Robust Monitoring of Network-wide Aggregates through Gossiping

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

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

March 20, 2008

***Abstract*— The paper makes two contributions in the context of robust and scalable monitoring. First, we present a gossip protocol, G-GAP, which enables continuous monitoring of network-wide aggregates. Aggregates are computed from local management variables using functions such as SUM, MAX, or AVERAGE. The hard part is making the protocol robust against node failures, and we offer a solution for the case of crash failures that are not contiguous (i.e., where neighbors do not fail within short time of each other). Regarding correctness of the protocol under failures, we prove protocol invariants (namely, mass conservation) and give convergence results. We evaluate the protocol through simulation using real traces. The simulation results suggest that the tradeoff between estimation accuracy and protocol overhead can be controlled, and a high estimation accuracy (below some 5% error in most of our measurements) can be achieved, even in large networks and under frequent node failures. The second contribution of the paper is 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.**

Keywords: gossip protocol, epidemic protocol, robust aggregation, decentralized monitoring

I. INTRODUCTION

The motivation for this research is to investigate the use of gossip protocols for decentralized real-time monitoring. Recent research in gossip protocols suggests 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][28]. Originally, they have been proposed for disseminating information in large dynamic environments [10], and, more recently, they have been applied to various tasks, including constructing robust overlays [1][17] and estimating the network size [9][6].

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 the average load across all network links or 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 instance, can make use of cross-device 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. The protocol aggregates this data, in a decentralized fashion, to estimate the SUM, MAX, AVERAGE, etc., of the device variables. A management station or an application server can access the management overlay 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 than outlined above, as long as that architecture includes monitoring nodes that execute the protocol, provides functions in the monitoring nodes to access local device variables, and maintains an overlay for monitoring nodes to communicate.)

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,

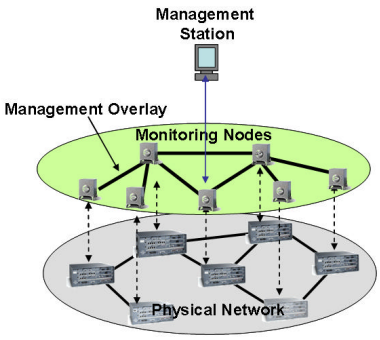

FIGURE 1: ARCHITECTURE OF THE DECENTRALIZED MONITORING SYSTEM. GOSSIP PROTOCOLS RUN IN THE MANAGEMENT OVERLAY (MIDDLE LAYER)

in tree-based aggregation, the result of an aggregation operation is available on the 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 loss (which is further explained in Section III), which causes systematic errors in aggregation operations. This problem of mass loss has not been sufficiently studied to date and thus needs to be addressed first when one wishes to perform a comparative assessment of tree-based and gossip-based aggregation.

The work in this paper is based on the *push-synopses* protocol of [3], an instance of a class of distributed agreement protocols that have been applied to a wide range of problems in distributed estimation and control (cf. [23]). Our main contribution is a significant extension of the push-synopses protocol that overcomes the mass loss problem and makes the protocol robust against node failures. We refer to this extended protocol as *G-GAP*, for *Gossip-based Generic Aggregation Protocol*. We establish a crucial mass conservation invariant for G-GAP and show that the logarithmic convergence results of [3] can be extended to G-GAP under node failures. 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 comparable overhead), which provides us with insight into the performance of tree-based vs. gossip-based monitoring.

This paper is an improved and extended version of [19]. The new additions include the proofs of the mass conservation invariant, a discussion and proofs of convergence properties of G-GAP, a completely revised and updated Section on related work, as well as additional simulation results that demonstrate the effect of mass loss on the accuracy of the protocol.

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 into an asynchronous robust protocol, namely G-GAP. Section IV presents results from our experimental evaluation. Finally Section V concludes the paper and presents future work.

II. RELATED WORK

Distributed averaging and agreement are special cases of the distributed aggregation problem. Many approaches to averaging and agreement, including several gossip algorithms, can be seen as instances of the following *iterative update scheme*, whereby a state vector $s(t) = (s_1(t), ..., s_n(t))$ is iteratively updated for each node i by the equation $s_i(t+1) = \sum_{j=1}^{n} a_{ij}(t) s_i(t)$ (cf. [23][24]). Here, $a_{ij}(t)$ are the elements of a non-negative, generally time-dependent matrix $A(t)$. In case $A(t)$ is time-invariant, [23] shows that, under mild assumptions on $A(t)$, any solution to the agreement problem based on the above scheme can be used for distributed averaging in time polynomial in network size. The push synopses protocol [3] can be seen as a generalization of the iterative update scheme to a range of aggregation functions besides average. Consensus propagation [25] uses a variant of the iterative update scheme for which strong relations to belief propagation [30] have been established.

General convergence properties of the iterative update scheme have been examined in [24]. Karp et al. [26] obtain a general lower bound for averaging using gossiping on an arbitrary graph. The (not very attractive) polynomial-time upper bound of [23] can be improved significantly for special cases of $A(t)$ and for specific graph topologies. In [3] a logarithmic upper bound is given for the push-synopses protocol (see Section III B.) on complete graphs and under assumption of uniform gossip. (Uniform gossip refers to a node selecting the recipient of a message uniformly at random [27].) Boyd et al. [7] use optimization for in-network configuration of a time-invariant transition matrix A to minimize the convergence time, and they obtain convergence bounds for several graph topologies.

A number of approaches have been explored in the literature to deal with node and link failures. Link failures, addressed for instance in [23], are generally easier to deal with, as nodes can preserve their state and hence avoid mass loss. An early example of a system addressing node failures is the *Astrolabe* monitoring system [8]. There, a tree-based scheme is used for aggregation, while a gossip protocol disseminates partial aggregates among peers in the tree hierarchy. Gossiping is used to achieve robustness, by allowing a preselected child node to instantly replace a parent node, in case the latter fails. In this case the mass loss problem does not arise, since the gossiping is not used for aggregation, but only for dissemination of information.

A straightforward approach to address the issue of mass loss is to restart the gossip protocol periodically [2]. While this does not prevent mass loss from occurring, it reduces the extent to which mass loss accumulates and thus mitigates estimation errors.

Mehyar et al. [20] have recently presented a solution for computing the average of local values on a dynamically changing network graph. Convergence is proved using the asynchronous framework of Bertsekas and Tsitsiklis [22]. No analytical upper bound on convergence time is reported. There are similarities in the use of recovery information between the protocol in [20] and the one developed independently by us. There are also significant differences between both protocols. In our case, the recovery mechanism is less deeply integrated into the underlying protocol, i.e., push-synopses. This allows us to establish convergence bounds by showing that there is a limit to the number of rounds needed to process recovery information before the protocol reverts to the behavior of the underlying protocol. Another difference is that our solution is designed to work correctly for a range of aggregation functions, whereas the protocol of [20] has been designed specifically for the computation of averages, and it is unclear to which extent it generalizes to other aggregation functions, such as SUM or other general synopses.

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 non-negative local (management) variable $x_i(t)$, or just x_i if the variable does not depend on time. 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.

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-synopsis*, a gossip protocol for computing aggregates proposed by Kempe et al.

```
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 a_{i,j} ≥ 0 for all nodes j
     such that Σ_j a_{i,j} = 1
  4. for all j send (a_{i,j} * s_i, a_{i,j} * w_i) to each j }
```

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

[3]. Our main contribution 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. While the aggregation function we focus on in this paper is AVERAGE, our results regarding protocol invariants and convergence are applicable to more general synopses, such as those discussed in [3].

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 a_i on node *i* 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 general network 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 $a_{i,j} = 0$ if $i \neq j$ and *j* is not adjacent to *i*.

The protocol executes in synchronized rounds. We assume reliable and timely communication in the sense 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 $s_{r,i}$ to refer to the value of variable s_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. ?

In [3] a logarithmic convergence result is given for the Push-Sum protocol, which applies to our instance of the push-

synopses protocol as well. We define the *relative error* at round *r* at node *i* as:

$$e(r,i) = \frac{1}{\sum_j x_j} \cdot \left| \frac{s_{r,i}}{w_{r,i}} - \frac{\sum_j x_j}{n} \right|$$

Then, we say that the protocol under consideration satisfies *log e -d convergence*, if for all $e \geq 0$, $d \geq 0$ the probability that there is a round $r' = O(\log n + \log 1/e + \log 1/d)$ for which $e(r,i) \leq e$ for all $r \geq r'$ is at least $1-d$.

Note that the above notion of log e -d convergence is formulated in the context of a synchronous model. It can be extended in a straightforward way to a discrete-time asynchronous model by considering time instants instead of rounds. This will be needed in Section III C.

The following result is due to Kempe et al [3]:

Theorem 2 (Convergence, Push-Synopses [3]) *On complete graphs and under the assumption of uniform gossip, the push-synopses protocol satisfies log e -d convergence.* ?

In the above theorem, *uniform gossip* refers to selecting the recipient of a message uniformly at random among all nodes of the graph (cf. [27]). Specifically, for all nodes *i*, $a_{i,i} = 0.5$ and there is exactly one node $j \neq i$ with $a_{i,j} = 0.5$. All other elements of $A(t)$ are 0.

The assumption of round synchronization can be lifted by adding round identifiers to messages and by buffering, so that a message with round number *r* is not overwritten by the arrival of a message with a later round number. We call the resulting protocol *asynchronous push-synopses*. Proposition 1, i.e., mass conservation, remains valid under these changes. To prove convergence, we assume that all messages sent during a round are also received during that round. The following result is then obtained as a straightforward adaptation of Theorem 2:

Corollary 3 (Convergence, Asynchronous Push-Synopses) *On complete graphs and under the assumption of uniform gossip, the asynchronous push-synopses satisfies log e -d convergence.* ?

The push-synopses (and asynchronous push-synopses) protocol is robust to message loss, provided that the underlying transport mechanism guarantees that this is reported. In the event of message loss, a node 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 Proposition 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 in Figure 2 for the case of *polling*, i.e., for the case 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 by adding the change in x_i to s_i in step 2 of Figure 2. Step 2 for round *r* must then be replaced by

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

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

C. Synchronous G-GAP

For crash failures (i.e., for failures 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, which we call *SG-GAP*, is shown in Figure 3.

The basic idea behind the restoration of the invariant is that a node *i* distributes recovery shares $rs_{j,i} = b_{i,j}(a_{i,i}s_i - x_i)$ 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. For reasons of brevity, we restrict the discussion here to the *s* variables of the local state. The same discussion applies to the *w* variables as well. Also, recall from Subsection III.B that, when the protocol is executed on general network graphs, a node needs to store state information for adjacent nodes, i.e., its neighbors, only.

The protocol in Figure 3 relies on five 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 general network graph, this assumption translates to the condition that adjacent nodes cannot fail within a period of two rounds.
5. *Connectedness*: No failure will cause a node to become disconnected.

```
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 j ∈ N − {i} {
        (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 L = {l ∈ N | l sent a message (s*_l, w*_l, rs*_l, rw*_l)
     to i 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 ∉ L {
        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 ∈ N choose shares a_{i,j} ≥ 0 such that
     Σ_j a_{i,j} = 1 and a_{i,j} = 0 whenever j ∉ L
  5. for all j ∈ N {(s2s_{i,j}, s2w_{i,j}) = (s1s_{i,j}, s1w_{i,j});
     (s1s_{i,j}, s1w_{i,j}) = (a_{i,j} s_i, a_{i,j} w_i)}
  6. for all j ∈ N choose shares b_{i,j} ≥ 0 such that
     Σ_j b_{i,j} = 1 and b_{i,i} = b_{i,j} = 0 whenever j ∉ L
  7. for all j ∈ N
     send (s1s_{i,j}, s1w_{i,j}, b_{i,j}(a_{i,i} s_i − x_i), b_{i,j}(a_{i,i} w_i − 1)) to j }
```

FIGURE 3: SYNCHRONOUS G-GAP – PSEUDO CODE FOR NODE *i*. N IS THE SET OF LIVE NODES AT ROUND 0.

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 s_i during *r* can be denoted by $s_{r,i}$, the value of $s1s_{i,j}$ by $s1s_{r,i,j}$, etc.

Round atomicity ensures that, during each round, if some message is sent from a node then all messages are sent. Round atomicity with reliable delivery is slightly weaker than atomic broadcast, as orderly delivery need not be guaranteed. On architectures supporting physical multicasting, round atomicity can be efficiently supported, though in general the assumption carries a heavy synchronization overhead [21]. Together, assumptions 1-3 imply 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 in step 3 in Figure 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 previous two 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 4 (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. $\sum_{i\in L_r} w_{r,i} + \sum_{i\in L_r, j\notin L_r} s2w_{r,i,j} + \sum_{i\in L_{r-1}-L_r, j\in L_r} rw_{r,j,i} = |L_r|$

For a detailed proof of Proposition 4, see [19]. That proof follows the same idea as the proof of Proposition 6, which is outlined below.

The invariant property #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*.

The convergence result below applies to graphs that are complete at some round r_s. Say that a node *i* is *stable* from r_s on, if *i* stays either live or failed throughout all rounds $r \geq r_s$. With this caveat Theorem 2 is easily extended to the SG-GAP protocol.

Theorem 5 (Convergence, SG-GAP) *Suppose the graph is complete at round r_s, and suppose all nodes are stable from round r_s onwards. Then, the SG-GAP protocol on a complete graph and under the assumption of uniform gossip satisfies log e -d convergence.*

Proof: The result follows since one round after r_s the value of the failure-handling variables $s1s_{i,j}$, $s2s_{i,j}$, $rs_{i,j}$, etc. will be 0, and so, by Proposition 4, the mass conservation invariant for push-synopses is reinstated. Since no more failures occur, the failure-handling variables will remain 0 and the behaviour of SG-GAP will consequently reduce to that of push-synopses, and the result follows by Theorem 2. ?

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. Message buffering and round numbering, as in Asynchronous push-synopses, is left implicit. 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*) 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. An extension of the protocol for this case is straightforward. The evaluation of the protocol is done by using G-GAP with such an extension.)

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, $rs_{i,j}$ is a local variable at node *i*, then $rs_{t,i,j}$ refers to the value of $rs_{i,j}$ at time instant t^+ that is immediately upon completion of the corresponding event at time $t \in \{t_n \mid n \in w\}$.

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 $M_{read,t,i,j} = \varnothing$.

round(0) {
1. $s_i = x_i$;
2. $w_i = 1$;
3. $L_i \leftarrow$ self;
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 M be all messages received by i during round r
2. $s_i = \sum_{m \in M} s(m) + (x_r - x_{r-1})$; $w_i = \sum_{m \in M} w(m)$
3. for all j $(acks_{i,j}, ackw_{i,j}) = (0,0)$
4. $L_i = L_i \cup orig(M)$
5. for all $j \in L_i$ {
   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. for all $j \in L_i$ 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. $L_i \leftarrow L_i \setminus j$
   }
7. for all $j \in L_i$ {
   a. choose $a_{i,j} \geq 0$ such that $\sum_j a_{i,j} = 1$
   b. choose $b_{i,j} \geq 0$ such that $\sum_j b_{i,j} = 1$ and $b_{i,i} = 0$
   c. compute $(srs_{i,j}, srw_{i,j}) = b_{i,j}(a_{i,i}s_i - x_i), b_{i,j}(a_{i,i}w_i - 1)$
   d. send $(a_{i,j}s_i, a_{i,j}w_i, srs_{i,j}, srw_{i,j}, acks_{i,j}, ackw_{i,j})$ to j
   e. $(rs_{i,j}, rw_{i,j}) = (rs_{i,j} + a_{i,j}s_i, rw_{i,j} + a_{i,j}w_i)$
   }
}

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

- $M_{write,t,i,j}$: 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* $n \in w$,

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

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

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

and, similarly, for other variables $w, rs, \ldots$, 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 storing the message content in a local variable or by adding sequence numbers to the protocol .
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 assumption is needed 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 6 (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 appendix A.

Proposition 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.

Proposition 6 allows us to derive a convergence result only under timely message delivery. That is, we assume that there is some time $t_d \le t_r$ such that, if a message is sent at time *t* and read at time t' then $t' - t \le t_d$. The timely delivery assumption has consequences in terms of the amount of asynchrony that can be tolerated. In particular, it must be possible to bound buffer sizes and hence also clock skew. This could be achieved in practice by periodically performing a rough synchronization of clocks to allow slower nodes to catch up with faster ones. We leave the problem of devising such a scheme to future work.

Under this assumption, if no more failures occur after some time *t*, then at time $t + t_d$ all failure signals will have been read by the receiving nodes, and Proposition 6 reduces to

$$\sum_{i \in L_{t_d}} x_i = \sum_{i, j \in L_{t_d}} s_{pending,t_d,i,j},$$

$$\left|L_{t_d}\right| = \sum_{i, j \in L_{t_d}} w_{pending,t_d,i,j}.$$

Moreover, since the failure-recovery variables *rs*, *srs*, *acks*, etc. can only affect the push-synopses variables *s* and *w* during processing of failure signals, the behavior of G-GAP after time $t + t_d$ reduces to that of Asynchronous push-synopses, at least up to assignments to the *s* and *w* variables. Say that a node *i* is *stable* from time t_s if *i* stays either live or failed at all times $t \ge t_s$, and let r_{t_s} be the round to which t_s belongs. We have thus shown:

Theorem 7 (Convergence, G-GAP) *Suppose the graph is complete at time* t_s*, and suppose all nodes are stable from time* t_s *onwards. Then, the G-GAP protocol on a complete graph and under assumption of uniform gossip satisfies log* e *-d convergence.*?

Recall that if G-GAP executes on a general network graph, the assumption of discontiguous failures relates to the neighborhood of a node, rather than the set of all network nodes. As a consequence G-GAP is robust to multiple concurrent failures as long as they occur in different neighborhoods.

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.

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

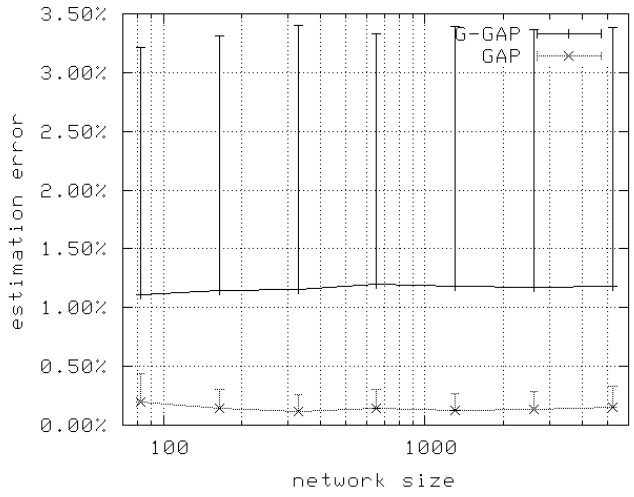

FIGURE 5: ESTIMATION ERROR VS. NETWORK SIZE

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.

$$x^*_{t,i} = \mathrm{int}\left(\left(1 + u\cos\left(2p\,t/30\right)\right)x_{t,i}\right)$$

where $u \in [0,1]$ is chosen uniformly at random, and $\mathrm{int}(y)$ the integer part of *y*.

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 6 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.

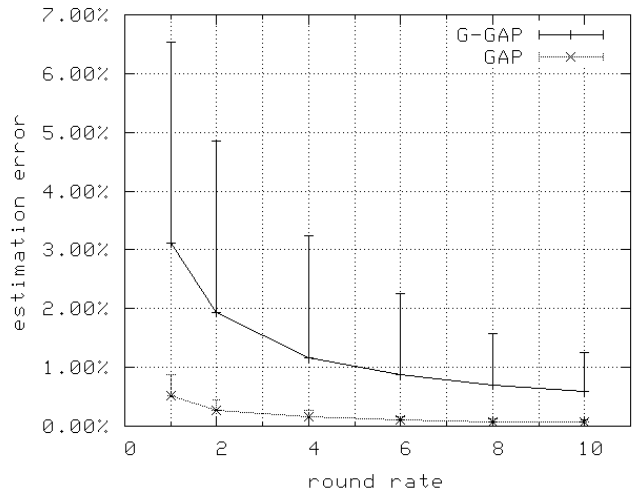

FIGURE 6: ESTIMATION ERROR VS. PROTOCOL OVERHEAD FOR UT TRACE

Figure 7 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.

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 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.

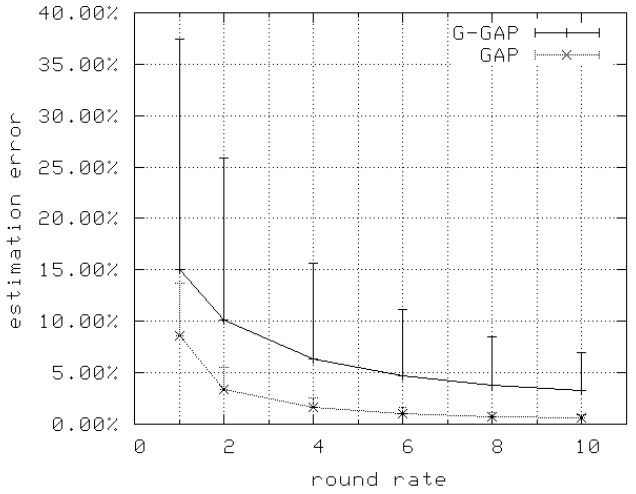

FIGURE 7: ESTIMATION ERROR VS. PROTOCOL OVERHEAD FOR RANDOMIZED PERIODIC UT TRACE

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.

D. Robustness against node failures

In this section, we evaluate the robustness properties of G-GAP in three 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. In the second scenario, we study the protocol accuracy under stochastic failures, where contiguous failures may occur. In the third scenario, we compare the estimation accuracy of G-GAP with that of GAP by measuring the estimation error in function of the failure rate for a comparable protocol overhead.

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 and skipping *detected_failure* calls. See Figure 4.) During the simulation run, we measure the mass loss, computed as $\sum_{i \in L_{t_n}} x_i - \sum_{j \in L, i \in L_{t_n}} s_{pending, t_n, j, i}$ for s and, similarly, for w. The simulation is run for 100sec and Figure 8 shows the result.

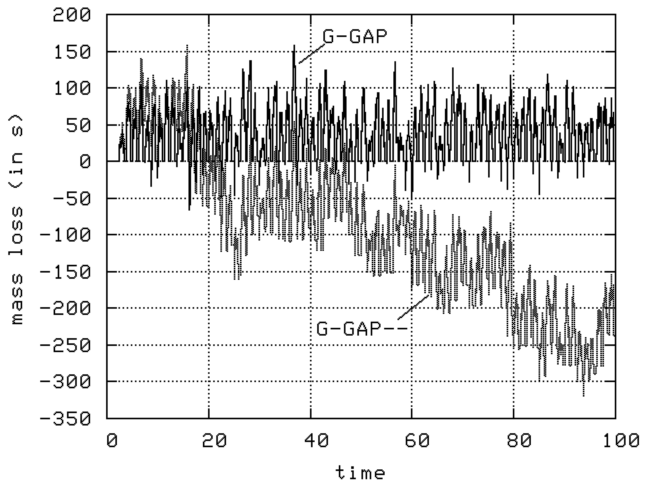

FIGURE 8: MASS LOSS IN G-GAP-- (BOTTOM) AND G-GAP (TOP) FOR DISCONTIGUOUS FAILURES IN A SIMULATION RUN

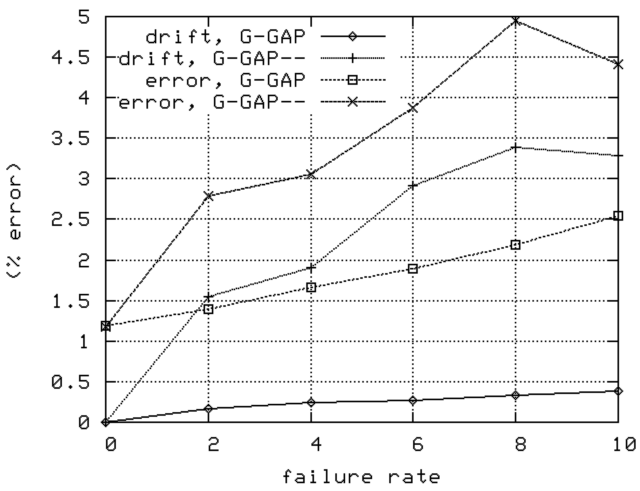

FIGURE 9: DRIFT AND OVERALL ESTIMATION ERROR VS. FAILURE RATE BY G-GAP AND G-GAP--

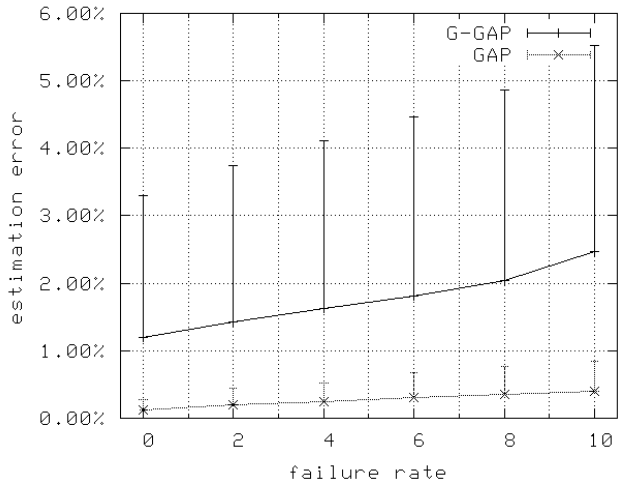

FIGURE 10: ESTIMATION ERROR VS. FAILURE RATE BY GAP AND G-GAP

As can be seen from the figure, G-GAP corrects the effects of node failures, and thus mass loss is transient (i.e., until the failure of the node is detected by all neighbors). For the case of G-GAP-- however, we observe that mass loss is not corrected and therefore 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 failures/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.

For both protocols (G-GAP and G-GAP--), we compute the drift in estimation, which is the estimation error due to mass loss, and the overall estimation error by the protocols. We obtain two curves per protocol, which are shown in Figure 9. Each measurement point on a curve corresponds to one simulation run. Each simulation run of the scenario is 150sec (which includes a 25sec warm-up period).

As expected, the overall estimation error by G-GAP-- is much larger than that by G-GAP in case of failures. This is because mass is lost at a faster rate by G-GAP-- than by G-GAP. The effect of the mass loss is directly visible in the two curves that show the drift. They show that the drift of G-GAP-- is larger than that of G-GAP and tends to increase with growing failure rate.

For the third scenario, we measure the estimation accuracy of G-GAP and that of GAP in function of failure rate, for a comparable overhead by both protocols. We use the same simulation parameters and produce failures in the same way as in the above scenario.

Figure 10 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 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 includes two main contributions. First, we present a gossip protocol, G-GAP, which enables continuous monitoring of network-wide aggregates. The hard part has been making the protocol robust against node failures, and we solved the problem for failures that are not contiguous (i.e., neighbors do not fail within short time of each other). Regarding correctness of the protocol, we provide results on protocol invariants (namely, mass conservation) and

convergence. Our robustness result complements a similar recent result by Mehyar et al. [20].

The simulation studies suggest that we have achieved the design goals for G-GAP set out in Section 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 real trace 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 to 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. A more recent study by us suggest that, in a resource-constrained environment characterized by high node mobility and large size, a gossip protocol potentially performs significantly better than a tree-based protocol [29]. While more work is 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.

We should like to understand the convergence properties of gossiping better. This is, however, outside the scope of the present paper. As we have shown, the convergence analysis for G-GAP reduce to that of push-synopses quite simply, as we can establish bounds after which, in stable state, the behavior of G-GAP and push-synopses is identical. This strongly suggests that improved understanding of convergence properties of the underlying failure-sensitive protocols can translate to corresponding bounds for their robust versions without too much effort.

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.

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

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,", Proc. Integrated Management (IM) 2007, pp. 226-235. Long version available as KTH Technical Report [TRITA-EE 2006:043], 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," IEEE/ACM Transactions on Networking, August 2007.

[21] F. Cristian, R. de Beijer, S. Mishra, "A performance comparison of asynchronous atomic broadcast protocols," Distrib. Syst. Engng. **1**, pp. 177-201, 1994.

[22] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, 1989.

[23] A. Olshevsky and J.N. Tsitsiklis, "Convergence rates in distributed consensus averaging," Proc. of the 45th IEEE Conference on Decision and Control (CDC'06), December 2006.

[24] J. N. Tsitsiklis, "Problems in Decentralized Decision Making and Computation," Ph.D. Dissertation, Dept. of Electrical Enginering and Computer Science, Mass. Institute of Technology, Cambridge, MA, 1984.

[25] C. C. Moallemi and B. Van Roy, "Consensus Propagation," IEEE Transactions on Information Theory, Vol. 52, No. 11, pp. 4753-4766, 2006.

[26] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, "Randomized rumor spreading," In Proc. of the 41st Annual Symposium on Foundations of Computer Science (FOCS'06), Washington DC, USA, November 12 - 14, 2000.

[27] D. Kempe, J. Kleinberg, "Protocols and Impossibility Results for Gossip-Based Communication Mechanisms," In Proc. The 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02), pp. 471, Ottawa, Ontario, Canada, November 17-22, 2002.

[28] K. Birman, "The promise, and limitations, of gossip protocols, "ACM SIGOPS Operating Systems Review archive, Volume 41 , Issue 5, October 2007.

[29] F. Wuhib and R. Stadler, "Adaptive Real-time Monitoring in Mobile Wireless Networks, KTH Technical Report TRITA-EE_2008:005, Jan 2008.

[30] Yedidia, J.S.; Freeman, W.T.; Weiss, Y., "Understanding Belief Propagation and Its Generalizations", *Exploring Artificial Intelligence in the New Millennium*, Chap. 8, pp. 239-269, Morgan Kaufman Publishers Inc., January 2003.

Appendix A: Proof of Proposition 6

Proposition 6 (Mass conservation, G-GAP)

Let L be the set of all nodes and L_t the set of live nodes at time t. At all times $t \geq 0$:

1. $$\sum_{i \in L_t} x_i = \sum_{j \in L, i \in L_t} s_{pending,t,j,i} + \sum_{i \in L_t,\, j \notin L_t} rs_{t,i,j} + \sum_{i \in L_t,\, j \notin L_t} (rs_{pending,t,j,i} - acks_{pending,t,j,i})$$

2. $$|L_t| = \sum_{j \in L, i \in L_t} w_{pending,t,j,i} + \sum_{i \in L_t,\, j \notin L_t} rw_{t,i,j} + \sum_{i \in L_t,\, j \notin L_t} (rw_{pending,t,j,i} - ackw_{pending,t,j,i})$$

Proof (sketch). We prove only 6.1 here. The proof of 6.2 is almost identical.

For a given (finite or infinite) run of the protocol, let $fail_1, fail_2, \ldots$ be the times of failure events during the run. We prove that 6.1 holds at all time instants up to and not including $fail_1$ (lemma 8), and that if 6.1 holds at time $fail_n - 1$ then it holds at all times $t \in [fail_n, fail_{n+1} - 1]$ (lemma 10). Note that we may assume $fail_1 > 0$. The result then follows by induction.

Lemma 8 (Base case) For all times $t \in [0, fail_1 - 1]$, $\sum_{i \in L_t} x_i = \sum_{j \in L, i \in L_t} s_{pending,t,j,i}$

Proof. If $t \in [0, fail_1 - 1]$ then $L = L_t$. For $t = 0$, by the protocol, $\sum_{j \in L, i \in L_0} s_{pending,0,j,i} = \sum_{i \in L} s_{pending,0,i,i} = \sum_{i \in L_0} (x_i)$. Assume the statement holds for $t < fail_1$ and we show it holds also for $t+1$. So assume that a cycle is executed on node a at time $t+1$. Then,

$$\sum_{j,i \in L} s_{pending,t+1,j,i}$$
$$= \sum_{j,i \in L} s_{pending,t,j,i} - \sum_{j \in L} s_{read,t+1,j,a} + \sum_{j \in L} s_{write,t+1,a,j}$$
(by axiom 1)
$$= \sum_{j,i \in L} s_{pending,t,j,i} - \sum_{j \in L} s_{read,t+1,j,a} + \sum_{j \in L} a_{a,j} s_{t+1,a}$$
(by the protocol)
$$= \sum_{j,i \in L} s_{pending,t,j,i} - \sum_{j \in L} s_{read,t+1,j,a} + s_{t+1,a}$$
(by the definition of a)
$$= \sum_{j,i \in L} s_{pending,t,j,i} = \sum_{i \in L} x_i$$
(by the protocol)

QED.

Note that Proposition 6 reduces to lemma 8 for time instants prior to $fail_1$.

Lemma 9 (Recovery information) For any node a and any time $t \geq 0$,

$$\sum_{j \in L_t} s_{pending,t,j,a} - x_a = \sum_{j \in L_t \setminus a} rs_{t,j,a} + \sum_{j \in L_t \setminus a} (rs_{pending,t,a,j} - acks_{pending,t,a,j}).$$

Proof. We first show that, for any two live nodes a and b at any time t where both nodes are alive,

$$b_{t,a,b}(a_{t,a,a} s_{t,a} - x_a) + s_{pending,t,b,a} = rs_{t,b,a} + (rs_{pending,t,a,b} - acks_{pending,t,a,b}) \quad (1)$$

The proof is a case analysis on which node a, b, or some $c \notin \{a, b\}$ performs a transition at the time and uses axiom 1 and the protocol. The details are left out. Then, (1) is summed over all live b to obtain the result. QED

Lemma 10 (Induction step) Assume that node $z \in L$ fails at time $fail_n$. If

$$\sum_{i \in L_t} x_i = \sum_{j \in L, i \in L_t} s_{pending,t,j,i} + \sum_{i \in L_t,\, j \notin L_t} rs_{t,i,j} + \sum_{i \in L_t,\, j \notin L_t} (rs_{pending,t,j,i} - acks_{pending,t,j,i}) \quad (2)$$

holds at time $t = fail_n - 1$, then (2) holds for all times $t: fail_n \leq t < fail_{n+1}$.

Proof. The proof is by induction on n. By discontiguous failures and correlated failure and message signaling, the failure at time $fail_{n-1}$, if it exists, will have been fully processed at time $fail_n$. If $fail_{n-1}$ does not exist we substitute 0 for it in the argument to follow. Thus, by (2) and the induction hypothesis, $\sum_{i \in L_{fail_{n-1}}} x_i = \sum_{j,i \in L_{fail_{n-1}}} s_{pending,fail_{n-1},j,i}$. We first show that (2) holds at time $fail_n$. The goal is to show

$$\sum_{i \in L_{fail_n}} x_i = \sum_{j \in L_{fail_{n-1}}, i \in L_{fail_n}} s_{pending,fail_n,j,i} + \sum_{i \in L_{fail_n}} rs_{fail_n,i,z} + \sum_{i \in L_{fail_n}} (rs_{pending,fail_n,z,i} - acks_{pending,fail_n,z,i})$$

By definition, $\sum_{i \in L_{fail_n}} x_i = \sum_{i \in L_{fail_{n-1}}} x_i - x_z$ and

$$\sum_{j \in L_{fail_{n-1}}, i \in L_{fail_n}} s_{pending,fail_n,j,i} = \sum_{j,i \in L_{fail_{n-1}}} s_{pending,fail_n-1,j,i} - \sum_{i \in L_{fail_{n-1}}} s_{pending,fail_n-1,i,z}$$

Therefore

$$\sum_{i \in L_{fail_n}} x_i$$
$$= \sum_{j \in L_{fail_{n-1}}, i \in L_{fail_n}} s_{pending,fail_n,j,i} + \sum_{i \in L_{fail_{n-1}}} s_{pending,fail_n-1,i,z} - x_z$$
$$= \sum_{j \in L_{fail_{n-1}}, i \in L_{fail_n}} s_{pending,fail_n,j,i} + \sum_{j \in L_{fail_n-1} \setminus z} rs_{fail_n-1,j,z}$$
$$+ \sum_{j \in L_{fail_n-1} \setminus z} (rs_{pending,fail_n-1,z,j} - acks_{pending,fail_n-1,z,j})$$
(by lemma 9)

$$= \sum_{j \in L_{fail_n - 1}, i \in L_{fail_n}} s_{pending, fail_n, j, i} + \sum_{j \in L_{fail_n}} rs_{fail_n, j, z} +$$

$$\sum_{j \in L_{fail_n}} (rs_{pending, fail_n, z, j} - acks_{pending, fail_n, z, j})$$

as desired. For the induction step assume that (*) holds at some time $t : fail_n \le t < fail_{n+1} - 1$. We show it also holds at time $t+1$. Let

$$M_t = \sum_{j \in L, i \in L_t} s_{pending, t, j, i} + \sum_{i \in L_t, j \notin L_t} rs_{t, i, j} +$$

$$\sum_{i \in L_t, j \notin L_t} (rs_{pending, t, j, i} - acks_{pending, t, j, i})$$

And assume that the round is executed on a at $t+1$. Then

$$M_{t+1} = \sum_{j \in L, i \in L_t} s_{pending, t, j, i} - \sum_{j \in L} s_{read, t+1, j, a} +$$

$$\sum_{j \in L_{fail_n - 1}} a_{a, j} s_{t+1, a} + \sum_{j \in L_t \setminus a} rs_{t, j, z} + rs_{t+1, a, z} +$$

$$\sum_{j \in L_t} (rs_{pending, t, z, j} - acks_{pending, t, j, i}) -$$

$$\sum_{j \in L_t} (rs_{read, t+1, z, j} - acks_{read, t+1, j, i})$$

(since only z's message is in the network)

$$= \sum_{j \in L, i \in L_t} s_{pending, t, j, i} - \sum_{j \in L} s_{read, t+1, j, a} +$$

$$\sum_{j \in L} s_{read, t+1, j, a} + \sum_{j \in L_t \setminus a} rs_{t, j, z} + rs_{t, a, z} +$$

$$\sum_{j \in L_t} (rs_{pending, t, z, j} - acks_{pending, t, j, i})$$

(since, if a learnt about the failure of z, then

$$\sum_{j \in L_{fail_n - 1}} a_{a, j} s_{t+1, a} = \sum_{j \in L} s_{read, t+1, j, a} +$$

$rs_{t, a, z} + rs_{read, t+1, z, a} - acks_{read, t+1, z, a}$ and $rs_{t+1, a, z} = 0$, and

if a did not learn about the failure of z then

$$\sum_{j \in L_{fail_n - 1}} a_{a, j} s_{t+1, a} = \sum_{j \in L} s_{read, t+1, j, a}$$ and

$rs_{t+1, a, z} = rs_{t, a, z} + rs_{read, t+1, z, a} - acks_{read, t+1, z, a}$)

$$= \sum_{j \in L, i \in L_t} s_{pending, t, j, i} + \sum_{j \in L_t} rs_{t, j, z} +$$

$$\sum_{j \in L_t} (rs_{pending, t, z, j} - acks_{pending, t, j, i})$$

$$= \sum_{i \in L_t} x_i$$

as was to be proved.