G. The Convergecast phase requires $O(|H|)$ messages in SpanningTree Protocol and $O(k|H|)$ messages in DirectedAcyclicGraph Protocol.

It is easy to see that both these protocols forsake query semantics for communication efficiency. Example 1.1 presented an instance where SpanningTree fails to satisfy Single-Site Validity in the presence of host failures. A similar example can be constructed for DirectedAcyclicGraph as well. In fact, the following theorem shows that both protocols may return results that are arbitrarily bad.

THEOREM 4.4. *There exist instances of G, h_q and $e \geq 2$ when a run of* SPANNINGTREE *or* DIRECTEDACYCLICGRAPH *returns $v = q(H)$ where $H \subset H_C$ and $|H| \leq \frac{1}{e}|H_C|$.* ∎

PROOF. We will prove the above for SPANNINGTREE on an instance of G and h_q with error $e = 2$. The reader may verify that similar instances can be generated for $e > 2$ and runs of DIRECTEDACYCLICGRAPH. Consider a network G constructed as follows. Arrange $2n + 2$ hosts labeled $h_0, h_1, \ldots, h_{2n+1}$ in a cycle, with a host h_{2n+2} connected to the cycle at h_{n+1} by a solitary edge. Host h_0 initiates SPANNINGTREE which creates 2 chains rooted at h_0. Without loss of generality, let h_1 be the neighbor of h_0 in the longer chain. If h_1 fails after Broadcast, Convergecast will return $v = q(H)$ where H is the set of hosts in the smaller chain. For this run, $|H_C| = |H_U - \{h_1\}| = 2n + 2$ and $|H| \leq n + 1$ yielding $e = 2$. □

5. ACHIEVING SINGLE-SITE VALIDITY

We now present a simple in-network algorithm, WILDFIRE, that achieves Single-Site Validity for aggregation queries (*minimum, maximum, count, sum* and *average*). The protocol requires us to define new "duplicate-insensitive" *count* and *sum* operators which are proposed and analyzed in Section 5.2. Efficiency of the protocol is discussed in Section 5.3. Continuous Approximate Single-Site Validity for a class of *count* queries is considered in Section 5.4.

5.1 WILDFIRE Protocol

As before, our protocol can be separated into two phases: *Broadcast* and *Convergecast*. We say that a host is *inactive* at time t if it is not participating in the protocol at that time; otherwise it is *active*. At the start of the protocol, h_q is active, while all other hosts are inactive. Each active host h maintains a partial aggregate A_h which is initiated on transition to the active state. The initial value of A_h is set to be the relevant attribute value at the host h for *minimum* and *maximum* queries. We defer the discussion of initializing A_h for *count* and *sum* queries to Section 5.2.

PROTOCOL WILDFIRE-PHASE I
▷ On receipt of Broadcast message $[q, 0, \widehat{D}]$
▷ at $h \neq h_q$ from h' at time t
if (h *is inactive* & $t < 2\widehat{D}\delta$) {
Change state to *active*
Initialize A_h
Send Broadcast message $[q, 0, \widehat{D}]$ to neighbors
}

Figure 3: Broadcast phase in WILDFIRE Protocol

Host h_q initiates Broadcast by sending a message containing the query, the initiating time 0 and an overestimate of network diameter $\widehat{D}$ to all its neighbors. A host which receives the Broadcast message checks if it is *inactive*. If so, the host changes state to *active*, initializes A_h and sends the message to all its neighbors. Otherwise, the host merely drops the message. The Broadcast phase completes when all hosts have received the query and the two parameters. Notice that unlike previous works, Broadcast does not construct any edge-subset network.

PROTOCOL WILDFIRE-PHASE II
▷ On receipt of Convergecast message $[q, A_{h'}]$
▷ at h from h' at time t
if ($t \leq 2\widehat{D}\delta$) {
$A_h^{new} := Combine(q, A_h, A_{h'})$
▷ Propagate A_h^{new} in the network
if ($A_h^{new} \neq A_h$) {
$A_h := A_h^{new}$
Send Convergecast message $[q, A_h]$ to neighbors
} **else if** ($A_h \neq A_{h'}$) {
Send Convergecast message $[q, A_h]$ to h'
}
} **else** Terminate

Figure 4: Convergecast phase in WILDFIRE Protocol

A host transitions into Convergecast when it becomes active. An active host sends its partial aggregate to all its neighbors.[4] An active host h that receives a partial aggregate $A_{h'}$ from its neighbor h' recomputes its own partial aggregate A_h using a query-dependent "combine" function. The combine function for *minimum* and *maximum* queries is the query itself; the discussion for *count* and *sum* queries is deferred to Section 5.2. If the host detects a change in its partial aggregate, it sends the new A_h to all its neighbors. Each host continues to participate in Convergecast until $2\widehat{D}\delta$ time. At the end of Convergecast, h_q declares its partial aggregate to be the query result.

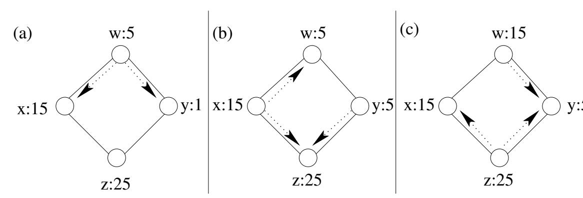

Figure 5: P2P Network with 4 hosts on which w initiates WILDFIRE to compute *maximum* value possessed. The messages sent at time instants (a) $t = 0$, (b) $t = 1$ and (c) $t = 2$ are shown.

EXAMPLE 5.1. *Consider the P2P Network with attribute values as shown in Figure 5(a). At time 0, host w initiates* WILDFIRE *to compute the maximum value possessed by hosts in the network. Host w becomes active, sets its partial aggregate $A_w = 5$ and sends a Broadcast message containing the query (*maximum*), initialization time 0, and an overestimate of the diameter $\widehat{D} = 3$ to x and y, piggybacking $A_w = 5$ on the message.*

At time 1, x and y receive the message, become active and set their partial aggregates $A_x = 15$ and $A_y = 1$. Host x now computes its new partial aggregate $A_x = max(A_w, A_x) = 15$. Host x forwards to z the Broadcast message with $A_x = 15$. Host x sends its A_x value to w as well. Similarly, host y forwards the Broadcast message to z with $A_y = 5$. Host y received its new A_y value from w, so it skips sending the value back to w.

[4]The first Convergecast message sent by a host to its neighbors can be piggybacked on the Broadcast message it sent.

At time 2, host z becomes active and sets $A_z = 25$. It computes its new $A_z = max(5, 15, 25)$, and sends $A_z = 25$ to x and y. Host w, meanwhile, computes its $A_w = max(15, 5)$, detects a change and sends $A_w = 15$ to y.

At time 3, host x computes its new $A_x = 25$ and sends it to w. Host y computes its new $A_y = max(15, 25, 5)$ and sends it to w as well.

At time 4, host w computes its new $A_w = max(15, 25)$. No messages are sent anymore, and at time $T = 2\widehat{D} = 2\times3 = 6$, w declares $v = A_w = 25$ as the query result. ∎

The reader may have noticed that w receives z's value twice. The end result is unaffected as *maximum* is duplicate insensitive. We also note that if either x or y had failed, w would still obtain z's value. If both x and y had failed, w would output $v = 5$, but this is acceptable as $H_C = \{w\}$. In fact, we can show that WildFire guarantees Single-Site Validity for duplicate-insensitive aggregate operators.

Theorem 5.1. WildFire *guarantees Single-Site Validity for* minimum *and* maximum *queries.* ∎

Proof. Consider a host h that possesses the answer (maximum or minimum attribute value) for the query (maximum or minimum query respectively). Let h be connected to h_q through a stable path P of length at most L. The Broadcast message takes time at most $L\delta$ to reach h, and the attribute value at h takes time at most $L\delta$ to reach h_q. Every host $h' \in P$ will transmit the Broadcast message since there are no failures on P. Similarly, every host h' on path P will transmit the value from h back to h_q. If host $h \in H_C$, then by definition, h has a path P to h_q with $L \leq \widehat{D}$. Further, only hosts $h \in H_U$ participate in query processing implying $v = q(H)$ for some $H_C \subseteq H \subseteq H_U$. □

5.2 Duplicate-Insensitive Aggregate Operators

The hurdle in using WildFire to compute *count* and *sum* aggregates is that the conventional combine function (+) for both is duplicate sensitive. We now propose duplicate-insensitive combine functions for *count* and *sum* adapted from an algorithm by Flajolet and Martin [12]. We begin with a description of the original algorithm (FM).

The FM algorithm takes as input a set M of values drawn from a domain V, and outputs an estimate of the number of distinct elements in M. Before counting, c vectors $B_1, B_2, \ldots, B_c$ of size $O(\log|V|)$ bits each are initialized to 0. The algorithm also generates c random functions $map_1, map_2, \ldots, map_c$ such that each $map_i : V \mapsto [0, 2\log|V|]$. Each function map_i has an exponential distribution: half the elements in V are mapped to 0, a quarter to 1, an eighth to 2, and so on. The algorithm then makes a single pass over M and for each element $v \in M$ and each $B_i, 1 \leq i \leq c$, the corresponding $b_v = map_i(v)$ bit is set to 1. At the end of the pass, the lowest-order bit z_i in each B_i that is still 0 is identified. The average value $\overline{z} = \sum_{i=1}^{c} z_i/c$ is computed and $2^{\overline{z}}/0.78$ is returned as the answer.

Lemma 5.1. (Alon et. al. [3]) *For every $c > 2$, given a set M of elements drawn from a set V of size n, the* FM *algorithm outputs an estimate $\widehat{m}$ of the number of distinct elements m in M such that $Pr(\frac{1}{c} \leq \frac{\widehat{m}}{m} \leq c) \geq 1 - \frac{2}{c}$.* ∎

Observe that FM sets one specific bit for each element $v \in M$ in each of the c vectors $B_1, B_2, \ldots, B_c$. An equivalent process is to create c vectors $B_1^v, B_2^v, \ldots, B_c^v$ (each with a specific bit set to 1) for each element $v \in M$, and then logically OR the $|M|$ vectors to obtain the final $B_1, B_2, \ldots, B_c$.

The above observation forms the basis of our adaptation of FM for distributed *count*. The input to this distributed procedure consists of a set M of values distributed across $|M|$ hosts. Host h_q initiates Broadcast and includes parameter c in its message. On receipt of the Broadcast message, each host creates c vectors $B_1^h, B_2^h, \ldots, B_c^h$. Each host now pretends to have an element distinct from other hosts by simulating $map_1, map_2, \ldots, map_c$ as follows. A total of c fair coin ($Pr[Head] = Pr[Tail] = 0.5$) toss sequences are generated, each of which ends when the first Head in the sequence is observed. Host h sets b_i bit in B_i^h to 1 where b_i is the index of the last Tail in the i^{th} sequence. Note that the lengths of coin toss sequences have an exponential distribution as required by FM: half the hosts have $b_i = 0$, a quarter have $b_i = 1$, an eighth have $b_i = 2$, and so on.

The final vectors $B_1, B_2, \ldots, B_c$ are to be obtained by performing a logical OR of the corresponding vectors across the M hosts. Since OR is a duplicate-insensitive operator, we can use WildFire to assure Single-Site Validity of the final vectors. Each active host initializes its partial aggregate $A_h := (B_1^h, B_2^h, \ldots, B_c^h)$. On transition into the Convergecast phase, each host sends its A_h to its neighbors. On receiving a partial aggregate from its neighbor, a host recomputes its new A_h using logical OR of the vectors as the combine function. The reader may note that by the end of Convergecast, the partial aggregate at h_q will be the final $(B_1, B_2, \ldots, B_c)$ vectors. At the end of Convergecast, h_q identifies the lowest-order bits z_i in B_i that are still 0. The average value $\widehat{z} = \sum_{i=1}^{c} z_i/c$ is computed and $2^{\widehat{z}}/0.78$ is returned as the *count*.

Our distributed *sum* procedure is similar to our *count* adaptation of FM. The *sum* takes as input a set M of values drawn from $[0, V]$ distributed across $|M|$ hosts, with one value h at each host. The final output is an estimate of the sum of elements in M. Host h_q initiates Broadcast and includes parameters c and $\log M$ in its message. On receipt of the message, a host creates c vectors $B_1^h, B_2^h, \ldots, B_c^h$ of size $O(\log|M||V|)$ each. Each host now pretends to have h elements distinct from other hosts and runs the *count* procedure h times. Let the *count* procedure for the i^{th} $(1 \leq i \leq h)$ element generate vectors $B_1^{h,i}, B_2^{h,i}, \ldots, B_c^{h,i}$. The host then sets its vectors $B_1^h, B_2^h, \ldots, B_c^h$ to be a logical OR of the *count* vectors: $B_j^h = \vee_{i=1}^{h} B_j^{h,i}$ where $1 \leq j \leq c$. Once the vectors are initialized, hosts participate in Convergecast as before. At the end of Convergecast, h_q computes $\widehat{z} = \sum_{i=1}^{c} z_i/c$ as before and reports $2^{\widehat{z}}/0.78$ as the *sum*.

The *sum* procedure requires h_q to include an estimate l_M of $\log|M|$ in its Broadcast message. This parameter is used by hosts merely to fix the size of their bit-vectors, and has no bearing on the accuracy of the final answer as long as $l_M \geq \log|M|$. Host h_q can thus overestimate l_M (e.g., 32). Since $\log|M|$ grows slowly with M, and $M \subseteq H$, $l_M = 32$ will not be adequate only if the size of network $|H| > 2^{32}$.

Theorem 5.2. *For every $c > 2$, given a set H, our* count *and* sum *procedures output an estimate $\widehat{v}$ of the actual value v such that $Pr[\frac{1}{c} \leq \frac{\widehat{v}}{v} \leq c] \geq 1 - \frac{2}{c}$.* ∎

Proof. The proof for the *count* procedure follows from Lemma 5.1 when we observe that the procedure counts the number of distinct elements in M. The *count* procedure can be used to obtain *sum* if we view value m at a host as contributing m distinct elements to the input set M. A direct application of *count* would require each host to communicate m bit-vectors. Instead, each host h can perform a logical OR of its local m bit-vectors to produce a single set of vectors $B_1^h, B_2^h, \ldots, B_c^h$ that represent m elements as indicated. □

THEOREM 5.3. *The* WILDFIRE$(q, \widehat{D})$ *algorithm guarantees Approximate Single-Site Validity within a factor c with probability at least $1 - \frac{2}{c}$ for the class of* count, sum *or* average *queries.* ■

5.3 Discussion

The WILDFIRE protocol discussed in Section 5.1 provides a framework for processing aggregate queries in dynamic networks. The ingenuity lies in selecting appropriate combine operators to ensure small messages and duplicate-insensitive processing. We presented such operators for *count* and *sum* queries. Can such operators be designed for other interesting classes of queries? In a recent work, Kempe et. al. [18] explored such operators for estimating join sizes, L_p norms and histograms while designing novel gossip-based algorithms. We believe that investigating the applicability and performance of such work in our framework will be an interesting direction of future research.

The concern, at first glance, is in the inefficiency of WILDFIRE protocol. In the worst case, every host will observe an update to its partial aggregate at every time instant during the query processing interval. Each update causes a host to propagate its new partial aggregates to its neighbors causing a worst-case traffic of $2\widehat{D}|E|$ messages as opposed to $|E| + |H|$ in SPANNINGTREE.

Experiments on real-life networks (Section 6), however, demonstrate that such worst-case behavior is rarely observed as early aggregation reduces updates at hosts. In addition, WILDFIRE can be engineered to improve efficiency: a host at distance l from h_q can continue in the Convergecast phase until $(2\widehat{D} - l + 1)\delta$ time instead of the $2\widehat{D}\delta$ time indicated in Figure 4. WILDFIRE can also leverage domain capabilities: the broadcast ability of wireless medium in Sensor Networks allows a host to send its partial aggregate to its neighbors by a single message, reducing worst-case cost to $2\widehat{D}|H|$.

5.4 Continuous Approximate Count Queries

We now turn our attention to scenarios where a "quick and dirty" answer is acceptable. We consider the problem of estimating the size of network. An accurate estimate of $|H|$ can be computed using *count* with WILDFIRE which requires $O(|E|)$ messages. Are there valid schemes that return a coarser answer albeit at a lesser price?

We can design schemes specific to a protocol to produce such estimates efficiently. For example, some P2P protocols [22, 31, 33] assign random identifiers to hosts and place them along a ring. Each host manages a segment on the ring between its own identifier and that of its immediate clockwise predecessor. A network size estimate on these networks can be deduced using the following insight [22]. Let X_s denote the sum of segment lengths managed by a sample set of s hosts. Then $\frac{s}{X_s}$ is an unbiased estimator for $|H|$. It can be shown that such an estimate satisfies Approximate Single-Site Validity under the following assumptions:

1 Sampling is "instantaneous" with respect to host lifetimes.
2 Every host in the network has the same probability of leaving the network at each instant.

Are there schemes for processing size estimate queries in dynamic networks that are not protocol-specific? We now present one that enables Continuous Approximate Single-Site Validity for estimates of $|H|$ and makes minimal assumptions on the network protocol. The key insight behind the scheme is to view dynamic networks as an "evolving ecology". Ecologists have long studied models for animal abundance and its dynamics. The Jolly-Seber model for interpreting Capture-Recapture experiments in open ecologies has occupied a central place in such studies. We outline an interpretation of the model in our context, directing the interested reader to an excellent monograph by Pollock et. al. [28] for an exact analysis and experimental studies.

The scheme assumes that the network protocol provides a "black-box" operation, which when invoked from h_q, returns s random hosts from $H \supseteq H_C$. For example, some P2P networks have expander graph topologies [19, 26]. On such graphs, the operation would perform s random walks of length $O(\log |H|)$ each to reach s random hosts. Specifically, the scheme makes the following assumptions:

1 Every host in the network has the same probability of occurring in a sample.
2 Sampling is "instantaneous" with respect to host lifetimes.
3 Every host in the network has the same probability of leaving the network at each instant.

The scheme samples hosts at periodic intervals. Let the sample sets of hosts be $N_1, N_2, \ldots$, where N_t is the sample set taken at the *end* of the t^{th} interval. Let M_t denote the set of hosts in N_t that also occur in N_i for $i < t$. Using this dataset, the scheme estimates $|\widehat{H}|_t$ which is the size of the network at the *beginning* of the t^{th} interval (or after the *end* of the $(t-1)^{th}$ interval) on obtaining sample N_{t+1}. Thus, it estimates network size in the window $[t-1, t+1]$.

Consider the set N_2 that was sampled at the end of second interval. Intuitively, we can use the uniform sampling assumption to derive a network size estimator based on the notion that the ratio {number of hosts in the current sample that had also been sampled earlier} : {size of the current sample} should reflect the same ratio in the network. Thus,

$$\frac{|M_2|}{|N_2|} = \frac{|N_1|}{|\widehat{H}|_2} \text{ or } |\widehat{H}|_2 = \frac{|N_2||N_1|}{|M_2|}$$

Similarly, at the t^{th} sampling,

$$\frac{|M_t|}{|N_t|} = \frac{|\cup_{i=1}^{t-1} N_i|}{|\widehat{H}|_t} \text{ or } |\widehat{H}|_t = \frac{|N_t||\cup_{i=1}^{t-1} N_i|}{|M_t|}$$

All of the above estimates can be worked out if we knew $|\cup_{i=1}^{t-1} N_i|$. This quantity would be precisely known if there were no host leaves during the t intervals. In a dynamic network, the above ratios are actually defined by the number of previously sampled hosts that are still *alive* ($|\cup_{i=1}^{t-1} N_i^a|$). To estimate this quantity, consider the set of hosts that were sampled earlier but not at t ($\cup_{i=1}^{t-1} N_i^a - M_t$). By construction, this set is disjoint from the set of hosts that were sampled at t (N_t). Now, we use the intuition that the *future* sampling rates of these two distinct groups of sampled hosts must be the same. Let Y_i and Z_i refer to the set of hosts that are sampled in the future from the first and second groups respectively. Then,

$$\frac{|Y_t|}{|\cup_{i=1}^{t-1} N_i^a - M_t|} = \frac{|Z_t|}{|N_t|} \text{ or } |\cup_{i=1}^{t-1} N_i^a| = |M_t| + \frac{|N_t||Y_t|}{|Z_t|}$$

6. EMPIRICAL EVALUATION

In this section, we compare WildFire, SpanningTree and DirectedAcyclicGraph protocols on real-life and synthetic networks. We also evaluate the accuracy of duplicate-insensitive *count* and *sum* (Section 5.2) and the Capture-Recapture scheme (Section 5.4). Our implementation of DirectedAcyclicGraph uses the distributed *count* and *sum* operators; WildFire uses the two optimizations discussed in Section 5.3. Our simulations show the following results.

- The *sum* and *count* operators show high accuracy with a few repetitions (small c values).
- WildFire satisfies Single-Site Validity across different degrees of dynamism while SpanningTree and DirectedAcyclicGraph deteriorate rapidly.
- WildFire pays a $5\times$ higher price for its semantics as compared to the efficient best-effort SpanningTree.
- WildFire pays most of the price early in Convergecast. This observation leads to fine tuning heuristics.
- The Capture-Recapture scheme satisfies Continuous Approximate Single-Site Validity.

Network Topology

Each host h in G possesses an attribute value that is drawn from a Zipfian distribution in the range $[10, 500]$. The network G was set to be one of the following.

A *Gnutella* is a real-life network topology with $|H| = 39,046$ obtained from a crawl of Gnutella [9].

B *Random* is a synthetic network topology with $|H| = 40K$ constructed by placing an edge between pairs of hosts with uniform probability such that average degree is 5.

C *Power-law* is a synthetic network topology with $|H| = 40K$ constructed to have a power-law distribution ($\gamma = 2.9$) in host degrees [4].

D *Grid* is a Sensor Network synthetic topology constructed by placing $|H| = 10K$ hosts in a 100×100 grid. Each host has hosts in the enclosing 2-unit square as its neighbors.

Dynamism Model

Single-site validity requires that $v = q(H)$ for some $H_C \subseteq H \subseteq H_U$. Of the two bounds, H_C is the more interesting, as hosts that join G *after* h_q initiated Broadcast may or may not contribute to v. Hence we do not model host joins in our simulations. We model host failures by removing a total of R randomly selected hosts from G at a uniform rate during $[t_0, t_n]$. The value of R is varied from 256 to 4,096 to characterize different degrees of dynamism.

As a frame of reference, an Oracle was devised that observes all events in G. The Oracle detects reachability of each host from h_q using which it computes H_C and H_U as the lower and upper bounds of Single-Site Validity. Clearly, such an Oracle is not feasible in practice.

Efficiency Measures

We consider the following measures of performance while evaluating the efficiency of a protocol.

A *Communication cost* of a protocol is the number of messages sent between any host pairs (h, h') in E. We note that all the protocols considered here involve fixed size messages. The communication cost is thus proportional to the actual byte traffic generated by the protocol.

B *Computation cost* at a host h is the number of messages processed at h. The computation cost of a protocol is the *maximum* computation cost among all hosts in G.

C *Time cost* of a protocol is the length of the longest chain of messages that occurs before the protocol terminates. The chain starts at h_q with the initiation of Broadcast. Each subsequent message in the chain is generated on the receipt of the preceding one.

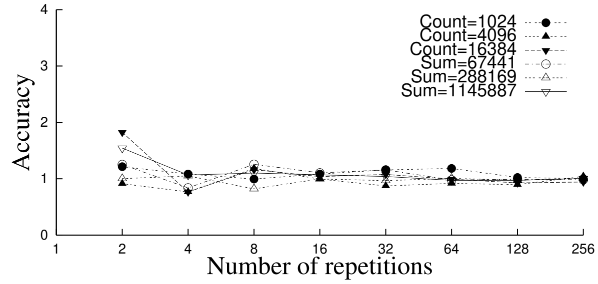

Figure 6: Accuracy of *Count* and *Sum* operators

Accuracy of Sum and Count Operators

We start with an evaluation of the accuracy of *count* and *sum* operators. We generated a set M of Zipf-distributed elements in the range $[10, 500]$ with $|M|$ as $2^{10}, 2^{12}$ and 2^{14} to simulate different orders of magnitude of operands. On this set M, we used *count* to estimate the number, and *sum* to estimate the total sum of elements. The ratio of estimate returned $\widehat{m}$ to the actual answer m represents the accuracy of the operators. Values of $\widehat{m}/m < 1$ depict an underestimate, $\widehat{m}/m > 1$ depict an overestimate, while $\widehat{m}/m = 1$ represents perfect accuracy. Figure 6 shows the accuracy ratios of operators (Y-axis) against the number of repetitions c performed (X-axis). We observe that the ratio quickly converges to 1 as c increases, indicating that *count* and *sum* produce accurate estimates. Moreover, the number of repetitions required are small (≈ 8), implying that WildFire can be used for *count* and *sum* queries with small values of c. A small c value leads to a correspondingly small cost in computing the c bit-vectors at each host, and small sized messages (containing the c B_i values each of size $32b$) sent during Convergecast.

Achieving Single-Site Validity

We next study the behavior of SpanningTree, DirectedAcyclicGraph and WildFire protocols under different degrees of dynamism in the network. Figures 7 and 8 show the query result returned v (Y-axis) against number R of host leaves (X-axis) for *count* and *sum* queries respectively on the *Gnutella* topology. The average answers returned by the protocols over 10 trials are plotted with a 95% confidence interval. The DirectedAcyclicGraph points are drawn for $k = 2$ and $k = 3$ parents per host.

The curves labeled Oracle show the upper and lower bounds for Single-Site Validity. Protocols that guarantee Single-Site Validity will return values within the two bounds. A value $v = q(H)$ less than the lower bound indicates $H \subset H_C$, i.e., there are hosts $(H_C - H)$ that were not included in v even though they were in G during query processing.

We observe that all protocols return similar counts for low dynamism (small R) in Figure 7. As dynamism increases, both SpanningTree and DirectedAcyclicGraph fall rapidly off the lower bound for Single-Site Validity. WildFire continues to return values within the Single-Site Validity bounds

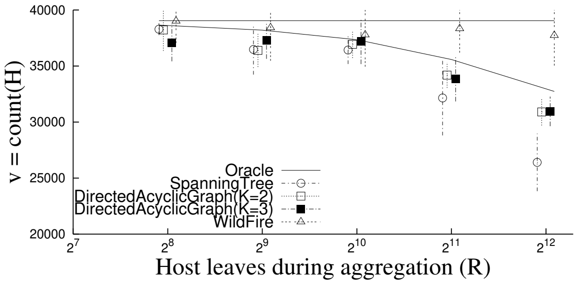

Figure 7: *Count* query on the *Gnutella* topology

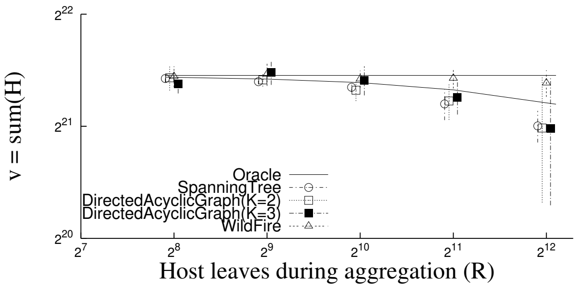

Figure 8: *Sum* query on the *Gnutella* topology

even for high dynamism rates (with nearly 10% of hosts leaving the network). DirectedAcyclicGraph does improve over SpanningTree but is unable to return valid answers as dynamism increases. The protocols behave similarly for $v = sum(H)$ queries as seen in Figure 8. Notice that the Y-axis here has a logarithmic scale which causes the points for SpanningTree and DirectedAcyclicGraph to appear closer to the lower bound.

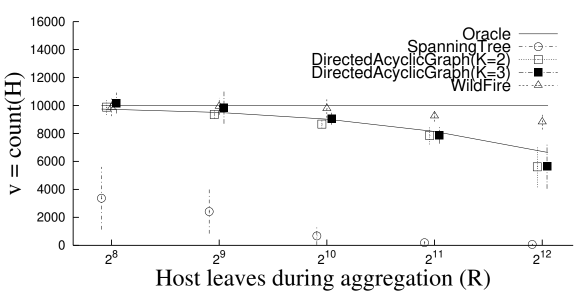

Figure 9: *Count* query on the *Grid* topology

We found that WildFire continues to satisfy Single-Site Validity, while SpanningTree and DirectedAcyclicGraph struggle to keep up with increasing dynamism on all of our synthetic topologies (*Random, Power-Law* and *Grid*). For example, Figure 9 plots results for *count* queries on *Grid*. We observe that SpanningTree performs extremely poorly which can be explained as follows. A spanning tree built on *Grid* has a large depth, with most of the hosts occupying interior positions in the tree. A removal of any of these interior hosts causes the non-inclusion of the entire sub-tree rooted at that host. As R increases, query results deteriorate rapidly.

Price of Single-Site Validity

Recall that SpanningTree and DirectedAcyclicGraph are *best-effort* protocols designed to optimize communication costs. WildFire, on the other hand, guarantees Single-Site Validity. What price does WildFire have to pay for such valid semantics? We compared the relative performance of protocols on synthetic topologies for various network sizes; the communication costs are shown in Figures 10 and 11. The Y-axis here plots the number of messages sent against the size $|H|$ of network on the X-axis.

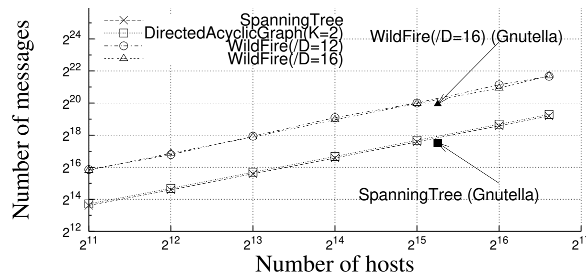

Figure 10: Communication costs on *Random*

Figure 10 shows results for a *count* query on *Random* topologies. The reader may remember that WildFire requires $\widehat{D}$ as an input parameter, and executes for $2\widehat{D}\delta$ time. We varied the input $\widehat{D} > D$ to study the effects of an overestimate on communication costs. However, we observe that the WildFire curves for different $\widehat{D}$ (top curves) overlap, indicating that communication costs are independent of $\widehat{D}$. The DirectedAcyclicGraph curve almost overlaps SpanningTree (bottom curves) as the cost of Broadcast ($|E|$) dominates that of Convergecast ($k|H|$). We observe that WildFire, on the other hand, incurs a $4\times$ communication cost over the efficient SpanningTree.

Figure 10 also shows communication costs incurred by SpanningTree and WildFire on the lone *Gnutella* topology we had access to. WildFire incurs $4\times$ the costs for SpanningTree. A similar scaling was also observed on *Power-Law* topologies, suggesting that the protocols might have analogous ratios on real-life networks.

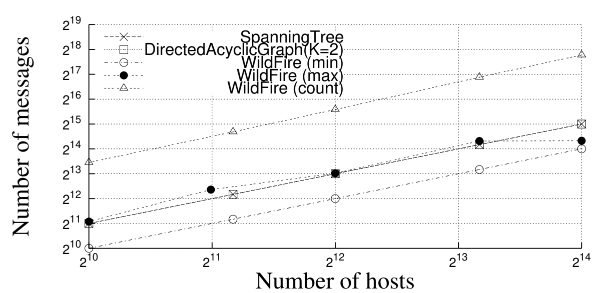

Figure 11: Communication costs on *Grid*

Figure 11 shows the results for queries on *Grid* topologies. Hosts here were simulated to have broadcast capabilities, allowing a host h to send a message to all its neighbors via a single message. Hence we observe that the DirectedAcyclicGraph curve overlaps SpanningTree as the cost of sending messages to $k \geq 1$ parents is the same. WildFire here incurs $5\times$ the SpanningTree costs for *count* queries.

Figure 11 also shows that the cost for *maximum* query on WildFire is smaller than that for *count*. In fact, the costs for *minimum* are smaller than those for SpanningTree! This interesting result can be attributed to the early aggregation performed by WildFire. The reader may have deduced that the SpanningTree and DirectedAcyclicGraph will send the same number of messages irrespective of the query type (e.g., a *sum* query will incur the same communication cost as a *minimum* query). Communication costs for WildFire, however, depend both on the data distribution as well as the query type. Convergecast in WildFire starts as soon as a host becomes active, before Broadcast has completed. The Broadcast carries with it the *minimum* value possessed by active hosts. A host that receives a smaller value than its own attribute value during Broadcast does not send its own attribute value. In contrast, each host *must* send a message during Convergecast for SpanningTree, leading to a higher observed cost.

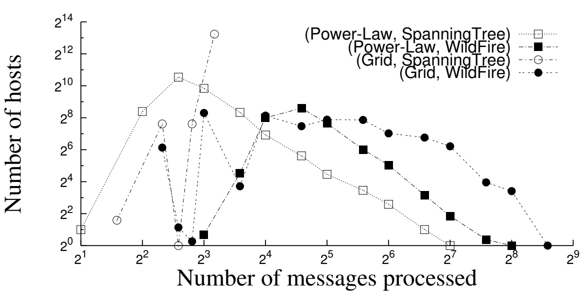

Figure 12: Computation cost on *Power-Law* and *Grid*

Computation Cost: Figure 12 plots the number of hosts (Y-axis) for each value of computation cost of a host (X-axis) observed during an evaluation of a *count* query. For example, the number of hosts that processed $X = 2^2$ messages is given by the corresponding Y value on the curves. The maximum number of messages processed by a host (maximum non-zero X value) represents the computation cost of a protocol. We observe that on *Power-Law*, WildFire pays a computation cost 2× that of SpanningTree; on *Random* (not shown in the figure) the cost was observed to be 4×. On both these topologies, WildFire had a similar shaped (distribution) curve as SpanningTree, except that the curve is shifted right to account for the larger number of messages sent.

The worst performance is observed on *Grid*, where the computation cost for WildFire is 44×, i.e, there are hosts in *Grid* which process 44× more messages during WildFire. We observed earlier (Figure 11) that communication costs in WildFire are 5× that of SpanningTree. The difference in numbers can be explained as follows. In SpanningTree, a message sent by a host during Convergecast is computed only by a single parent. In WildFire, a message sent by a host has to be received and computed by each neighbor. Thus an increase in communication cost results in a larger increase in computation costs.

Time Cost: Figure 13(a) shows the time cost of protocols (Y-axis) against network size (X-axis) on *Random* topologies. SpanningTree provides the least latency. WildFire returns a result at $t_0 + 2\widehat{D}\delta$ time, which is a constant for a given $\widehat{D}$. We observe that increasing $\widehat{D}$ increases time

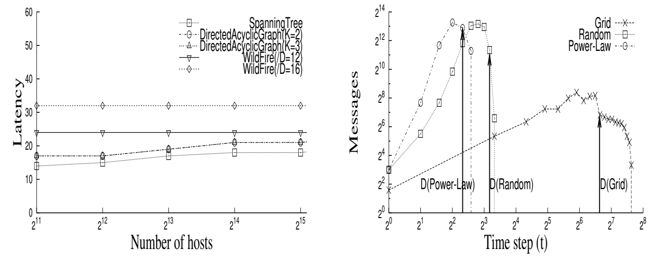

Figure 13: (a) Time cost on *Random*, (b) Number of messages sent by WildFire at each time instant.

cost proportionally. Recall that an overestimate of $\widehat{D}$ was observed to have no effects on the communication costs for WildFire. The two observations can be explained using Figure 13(b) which plots the number of messages sent (Y-axis) per time instant (X-axis) by WildFire for a *count* query on the synthetic topologies. In each topology, the number of messages peaks close to $D\delta$ (indicated by arrows), and decreases to 0 by $2D\delta$ time. This means that during the overestimate period $[2D\delta, 2\widehat{D}\delta]$, no messages are sent in G and hence the partial aggregate at h_q remains unchanged. Yet, h_q cannot declare a result until $2D\delta$ time has elapsed. Thus, time costs suffer while communication costs remain unaffected due to an overestimate of $\widehat{D}$.

A user may decrease query latency by making $\widehat{D}$ closer to D. How can such a good $\widehat{D}$ be deduced? A heuristic is to initially use WildFire itself with a large $\widehat{D}$ to find the *maximum* D among hosts in G, and then use the result to construct $\widehat{D}$ for subsequent queries.

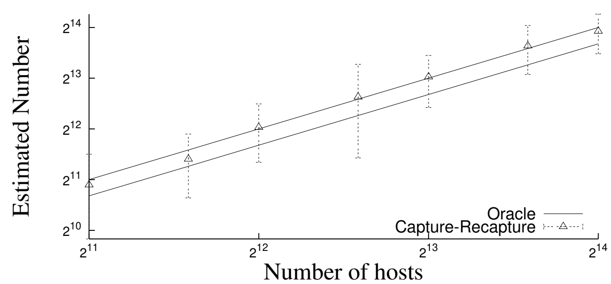

Figure 14: Network size estimate query using the Capture-Recapture scheme

Continuous Approximate Single-Site Validity

We next study the behavior of the Capture-Recapture scheme for estimating the size of the network. Recall that the scheme assumes "instantaneous" uniform-random sampling of hosts to estimate $|\widehat{H}|_t$, the number of hosts in the window $[t-1, t+1]$. We simulated runs of the scheme for various actual sizes of the network in which 10% of the hosts left the network in each interval. Figure 14 plots the estimates returned by the scheme at the end of $(t+1)^{th}$ interval on the Y-axis against the actual size of the network at start of the t^{th} interval on the X-axis. The curves labeled Oracle show the upper and lower bounds of network size in the window. The estimates returned by Capture-Recapture scheme are plotted with 95% confidence intervals and are seen to lie close to the Single-Site Validity bounds.

In a separate set of experiments, we also observed that the variance of estimates returned by Capture-Recapture can be decreased by increasing the sample size. A linear sized sample ($s = f|H|; 0 < f < 1$) was found to return good results; Figure 14 uses $f = 0.1$. The cost of the scheme is thus $O(|H|)$ instead of $O(|E|)$ incurred by WildFire. We also note that the variance of estimates returned by Capture-Recapture is larger than that of WildFire. The two schemes thus provide a trade-off option between communication efficiency and result accuracy.

A continuous query processing scheme must provide a feedback mechanism to adjust sample sizes if the network grows or shrinks substantially during the monitoring period. How can we bootstrap a good sample size and adjust it while processing? We could: (1) Start processing the query with an initial sample size, and then iteratively increase (or decrease) the sample size after each interval until variance of the answer decreases. (2) Use WildFire at the start of the continuous query and at periodic intervals to obtain an accurate network size which is then used to set an appropriate sample size.

7. SUMMARY AND FUTURE WORK

Massive-scale self-administered networks like Peer-to-Peer and Sensor Networks have data distributed across thousands of participant hosts. These networks are highly dynamic with short-lived hosts being the norm rather than an exception. In such networks, we would like query processing to continue even if a “few” hosts fail. Hence, it becomes necessary to give a meaning to possible interleaving of host failures with query processing.

We defined *Single-Site Validity*, a correctness requirement imposed on in-network query processing algorithms, and discussed its extensions to the continuous and approximate domains. We proposed the WildFire protocol that achieves Single-Site Validity for duplicate-insensitive queries. We showed how common aggregate queries (*maximum, minimum, count, sum* and *average*) can be adapted to work with WildFire. The protocol pays a 5× price over the most efficient best-effort SpanningTree algorithm. We presented the Capture-Recapture scheme that achieves Continuous Approximate Single-Site Validity at a lower cost.

We believe this work can be extended in several interesting directions. We intend to investigate ways of improving efficiency of WildFire and its adaptation for achieving Continuous Single-Site Validity. Another promising direction of future work is the design of duplicate-insensitive operators for complex aggregation queries.

8. REFERENCES

[1] D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases. In *Proc. of PODS*, 1997.

[2] R. Albert, H. Jeong, and A.-L Barabasi. Diameter of the world wide web. *Nature*, 401, 1999.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. of STOC*, 1996.

[4] A.-L Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286, 1999.

[5] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of MDM*, 2001.

[6] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proc. of ICDE*, 2004.

[7] D. De Couto, D. Aguayo, B. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. In *Proc. of HotNets*, 2002.

[8] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. of PODC*, 1987.

[9] The dss clip2 company.

[10] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proc. of OSDI*, 2002.

[11] D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proc. of MOBICOMM*, 1999.

[12] P. Flajolet and G. N. Martin. Probabilistic counting. In *Proc. of FOCS*, 1983.

[13] I. Gupta, R. van Renesse, and K. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proc. of DSN*, 2001.

[14] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proc. of SIGMOD*, 1997.

[15] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *Proc. of VLDB*, 1992.

[16] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of MOBICOMM*, 2000.

[17] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for smart dust. In *Proc. of MOBICOMM*, 1999.

[18] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. of FOCS*, 2003.

[19] P. Keyani, B. Larson, and M. Senthil. Peer pressure: Distributed recovery from attacks in peer-to-peer systems. In *Proc. of Networking*, 2002.

[20] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proc. of STOC*, 2000.

[21] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *Proc. of OSDI*, 2002.

[22] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. of PODC*, 2002.

[23] S. Milgram. The small world problem. *Psychology Today*, 1, 1967.

[24] D. L. Mills. Network time protocol (version 2) specification and implementation, 1989.

[25] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *Proc. of SIGKDD*, 2002.

[26] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter p2p networks. In *Proc. of FOCS*, 2001.

[27] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of WMCSA*, 1999.

[28] K. H. Pollock, J. D. Nichols, C. Brownie, and J. E. Hines. Statistical inference for capture-recapture experiments. *Wildlife Society Monographs*, 107, 1990.

[29] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5), 2000.

[30] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *Internet Computing Journal*, 6(1), 2002.

[31] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems. In *Proc. of Middleware*, 2001.

[32] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of MMCN*, 2002.

[33] I. Stoica, R. Morris, D. Karger, M. Fran Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM*, 2001.

[34] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management and data mining. *ACM Transaction on Computer Systems*, 21(2), 2003.

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

[36] Z. Zhang, S. Shi, and J. Zhu. SOMO: Self-organized metadata overlay for resource management in p2p dht. In *Proc. of IPTPS*, 2003.

[37] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proc. of SNPA*, 2003.