Robust Monitoring of Network-wide Aggregates through Gossiping

Fetahi Wuhib, Mads Dam, Rolf Stadler
KTH Royal Institute of Technology
Stockholm, Sweden
{fetahi, mfd, stadler}@kth.se

Alexander Clemm
Cisco Systems
San Jose, California, USA
alex@cisco.com

***Abstract*— We examine the use of gossip protocols for continuous monitoring of network-wide aggregates. Aggregates are computed from local management variables using functions such as AVERAGE, MIN, MAX, or SUM. A particular challenge is to develop a gossip-based aggregation protocol that is robust against node failures. In this paper, we present G-GAP, a gossip protocol for continuous monitoring of aggregates, which is robust against discontiguous failures (i.e., under the constraint that neighboring nodes do not fail within a short period of each other). We formally prove this property, and we evaluate the protocol through simulation using real traces. The simulation results suggest that the design goals for this protocol have been met. For instance, the tradeoff between estimation accuracy and protocol overhead can be controlled, and a high estimation accuracy (below some 5% error in our measurements) is achieved by the protocol, even for large networks and frequent node failures. Further, we perform a comparative assessment of G-GAP against a tree-based aggregation protocol using simulation. Surprisingly, we find that the tree-based aggregation protocol consistently outperforms the gossip protocol for comparative overhead, both in terms of accuracy and robustness.**

I. INTRODUCTION

The motivation of this research is to investigate the use of gossip protocols for decentralized real-time monitoring. Recent research in gossip protocols suggests to us that these types of protocols may help engineering a new generation of monitoring systems that are highly scalable and fault tolerant.

Gossip protocols, also known as epidemic protocols, can be characterized by asynchronous and often randomized communication among nodes in a network [10][3]. Originally, they have been proposed for disseminating information in large dynamic environments [10], and more recently, they have been applied for various tasks, including constructing robust overlays [1][17], estimating the network size [9][6], etc.

We are specifically interested in assessing the use of gossip protocols for decentralized aggregation of device data in near real-time. Aggregation functions, commonly used by management applications, include SUM, MAX and AVERAGE of device-level counters and other variables. Examples of such aggregations are:

(a) average load across all network links;

(b) the number of active voice calls in a given domain.

Specific applications that require such information include network surveillance, service assurance, and traffic control in large-scale or dynamic networks. Admission control, for example, can make use of cross-device network load and QoS measurements to decide whether to accept or reject flows into a network domain.

A gossip protocol for monitoring network-wide aggregates executes in the context of a decentralized management architecture. Figure 1 shows an example of such an architecture, which we propose using for this purpose. In this architecture, monitoring nodes with identical functionality organize themselves into a management overlay. The aggregation protocol (in this case, the gossip protocol) runs in the monitoring nodes, which communicate via the overlay. Each monitoring node collects data from one or more network devices. This data is aggregated, in a decentralized fashion, to estimate the MIN, MAX, SUM, AVERAGE, etc., of the device variables through the aggregation protocol. A management station or an application server can access the monitoring layer at any node. Node or link failures—on the physical network or the management overlay—trigger a re-organization of the management overlay, thereby enabling continuous operation. (Note that the protocol introduced in this paper can also run on a different architecture, as long as the architecture includes monitoring nodes that execute the protocol, it provides functions in the monitoring nodes to access local device variables, and it maintains an overlay topology for monitoring nodes to exchange information.)

Recently, other approaches to decentralized aggregation, which are based on creating and maintaining spanning trees in the management overlay, have been investigated by others [12][13][14], and also by us [4][15][16][18]. There are qualitative and quantitative differences between tree-based and gossip-based aggregation. First, gossip-based aggregation protocols tend to be simpler in the sense that they do not maintain a distributed tree in the management overlay. Second, in tree-based aggregation, the result of an aggregation operation is available on the so-called root node of the tree, while in gossip-based aggregation, the result is available on all nodes. Third, failure handling is very different for tree-based aggregation than for gossip-based aggregation. If a node fails, a tree-based aggregation protocol needs to reconstruct the aggregation tree for which there are well understood techniques. In gossip protocols, node failure can produce mass

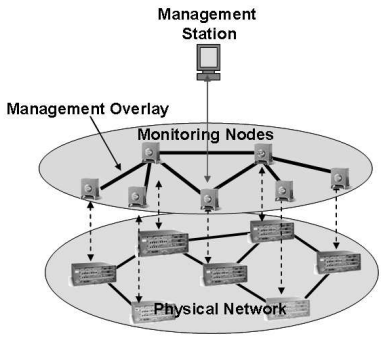

FIGURE 1: ARCHITECTURE OF THE DECENTRALIZED MONITORING SYSTEM. GOSSIP PROTOCOLS RUN IN THE MANAGEMENT OVERLAY (MIDDLE LAYER)

loss (which is further explained in section III). Unfortunately, the problem of mass loss has not been sufficiently studied to date. As a consequence, there is no gossip protocol for monitoring aggregates available today that is robust to node failures. We conclude that, in order to perform a comparative assessment of tree-based and gossip-based aggregation, the problem of mass loss needs to be addressed first.

In this paper, we extend a gossip protocol in [3], which computes a snapshot of the network aggregate at a specific time, to a protocol that continuously produces an estimate of the aggregate as it changes over time. More importantly, we extend the protocol in such a way that certain mass invariants hold, which makes the protocol robust against many classes of node failures. We call the resulting protocol G-GAP, for Gossip-based Generic Aggregation Protocol. We prove these invariants for the synchronous and the asynchronous version of the protocol. We evaluate G-GAP through simulation, focusing on the accuracy of the estimates produced, the tradeoff between estimation accuracy and protocol overhead, the relationship between accuracy and network size (i.e., scalability), and, the relationship between accuracy and failure rate (i.e., robustness). For the sake of comparison, we run the same simulation scenarios with a tree-based aggregation protocol (that has a similar overhead), which provides us with insight into the performance of tree-based vs. gossip-based monitoring.

The rest of the paper is organized as follows. Section II reviews related work. Section III presents our protocol, starting out with *Push-Synopses* [3], which is developed first into a synchronous robust protocol, then an asynchronous robust protocol, namely G-GAP. Section IV presents results from our experimental evaluation. Finally Section IV concludes the paper and presents future work.

II. RELATED WORK

A paper by Renesse et al. [8] is the first work known to us where a gossip protocol is used in the context of monitoring aggregates. Here, the authors present a hierarchically structured monitoring system called *Astrolabe*. Aggregation functions are computed using tree-based aggregation, while a gossip protocol disseminates partial aggregates among peers in the hierarchy. The choice of using gossiping is motivated by achieving robustness. In case a parent node fails, an elected child node can replace the parent instantly. In contrast to our research, [8] aims at providing periodic estimates of a (slowly) changing global aggregate, while the goal for our protocol is to give a continuous estimate of a global aggregate in near real-time and with high accuracy. Since [8] uses a hierarchical aggregation scheme, the authors do not address the problem of mass loss (see section III) in gossip protocols as our work does.

Jelasity et al. in [2] present a gossip protocol for computing global aggregates. The protocol can be used for distributed polling or computing a snapshot of the system state. In order to support continuous monitoring of aggregates, the authors suggest restarting the protocol periodically, which amounts to periodic polling. While the protocol in [2] is not robust against node failures, our protocol is robust in the sense that it recovers lost mass for many types of node failures. (Since the protocol in [2] is restarted periodically, mass loss does not accumulate, and estimation errors are mitigated in this way.)

In a paper by Kempe et al. [3], the authors introduce a set of gossip protocols for computing a number of aggregation functions. The protocols presented do not directly support continuous monitoring of an aggregate, and they are not robust to node failures. Our work is based on one of these protocols, and, in this paper, we extend it to support continuous monitoring and asynchrony and robust operation.

Godsi et al. in [6] present a gossip algorithm for estimating the system size of a peer-to-peer system that uses DHTs and a ring topology. The algorithm computes the average inter-node distance on the ring to determine the system size. The protocol is robust to node failures. It periodically generates a wave that moves around the ring for the purpose of detecting mass loss and recovering from it. This work is similar to ours in that the protocol provides a continuous estimate and is robust to node failures. However, the specific recovery mechanism is different form ours, since it relies on the ring topology and is directly applicable only to estimating the system size.

Mehyar et al. in recent work which will be published in [20], present a distributed asynchronous algorithm for computing the average of local values on a network graph. The algorithm is robust to graceful topology changes and can be extended to support continuous estimation of averages. The authors prove convergence of their algorithm for a generalized asynchronous communication model that requires the eventual delivery of sent messages. By contrast, while our protocol makes stronger assumptions on the communication model, it has the added advantage of exponential convergence. Both protocols are robust to discontiguous crash failures and use a similar approach to achieve robustness. (We do not show the exponential convergence in this paper, but we prove the robustness properties.) Additionally, the protocols presented here can be adapted to a general range of aggregation functions in a straight-forward way while this is not clear for the protocol of [20].

III. THE PROTOCOL: G-GAP

A. Design goals and design approach

The management architecture. G-GAP is designed for a management architecture shown in Figure 1, in which each network device participates in protocol processing, by running

a management process, either internally or on an external, associated device. (A monitoring node in Figure 1 corresponds to a management process.) These management processes communicate via a network overlay for the purpose of monitoring a network-wide aggregate. We refer to this overlay also as the *network graph*. A node of this graph represents a network device together with its management process.

Each node of the network graph has an associated local (management) variable $x_i(t) \geq 0$. The local variable can represent a MIB variable, e.g., a device counter. In this paper, we assume that the variables are aggregated using AVERAGE. It is also possible to use other aggregation functions, such as SUM, MIN, or MAX with this protocol.

Design goals. Our aim is to develop a distributed protocol for continuously computing aggregation functions in a scalable and robust manner. The design goals for G-GAP are as follows:

- *Accuracy*: for a given protocol overhead, the estimation error should be small, and the variance of the estimation error across all nodes should be small.
- *Controllability*: it should be possible from a management station to control the tradeoff between protocol overhead and accuracy of the estimation.
- *Scalability*: for a fixed accuracy, the local protocol overhead at any node or link should in general increase sub-linearly with the system size.
- *Robustness*: the protocol should be robust to node failures and should allow for nodes dynamically joining and leaving the network. During transient periods, the estimation error due to reconfiguration should be small.

Design approach. G-GAP is based on "Push-Synopses", a gossip protocol for computing aggregates proposed by Kempe et al. [3]. Here we consider Push-Synopses applied only to the computation of averages, although we don't envisage any problems in adapting our results to more general synopses, such as those discussed in [3]. Our main contribution in this paper is to extend the Push-Synopses protocol with a scheme to provide accurate estimates in the event of node failures of different types. These extensions are introduced in two steps; first, for the case of fully synchronized rounds with guaranteed, timely message delivery; then, for the more general, asynchronous case.

B. Push-Synopses

In the Push-Synopses protocol given in Figure 2, each node *i* maintains, in addition to the local management variable x_i, a *weight* w_i and a *sum* s_i. The local estimate of the aggregate is computed as $a_i = s_i / w_i$. Following [3] the protocol is given for the case of a complete (i.e. fully connected) network graph of *n* nodes. However, the protocol is easily adapted to graphs where only adjacent nodes are allowed to communicate directly with each other. This is the relevant case in practice, for scalability reasons. In this case $\alpha_{i,j} = 0$ if $i \neq j$ and j is not adjacent to *i*.

```
Round 0 {
  1. s_i = x_i;
  2. w_i = 1;
  3. send (s_i, w_i) to self }
Round r+1 {
  1. Let {(s_l*, w_l*)} be all pairs sent to i
     during round r
  2. s_i = Σ_l s_l*;  w_i = Σ_l w_l*
  3. choose shares α_{i,j} ≥ 0 for all nodes j
     such that Σ_j α_{i,j} = 1
  4. for all j send (α_{i,j} * s_i, α_{i,j} * w_i) to each j }
```

FIGURE 2: PUSH-SYNOPSES – PSEUDO CODE FOR NODE *i*

The protocol executes in synchronized rounds, assuming reliable and timely communication, such that a message sent within a given round is guaranteed to be delivered within that round.

For the analysis of the Push-Synopses protocol we use $x_{r,i}$ to refer to the value of variable x_i at the end of round r. Protocol correctness relies crucially on the following invariant which expresses "mass conservation".

Proposition 1 (Mass Conservation, Push-Synopses [3]) *For all rounds* $r \geq 0$,

1. $\sum_i s_{r,i} = \sum_i x_i$
2. $\sum_i w_{r,i} = n$

Proof: Since the only communication between nodes is by message passing, it suffices to show that, if the property holds before the main protocol cycle is simultaneously executed at all nodes, then it holds after execution as well. This is straightforward. □

The Push-Synopses protocol tolerates message loss, provided that the underlying transport mechanism guarantees that this is reported. In the event of message loss, the protocol retransmits the lost message to itself. The mass conservation invariant holds, if message loss is always reported within the same round the message is sent; if this cannot be guaranteed, then the statement of prop. 1 must be adapted, by taking into account the "mass" of lost messages that were sent but have - so far - not been reported lost.

The protocol is given for the case of *polling*, i.e., where the variables x_i are constant. It is easily adapted to *continuous monitoring*, by sampling the value of x_i at each round and adding the change in x_i to s_i in step 2 of Figure 2. Step 2 for round *r* then must be replaced by

$$2. \quad s_i = \sum_l s_l^* + (x_{r,i} - x_{r-1,i}), \quad w_i = \sum_l w_l^*$$

The evaluation in section IV is done for a protocol modified in this way, since we consider continuous monitoring more relevant from an application perspective.

```
Round 0 {
  1. s_i = s1s_{i,i} = s2s_{i,i} = x_i;
  2. w_i = s1w_{i,i} = s2w_{i,i} = 1;
  3. for each node j {
        (rs_{i,j}, rw_{i,j}) = (0,0)
           //recovery share for node j
        (s1s_{i,j}, s1w_{i,j}) = (0,0)
           //share sent previous round to j
        (s2s_{i,j}, s2w_{i,j}) = (0,0) };
           //share sent round before last
  4. send (s_i, w_i, 0, 0) to self
  5. send (0,0,0,0) to all other nodes }
Round r+1 {
  1. let {(s*_l, w*_l, rs*_l, rw*_l) | l ∈ L} be all messages
     sent to i from sender l during round r
  2. s_i = Σ_{l∈L} s*_l ; w_i = Σ_{l∈L} w*_l ;
     for all l ∈ L let (rs_{i,l}, rw_{i,l}) = (rs*_l, rw*_l)
  3. for all k that did not send a message {
        s_i = s_i + s1s_{i,k} + s2s_{i,k} + rs_{i,k} ;
        s1s_{i,k} = s2s_{i,k} = rs_{i,k} = 0 ;
        w_i = w_i + s1w_{i,k} + s2w_{i,k} + rw_{i,k} ;
        s1w_{i,k} = s2w_{i,k} = rw_{i,k} = 0 }
  4. for all j choose shares α_{i,j} ≥ 0 such that
     Σ_j α_{i,j} = 1 and α_{i,j} = 0 whenever j ∉ L
  5. for all j {(s2s_{i,j}, s2w_{i,j}) = (s1s_{i,j}, s1w_{i,j}) ;
     (s1s_{i,j}, s1w_{i,j}) = (α_{i,j} s_i, α_{i,j} w_i)}
  6. for all j choose shares β_{i,j} ≥ 0 such that
     Σ_j β_{i,j} = 1 and β_{i,j} = 0 whenever j ∈ L̄ ∪ {i}
  7. for all j
     send (s1s_{i,j}, s1w_{i,j}, β_{i,j}(α_{i,i} s_i − x_i), β_{i,j}(α_{i,i} w_i − 1)) to j }
```

FIGURE 3: SYNCHRONOUS G-GAP – PSEUDO CODE FOR NODE *i*

C. Synchronous G-GAP

In case of a crash failure (i.e., where the local node state is lost) the Push-Synopses protocol of section III.B can no longer be guaranteed to converge to the true value, since the local variable of the failed node has been included in the computation, and as a consequence, the mass conservation invariant does not hold after the failure. To restore the invariant, the contribution of the local variable of the failed node to the total mass of the system needs to be removed.

In this subsection, we present a first adaptation of the Push-Synopses protocol to the case of crash failures, under rather strict assumptions. Later we show how these assumptions can be partially lifted, at the expense of a somewhat more complex protocol. The first adaptation, the synchronous G-GAP protocol, is shown in Figure 3.

The basic idea behind the restoration of the invariant is that a node *i* distributes recovery shares $rs_{j,i} = \beta_{i,j}(\alpha_{i,i} s_i - x_i)$ to each neighbor *j* and every node keeps track of its previously sent messages $s1s_{k,l}$ and $s2s_{k,l}$. This way, if node *j* discovers that *i* has failed, it uses $rs_{j,i}$, $s1s_{j,i}$ and $s2s_{j,i}$ to undo the contribution of node *i* to the computation of the aggregate. (This discussion relates to the *s* variables of the local state. It applies as well to the *w* variables. In the following, we proceed similarly and present arguments for the case of the *s* variables.) Note that when the protocol is executed on network graphs, only state information for adjacent nodes need be stored as explained in subsection A.

The protocol in Figure 3 relies on five rather strong assumptions:

1. *Reliable and timely message delivery*: There is a maximum communication delay $t_d < t_r$ (the round duration) such that a message sent from a node *i* to a node *j* at time *t* is delivered to *j* no later than $t + t_d$.
2. *Synchronized rounds*: Rounds are globally synchronized to within some bound $t_{\Delta r}$. That is, all live nodes start a round within $t_{\Delta r}$ of each other.
3. *Round atomicity*: All protocol cycles are executed as atomic statements. (Actually, it is sufficient that the send operation in step 7 is executed as an atomic operation.)
4. *Discontiguous crash failures*: No two nodes fail within two rounds of each other. When running this protocol on a network graph, this assumption translates to condition that adjacent nodes can not fail within a period of two rounds.
5. *Connectedness*: No failure will cause a node to become disconnected.

The assumption of a coarse round synchronicity allows us to unambiguously determine the value of a variable during each (global) round *r*. As a consequence, the value of x_i during *r* can be denoted by $x_{r,i}$, the value of $s1s_{i,j}$ by $s1s_{r,i,j}$, etc.

Round atomicity reduces, essentially, to atomic broadcast, since then the protocol cycles can be executed within a single transaction. This can be realized efficiently on some physical media, but the general implications of this assumption remain to be investigated. Round atomicity ensures that, during each round, if some message is received then all messages are received. With this, assumptions 1 and 2 guarantee that a round duration t_r can be found such that all messages are received during the same round they were sent.

Node failure, then, can be detected as in step 3 when the node fails to receive a message from a neighbor. Thus, if node *i* realizes in round *r* that it has not received a message from node *k* in the previous round, it concludes that node *k* did not receive the message ($s1s_k$, …) it sent to *k* in round $r-1$, neither did *k* process the message ($s2s_k$, …) sent in round $r-2$. Hence node *i* must in round *r* in this situation restore not only its recovery share for *k* but also the contributions it sent to *k* in the two previous rounds.

For the statement of the mass conservation property for SG-GAP we assume that round 0 by convention refers to the initialization phase, and that all nodes are alive during this round.

Proposition 2 (Mass Conservation, SG-GAP)

Let L_r *be the set of nodes that are alive during round* $r \geq 0$. *At the end of each round r*

1. $\sum_{i\in L_r} s_{r,i} + \sum_{i\in L_r, j\notin L_r} s2s_{r,i,j} + \sum_{i\in L_{r-1}-L_r, j\in L_r} rs_{r,j,i} = \sum_{i\in L_r} x_{r,i}$
2. $\sum_{i\in L_r} w_{r,i} + \sum_{i\in L_r, j\notin L_r} s2w_{r,i,j} + \sum_{i\in L_{r-1}-L_r, j\in L_r} rw_{r,j,i} = |L_r|$

Proof: See [19].

Invariant #1 can be explained as follows: the total mass $\sum_{i\in L_r} x_{r,i}$ at the end of round r is the sum of three components:

1. *Local mass*: the sum the local states $s_{r,i}$ of each live node *i*,
2. *Lost mass*: The sum of $s2s_{r,i,j}$ which were sent by currently live nodes *i* to currently dead nodes *j* in round $r-1$,
3. *Recovery mass*: The sum of $rs_{r,j,i}$ which were sent in round $r-1$ from a now dead node *i* to a currently live node *j*.

D. Asynchronous G-GAP

In this section we relax the synchrony assumptions of SG-GAP. The basic idea is to drop the method of determining the mass lost due to failures, which relies on the synchronous nature of the network, and, instead, letting a node compute recovery shares incrementally, by explicitly acknowledging the mass it receives from a peer. The pseudo-code for a protocol based on this idea, which we call Asynchronous G-GAP, is shown in Figure 4. From an application point of view, this asynchronous version of the protocol is the most significant protocol described here, and, therefore, we refer to it simply as G-GAP in the rest of this paper. As in the case of SG-GAP, the protocol is given for polling and for a complete network graph. The application of the protocol in the context of general network graphs is described in subsection III.B.

Compared to SG-GAP, the assumption of round synchronization is removed, as is the assumption of timely message delivery. With these modifications, nodes have less precise knowledge of each others' state. For this reason, in addition to the recovery mass already introduced in SG-GAP, a further pair $(acks_{i,j}, ackw_{i,j})$ is included in the messages sent in step 7.d of Figure 4, in order to let nodes acknowledge the receipt of messages. This pair $(acks_{i,j}, ackw_{i,j})$ is computed as what node *i* believes to be *j*'s recovery information on *i*, typically, the pair $(rs_{j,i}, rw_{j,i})$. However, lack of synchronization and message transmission delays may make these values different.

The asynchronous setting makes some changes in notation convenient. Most importantly, we consider system events to be serialized in a discrete time model. That is, failure events and protocol cycle executions (both of which we call *transitions*)

Round 0 {
1. $s_i = x_i$;
2. $w_i = 1$;
3. $L_i = \{i\}$;
4. for each node j $(rs_{i,j}, rw_{i,j}) = (0,0)$;
5. for each node j $(srs_{i,j}, srw_{i,j}) = (0,0)$;
6. send $(s_i, w_i, 0,0,0,0)$ to self;
7. for all $j \neq i$ send $(0,0,0,0,0,0)$ to j }

Round r+1 {
1. Let *Rec* be all messages received by i during round r
2. $s_i = \sum_{m\in M} s(m)$; $w_i = \sum_{m\in M} w(m)$
3. for all j $(acks_{i,j}, ackw_{i,j}) = (0,0)$
4. $L_i = L_i \cup orig(Rec)$
5. for all $j \in N$ {
   a. $(rs_{i,j}, rw_{i,j}) = (rs_{i,j}, rw_{i,j}) + \sum_{m:orig(m)=j}((rs(m), rw(m) - acks(m), ackw(m)))$
   b. $(acks_{i,j}, ackw_{i,j}) = (srs_{i,j}, srw_{i,j}) + \sum_{m:orig(m)=j}(s(m), w(m))$
   }
6. if (detected_failure(j)) {
   a. $(s_i, w_i) = (s_i, w_i) + (rs_{i,j}, rw_{i,j})$
   b. $(rs_{i,j}, rw_{i,j}) = (srs_{i,j}, srw_{i,j}) = (0,0)$
   c. $L_i = L_i \setminus j$
   }
7. for all $j \in L_i$ {
   a. choose $\alpha_{i,j} \geq 0$ such that $\sum_j \alpha_{i,j} = 1$
   b. choose $\beta_{i,j} \geq 0$ such that $\sum_j \beta_{i,j} = 1$ and $\beta_{i,i} = 0$
   c. $(srs_{i,j}, srw_{i,j}) = \beta_{i,j}(\alpha_{i,i} s_i - x_i), \beta_{i,j}(\alpha_{i,i} w_i - 1)$
   d. send $(\alpha_{i,j} s_i, \alpha_{i,j} w_i, srs_{i,j}, srw_{i,j}, acks_{i,j}, ackw_{i,j})$ to j
   e. $(rs_{i,j}, rw_{i,j}) = (rs_{i,j} + \alpha_{i,j} s_i, rw_{i,j} + \alpha_{i,j} w_i)$
   }
}

FIGURE 4: (ASYNCHRONOUS) G-GAP – PSEUDO CODE FOR NODE *i*

are considered to be atomic, and the time axis to be discretized into points $t_0, t_1, \ldots$ such that, at each instant t_n, exactly one of two events occurs on some node *i*: either a protocol cycle is executed at node *i*, or node *i* fails.

(Note that this protocol does not consider that failed nodes can recover. We believe that an extension of the protocol for this case is possible and we will pursue this issue.)

In the absence of round synchrony, local variables need to be sampled in a slightly different way than in the case of SG-GAP. Here we apply the following convention: if, say, $s_{i,j}$ is a local variable at node *i*, then $s_{t,i,j}$ refers to the value of $s_{i,j}$ at time instant t^+ that is immediately upon completion of the corresponding event at time $t \in \{t_n \mid n \in \omega\}$.

Concerning the communication model, we assume reliable message transmission in the sense that a message generated (i.e., sent) by the execution of a protocol cycle at node *i* is regarded as "pending", until either the destination node *j* fails or the message is read by the execution of a protocol cycle at *j*. We can thus define the following sets:

- $M_{pending,t,i,j}$: The set of all messages from origin *i* to destination *j* which are pending at time t^+.
- $M_{read,t,i,j}$: The set of messages from origin *i* to destination *j* which are read during a transition on node *j* at time *t*. If no transition takes place on node *j* at time *t*, then $M_{read,t,i,j} = \varnothing$.
- $M_{write,t,i,j}$: The set of messages from origin *i* to destination *j* which are generated by node *i* through the execution of a protocol cycle at time *t*. Again, if no cycle is executed on node *i* at time *t*, then $M_{write,t,i,j} = \varnothing$.
- $M_{transit,t,i,j} = M_{pending,t,i,j} - M_{write,t,i,j}$: The set of messages that are in transit, i.e., pending but not generated at time *t*.

We obtain the following straightforward axiom reflecting this model:

Axiom 1 (Communication model) *For all* $n \in \omega$,

$$M_{pending,t_{n+1},i,j} = (M_{pending,t_n,i,j} \cup M_{write,t_{n+1},i,j}) - M_{read,t_{n+1},i,j}$$

Messages have the format $(s, w, srs, sws, acks, ackw)$ where all variables are real-valued. We use the notation

$$s_{pending,t,i,j} = \sum \left\{ s \mid \exists w, srs, \ldots : (s, w, srs, \ldots) \in M_{pending,t,i,j} \right\}$$

and, similarly, for other variables $w, srs, \ldots$, and indices *read*, *write* and *transit*.

For provably correct operation, the G-GAP protocol requires the following assumptions to be satisfied:

1. *Self messages*: A message a node sends to itself will be immediately available for reading. This assumption can be lifted, we conjecture, by, for example, storing the message content in a local variable.
2. *Correlated failure and message signaling*: Message generation / reading (as part of a execution event) and failure events occur in the same relative order at an origin and a destination node. For instance, if a node *i* sends a message *m* to node *j* at time *t*, and at some later time $t' > t$ node *i* fails, then node *j* can only detect the failure of *i* after *m* is read. The reason for this assumption is to avoid mass loss. A likely consequence of this in terms of implementation is that failure signals are realized as messages and are buffered along with (other) messages.
3. *Discontiguous failures*: If a failure occurs, then no other live node can fail until the failure event has been processed by all nodes. More precisely, if a failure event occurs at time t_{fail}, then, there is a time Δt_{detct} such that, all nodes alive at time t_{fail} have processed the failure event by $t_{fail} + \Delta t_{detect}$. In addition, no node failures can occur during $\left[t_{fail}, t_{fail} + \Delta t_{detect} \right]$. As discussed before, on a general network graph, this assumption needs to hold only locally, i.e., for each node and its immediate neighbors.

Observe that no assumptions are made on transmission delays, node clock synchronization, or relative clock speeds. The statement of mass conservation now needs to take into account both received and pending messages.

Proposition 3 (Mass conservation, G-GAP) *Let L be the set of all nodes and* L_{t_n} *the set of live nodes at time* t_n. *Then, at all times* $t_n > 0$*:*

1. $$\sum_{i \in L_{t_n}} x_i = \sum_{j \in L, i \in L_{t_n}} s_{pending,t_n,j,i} + \sum_{i \in L_{t_n}, j \notin L_{t_n}} rs_{t_n,i,j} + \sum_{i \in L_{t_n}, j \notin L_{t_n}} (rs_{pending,t_n,j,i} - acks_{pending,t_n,j,i})$$
2. $$\left| L_{t_n} \right| = \sum_{j \in L, i \in L_{t_n}} w_{pending,t_n,j,i} + \sum_{i \in L_{t_n}, j \notin L_{t_n}} rw_{t_n,i,j} + \sum_{i \in L_{t_n}, j \notin L_{t_n}} (rw_{pending,t_n,j,i} - ackw_{pending,t_n,j,i})$$

Proof: See [19].

Prop. 6 expresses that the total mass of the system (i.e., the sum of local variables at all live nodes $\sum_{i \in L_{t_n}} x_i$) can be computed as the sum of the pending mass to all live nodes, plus the sum of the recovery shares for the failed nodes at the live nodes, plus the sum of the pending recovery shares, minus the sum of the pending acknowledgements.

IV. EXPERIMENTAL EVALUATION

We have evaluated G-GAP (called asynchronous G-GAP in section III.D) through extensive simulations using the SIMPSON simulator, a discrete event simulator that allows us to simulate packet exchanges over large network topologies and packet processing on the network nodes [5]. In various scenarios, we measure the estimation error by G-GAP on the network nodes, in function of the round rates, the network size, and the failure rate, in order to evaluate the protocol against our design objectives.

In addition to G-GAP, we run most simulation scenarios also with GAP [4], a tree-based aggregation protocol that gives an estimate of the aggregate at the root node. This allows us to compare the use of a gossip protocol with a protocol that is based on spanning trees for the purpose of monitoring network-wide aggregates. To make the comparison fair, we measure the performance metrics of both protocols for a comparable overhead. An overview of GAP is presented in [19].

A. Simulation setup and evaluation scenarios

Evaluation metrics. The main evaluation metric is the *estimation error* of the protocols. For G-GAP, we compute the estimation error as the (absolute) difference between the actual aggregate and the estimate of the aggregate on the nodes. For each simulation run, we determine the average estimation error over the simulation time and over all nodes. In addition, to indicate the dispersion of the error values, we determine the 90th percentile of the error values. In the case of GAP, all

measurements relate to the root node, since the estimate of the aggregate is available only at this node. A second evaluation metric is the *mass loss*, which measures the correctness of the protocol in the case of failures.

Local variables. For all simulation runs, a local variable represents the number of HTTP flows that enter the network at a specific router, and the aggregate represents the current average number of these flows in the network. We simulate the behavior of the local variables based on packet traces captured at the University of Twente [11]. Specifically, we use two traces

The first trace, which we call the University of Twente (UT) trace, is obtained as follows. Packet traces captured at two measurement points were divided into 150sec segments. From each segment *i*, we sample every second the number of HTTP flows that were traversing the measurement point. This number gives the value of the local variable $x_{t,i}$ of node *i* at time *t*. Across all segments, the average value of $x_{t,i}$ is about 45 flows, and the standard deviation of the change between two consecutive values is about 3.4. The second trace, which we call Randomized Periodic UT trace, is obtained by scaling the UT trace with a random periodic factor as follows.

$$x^*_{t,i} = \text{int}\left(\left(1+rand(0..1)\cos\left(2\pi t/30\right)\right)x_{t,i}\right) \text{ where}$$

rand(0..1) returns a random number uniformly distributed between 0 and 1, and *int()* the integer part of the argument.

The average value of $x_{t,i}^*$ across all segments is about 47 and the standard deviation of the change between two consecutive values is about 14. The second trace provides us with local variables that have higher dynamics than the first trace.

Overlay topology. The overlay topologies used for our simulations are generated by GoCast [17], a gossip protocol that builds topologies with bidirectional edges and small diameters. The protocol allows setting the (target) connectivity of the overlay. For this evaluation of G-GAP, we do not simulate the dynamics of GoCast. This means that the overlay topology does not change during a simulation run. Unless stated otherwise, the overlay topology used in the simulations has 654 nodes (this is the size of Abovenet, an ISP). It is generated with target connectivity of 10, which produces an average distance of 3.1hops and a diameter of 4hops in the overlay.

Failures. We assume a failure detection service in the system that allows a node to detect the failure of a neighbor. For our simulations, we assume that the failure of a node is detected within 1sec.

Other Simulation Parameters. In addition to the above, we run the simulations with the following parameters unless stated otherwise.

- For G-GAP, the default round length is 250ms, which means 4 rounds/sec. For GAP, the maximum message rate is 4 msg/sec per overlay link.
- For all nodes *i* and time *t*, $\alpha_{t,i,j}=1/(1+\#\text{ of neighbors})$ and $\beta_{t,i,j}=1/\#\text{ of neighbors}$
- Processing overhead: 1ms/cycle
- Network delay across overlay links: 20ms
- The length of a simulation run is 50sec, with a warm-up period of 25sec and a measurement period of 25 sec.

B. Estimation Accuracy vs. Protocol Overhead

In this experiment, we measure the estimation accuracy of G-GAP and GAP in function of the protocol overhead. We run simulations for round rates of 1, 2, 4, 6, 8 and 10 messages per sec. (For G-GAP, a round rate of 1 per sec means that the protocol executes one round per second, i.e., one protocol cycle per second. For GAP a round rate of 1 per sec means a maximum of 1msg/sec on an overlay link). We run two scenarios for the evaluation of estimation accuracy.

For the first scenario, we use the UT trace to simulate behavior of the local variables. Figure 5 shows the results. Each measurement point corresponds to one simulation run. The top of the bar indicates the 90th percentile of the estimation error.

As expected, for both protocols, increasing the round rate results in the decreasing of the estimation error. Therefore, the round rate controls the tradeoff between estimation accuracy and protocol overhead. In addition, for comparable overhead (i.e. the same round rates), the average error in G-GAP is around 8 times that of GAP.

In the second scenario, we study the influence of higher dynamicity of the local variables by using the Randomized Periodic UT trace to simulate behavior of the local variables. Figure 6 shows the results. The top of the bars indicate the 90th percentile of the estimation error.

The result shows that the average estimation error in both protocols is larger than that in the UT trace (2 times larger for G-GAP and 2-10 times for GAP). We explain this by fact that changes in the values of the local variables tend to be the larger for this trace than for the UT trace. We observe that the estimation error for GAP is smaller than the error for G-GAP: namely, by a ratio of 1.5 for low round rates and by a ratio of 5 for high round rates. This ratio is smaller than that for the UT trace.

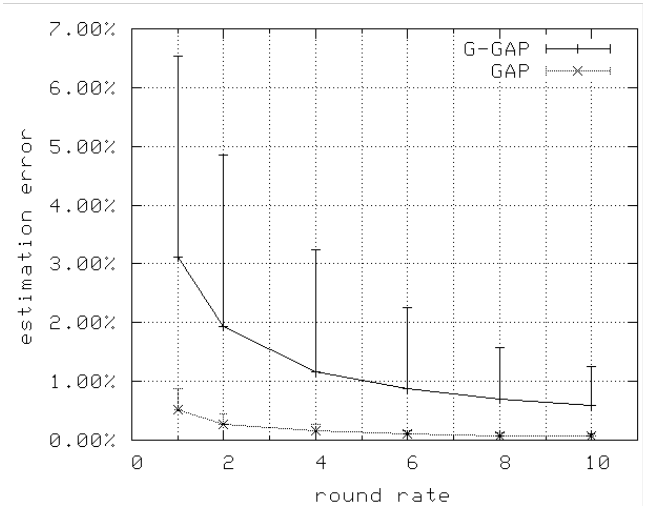

FIGURE 5: ESTIMATION ERROR VS. PROTOCOL OVERHEAD FOR UT TRACE

More importantly, within the parameter ranges explored, we conclude that GAP outperforms G-GAP in terms of accuracy.

C. Scalability

In this scenario, we measure the estimation accuracy of G-GAP and GAP in function of the network size. The round rate is set to 4 round/sec. We run simulations with GoCast-generated overlays for networks of size 82,164,327,654, 1308, 2626 and 5232 nodes. The target connectivity of GoCast is 10, which results in about 80% of the nodes having a connectivity of 10 and the rest a connectivity of 11. We use the UT trace to simulate the behavior of the local variables. Topological properties of the overlays are presented in Table 1.

TABLE 1: TOPOLOGICAL PROPERTIES OF GOCAST-GENERATED OVERLAYS USED IN SIMULATIONS

# nodes	diameter	avg. distance
82	3 hops	2.1 hops
164	4	2.4
327	4	2.7
654	4	3.1
1308	5	3.4
2616	5	3.7
5232	6	4

Figure 7 shows the results. Each measurement point corresponds to one simulation run. The top of the bar indicates the 90th percentile of the estimation error.

We observe that for both protocols, the estimation error seems to be independent on the network size. In the general case, for synthetic traces generated by the same (random) process, we would expect such a result for both GAP and G-GAP. Further, [2] shows for a polling-based gossip protocol, that variance of the estimates of the global average across all nodes is independent of the network size. Therefore, this simulation result is not entirely surprising.

Also, in this scenario, GAP clearly outperforms G-GAP in terms of accuracy.

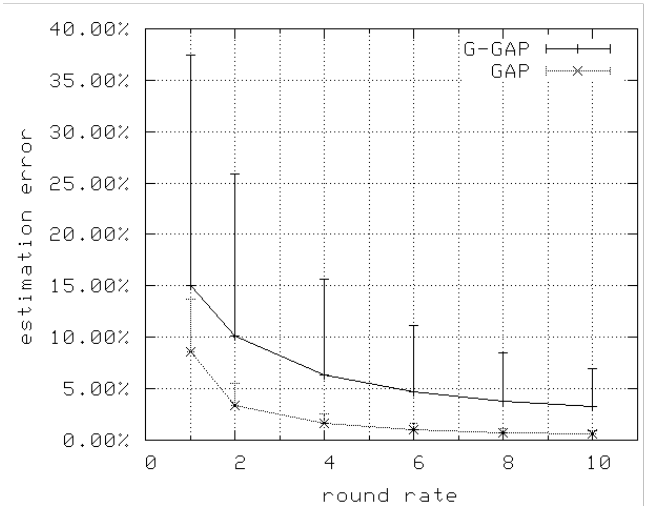

FIGURE 6: ESTIMATION ERROR VS. PROTOCOL OVERHEAD FOR RANDOMIZED PERIODIC UT TRACE

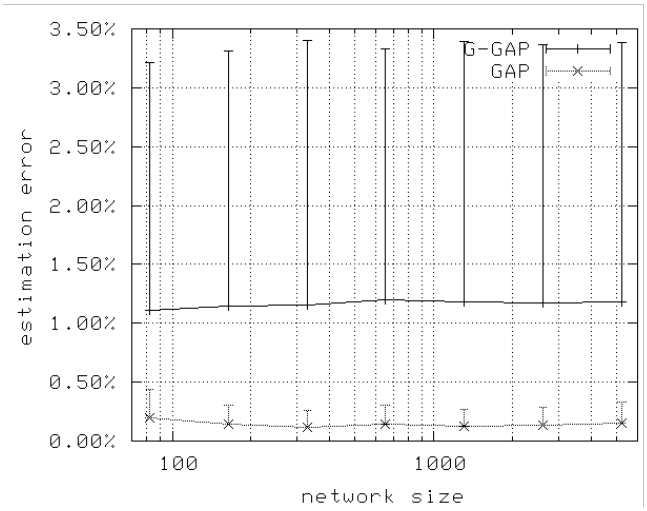

FIGURE 7: ESTIMATION ERROR VS. NETWORK SIZE

D. Robustness against node failures

In this section, we evaluate the robustness properties of G-GAP in two scenarios. In the first scenario, we validate the mass conservation property of G-GAP for the case of discontiguous failures (for which we proved the protocol to be robust), and we compare the result to a gossip protocol for computing aggregates that is not robust. Then, we compare the estimation accuracy of G-GAP and GAP by measuring the estimation error in function of the failure rate.

For the first scenario, we use the default topology (654 nodes) and simulation settings as described in section IV.A, and simulate the local weight changes using the UT trace. We generate failures as follows. Every 1.25sec, a node is selected at random. The node fails and recovers after 10sec. (Note that the generated failures are discontiguous, and therefore the protocol is robust by design.)

We run the scenario with G-GAP and with G-GAP without failure recovery, which we call here G-GAP--. (In our simulation runs, G-GAP-- was realized by executing G-GAP

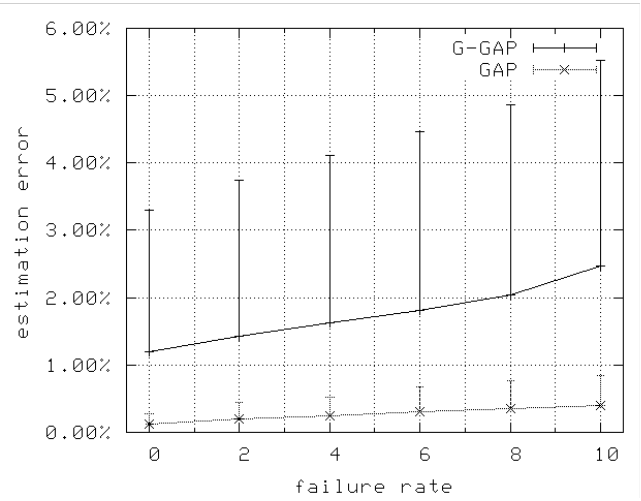

FIGURE 9: ESTIMATION ERROR VS. FAILURE RATE FOR GAP AND G-GAP

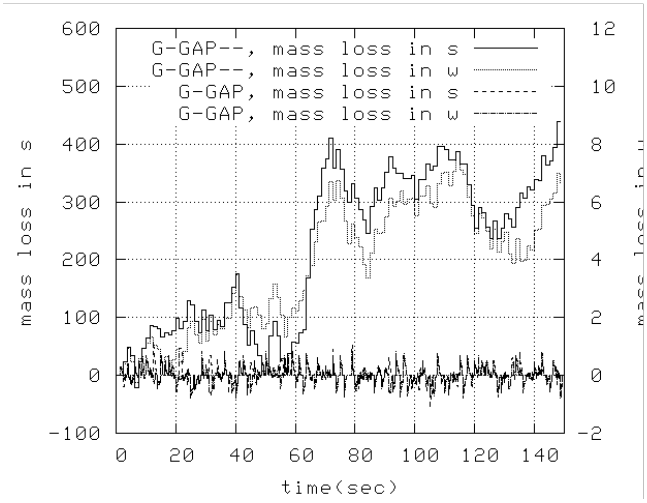

FIGURE 8: MASS LOSS IN G-GAP-- (TOP) AND G-GAP (BOTTOM) FOR DISCONTIGUOUS FAILURES

and skipping *nodefailed* calls.) During the simulation run, we measure the mass loss, computed as $\sum_{i\in L_t} x_i - \sum_{i\in L_t} s_{t,i} - \sum_{i\notin L_{\le t}} \sum_j \sum s_{transit}(t,i,j)$ in proposition 3.1 for the case of *s*, and, similarly, for the case of *w*. The simulation is run for 150sec and Figure 8 shows the result.

As can be seen from the figure, G-GAP corrects the effects of node failures, and thus mass loss is transient. For the case of G-GAP-- however, we observe that mass loss is not corrected but accumulates over time. Note that mass loss can be positive or negative. This scenario experimentally validates the robustness property of our G-GAP implementation, which says that, as long as failures are discontiguous, the protocol recovers lost mass and executes correctly.

For the second scenario, we use the same simulation parameters as above but vary the failure rate from 0 to 10 node failure/sec. Failure arrivals are generated by a Poisson process, and failures are uniformly distributed over all running nodes. A node that failed recovers after 10sec and reappears in the place it had in the overlay before the failure. Note that there is a chance that contiguous failures can occur and that the chance of contiguous failures increases with growing failure rate.

We use the UT trace to simulate the behavior of the local variables. Each simulation run of the scenario is 150sec (which includes a 25sec warm-up period).

Figure 9 shows the result obtained. Each measurement point corresponds to one simulation run. The top of the bars indicate the 90th percentile of the estimation error.

As can be seen from the figure, the estimation error for both GAP and G-GAP increases with the failure rate. We also see that the slope is steeper and the spread is wider for G-GAP than for GAP. This result is somewhat surprising for us. We would have expected a gossip protocol to perform better, compared to a tree-based protocol, under high node failure rates.

V. DISCUSSION AND FUTURE WORK

This paper makes two major contributions. First, we present a gossip protocol, G-GAP, which enables continuous monitoring of network-wide aggregates. The hard part is making the protocol robust against node failures, and we solved the problem for failures that are not contiguous (i.e., neighbors do not fail in short time of each other). We formally prove the correctness of the protocol by showing that invariants for mass conservation hold. Applying gossip protocols to continuous monitoring is not possible without solving the problem of mass loss due to node failures, and this paper gives a significant result in this direction. As we learned, Mehyar et al. independently produced a similar result recently, which will be published in [20].

The simulation results suggest that we have achieved the design goals for G-GAP set out in III.A. First, we have shown that the tradeoff between estimation accuracy and the protocol overhead can be controlled by varying the round rate. Second, with the traces we used, an estimation error of some 5% or less can be achieved for all network sizes and failure scenarios we simulated. We have observed that the estimation accuracy of the protocol, for a given overhead, does not seem depend on the network size, which makes the protocol scalable. Finally, we have proven and validated that the protocol is robust to discontiguous failures.

The second contribution of this paper is a comparative assessment of G-GAP with GAP, a fairly standard tree-based aggregation protocol. The significance of this assessment is a comparison between gossip based and tree based monitoring. Our simulation results show that within the parameter ranges of the simulation scenarios, the tree-based protocol consistently outperforms the gossip-based protocol. For comparable overhead, the tree-based protocol shows a smaller average estimation error and a smaller variance of the error than the gossip-based protocol, independent of network size and independent of frequency of failures that occur in the network.

While more work in needed to evaluate the relative advantages and disadvantages of tree-based vs. gossip-based monitoring, this paper makes a significant contribution to the discussion towards a new paradigm for distributed real-time monitoring.

Our simulation results show that the dynamics of the local variables influences the estimation accuracy in G-GAP. Not surprisingly, local variables with high dynamics lead to a lower accuracy and vice versa.

Our experience shows that the choice of the overlay topology significantly affects the performance of G-GAP, e.g., the estimation accuracy of the protocol. Generally speaking, a lower diameter and a higher connectivity of the overlay topology lead to a better performance. On the other hand, increasing the connectivity increases the load on the management nodes for a given round rate. Taking all this into account, we chose an overlay protocol that produces a uniform connectivity and, for our scenarios, we found out that a connectivity of 10 is an appropriate choice for real-time monitoring purposes.

All simulation results given in this paper are for AVERAGE as the aggregation function. We expect the performance of G-GAP to be affected by the particular choice of the aggregation function. Specifically, in scenarios with contiguous failures, we expect the estimation error to be different, and we plan to investigate this issue further. For instance, in the case of SUM, we expect the estimation error to be larger, while we expect it to be smaller for MIN and MAX.

G-GAP, as presented in this paper, is robust against discontiguous node failures. Our simulations have shown that in the case of frequent contiguous failures where 20% of the nodes are down, mass loss and hence estimation errors can accumulate. Therefore, in a real system, the protocol would have to be restarted in such cases. We see the possibility of further improving the robustness of G-GAP and plan more work in this direction. Further, a future publication will include a proof of the exponential convergence of G-GAP.

Acknowledgements. This work has been supported by a grant from Cisco Systems, a personal grant from the Swedish Research Council, and the EC IST-EMANICS Network of Excellence (#26854).

VI. BIBLIOGRAPHY

[1] D. Ernst, A. Hamel, and T. Austin, "Cyclone: a broadcast-free dynamic instruction scheduler with selective replay," In Proc. of the 30th Annual international Symposium on Computer Architecture (ISCA'03), San Diego, California, June 7–14, 2003.

[2] M. Jelasity, A. Montresor and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," ACM Transactions on Computer Systems, vol. 23, Issue 3, pp. 219-252, August 2005.

[3] D. Kempe, A. Dobra and J. Gehrke, "Gossip-Based Computation of Aggregate Information," In Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03), Cambridge, MA, USA, October 11-14, 2003.

[4] M. Dam and R. Stadler, "A generic protocol for network state aggregation," In Proc. Radiovetenskap och Kommunikation (RVK'05), Linköping, Sweden, June 14-16, 2005.

[5] K. S. Lim and R. Stadler, "SIMPSON — a SIMple Pattern Simulator fOr Networks," http://www.s3.kth.se/lcn/software/simpson.htm, August 2006.

[6] A. Ghodsi, S. El-Ansary, S. Krishnamurthy, and S. Haridi, "A Self-stabilizing Network Size Estimation Gossip Algorithm for Peer-to-Peer Systems," SICS Technical Report T2005:16, 2005.

[7] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, "Randomized Gossip Algorithms," IEEE/ACM Transactions on Networking, vol. 14, issue SI, pp. 2508-2530, June 2006.

[8] R. van Renesse, K. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring," ACM Transactions on Computer Systems, vol 21, issue 2, pp.164-206, May 2003.

[9] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, A. Demers, "Decentralized Schemes for Size Estimation in Large and Dynamic Groups," In proc. of the 4th IEEE International Symposium on Network Computing and Applications (NCA'05), Cambridge, MA, USA, July 27-29, 2005.

[10] A. Demers, D. Green, C. Hauser, W. Irish, J. Larson, "Epidemic algorithms for replicated database maintenance," In proc. the 6th Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10 - 12, 1987

[11] University of Twente - Traffic Measurement Data Repository, http://m2c-a.cs.utwente.nl/repository/, August 2006.

[12] A. Deligiannakis, Y. Kotidis and N. Roussopoulos, "Hierarchical in-Network Data Aggregation with Quality Guarantees,", In proc. 9th International Conference on Extending Database Technology (EDBT'04), Heraklion – Crete, Greece, March 14-18, 2004.

[13] S. Madden and M. Franklin and J. Hellerstein and W. Hong, "TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks," Fifth Symposium on Operating Systems Design and Implementation (USENIX - OSDI '02), Boston, MA, USA, December 9-12, 2002.

[14] M. A. Sharaf, J. Beaver, A. Labrinidis and P. K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," The International Journal on Very Large Data Bases, vol. 13, issue 4, pp. 384-403, December 2004.

[15] K.S. Lim and R. Stadler, "Real-time Views of Network Traffic Using Decentralized Management," 9th IFIP/IEEE International Symposium on Integrated Network Management (IM'2005), Nice, France, May 16-19, 2005.

[16] F. Wuhib, A. Clemm, M. Dam and R. Stadler, "Decentralized Computation of Threshold Crossing Alerts," 16th IFIP/IEEE Distributed Systems Operations and Management (DSOM'05), Barcelona, Spain, October 24-26, 2005.

[17] C. Tang, and C. Ward, "GoCast: Gossip-Enhanced Overlay Multicast for Fast and Dependable Group Communication," In Proc. International Conference on Dependable Systems and Networks (DSN'05), Yokohama, Japan, June 28 - July 1,2005.

[18] A. G. Prieto and R.Stadler "Adaptive Distributed Monitoring with Accuracy Objectives," to appear In proc. ACM SIGCOMM workshop on Internet Network Management (INM'06), Pisa, Italy, September 11, 2006.

[19] F. Wuhib, M. Dam, R. Stadler, A. Clemm, "Robust Monitoring of Network-wide Aggregates through Gossiping (long version)," KTH Technical Report [TRITA-EE 2006:043], available at http://www.ee.kth.se/php/index.php?action=publications, September 2006

[20] M. Mehyar, D. Spanos, J. Pongsajapan, S. Low, R. Murray, "Asynchronous Distributed Averaging on Communication Networks," To appear in IEEE/ACM Transactions on Networking, August 2007.