Robust Monitoring of Network-Wide Aggregates Through Gossiping

Fetahi Wuhib, Mads Dam, Rolf Stadler, and Alexander Clem

***Abstract*—We investigate the use of gossip protocols for continuous monitoring of network-wide aggregates under crash failures. Aggregates are computed from local management variables using functions such as SUM, MAX, or AVERAGE. For this type of aggregation, crash failures offer a particular challenge due to the problem of mass loss, namely, how to correctly account for contributions from nodes that have failed. In this paper we give a partial solution. We present G-GAP, a gossip protocol for continuous monitoring of aggregates, which is robust against failures that are discontiguous in the sense that neighboring nodes do not fail within a short period of each other. We give formal proofs of correctness and convergence, 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.**

***Index Terms*—Gossip protocol, epidemic protocol, aggregation, real-time monitoring.**

I. Introduction

THE goal of 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. To date, however, no gossip-based monitoring systems have been built, nor has gossip-based monitoring protocols been evaluated against alternatives, such as tree-based monitoring protocols.

Gossip protocols, also known as epidemic protocols, are round-based distributed algorithms. During a round, each node selects a subset of other nodes to interact, whereby the selection function is often probabilistic. Nodes interact via "small" messages, which are processed and trigger local state changes (cf. [9], [3], [28]). The authors of [9] have originally proposed gossip protocols for the purpose of disseminating updates in large database systems. More recently, these protocols have been developed for various other tasks, including constructing robust overlays (e.g., [1], [17]) and estimating the network size (e.g., [8]).

We are specifically interested in assessing the use of gossip protocols for decentralized aggregation of device data in near real-time, in support of network monitoring. 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 management tasks 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.

Currently, the most common approach to decentralized aggregation is based on creating and maintaining spanning trees in the management overlay. This approach has been investigated by others (cf. [11], [12], [13], [15]) and also by us (cf. [4], [14], [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 root node of the tree, while in gossip-based aggregation the result is available on all nodes. Third, failure handling is very different for both types of protocols. If a node fails, tree-based aggregation protocols are generally able to reconstruct the aggregation tree and re-compute the aggregate. In gossip protocols, the computation of the aggregate is unstructured, and a node failure generally causes irrecoverable information loss–a phenomenon we refer to as "mass loss" in this paper. 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 range of problems in distributed estimation and control (cf. [23]). Our main contribution in this paper is constructing an extension of the push-synopses protocol that overcomes the mass loss problem and renders the protocol robust against crash failures. We refer to this extended protocol as *G-GAP* for *Gossip-based Generic Aggregation Protocol*. We first introduce an intermediate version, which we call *Synchonous G-GAP* (or *SG-*

Manuscript received DATE; revised DATE; accepted DATE. The associate editor coordinating the review of this paper and approving it for publication was NAME.

F. Wuhib, M. Dam, and R. Stadler are with ACCESS Linnaeus Center, KTH Royal Institute of Technology, Stockholm, Sweden (e-mail: {fetahi, mfd, stadler}@kth.se).

A. Clemm is with Cisco Systems, San Jose, CA USA (e-mail: alex@cisco.com).

Digital Object Identifier 10.1109/TWC.2008.xxxxx.

GAP), for the case of synchronized rounds with guaranteed, timely message delivery; then, this solution is extended to the more general, asynchronous case. For both protocols we establish a crucial mass invariant and prove a key lemma allowing to reduce convergence properties of the extended protocols to convergence properties of the underlying push-synopses. 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 incurs a comparable overhead), which provides us with insight into the performance of tree-based vs. gossip-based monitoring.

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

This paper includes two contributions. First, we present a new gossip protocol for continuous monitoring of network-wide aggregates, which is robust against crash failures. We prove results on protocol invariants and convergence. Simulation studies suggest that the tradeoff between estimation accuracy and the protocol overhead can be effectively controlled and that a small estimation error (below 5%) can be achieved for network sizes (up to 10,000 nodes) and failures scenarios considered. The second contribution is a comparative assessment of our protocol vs. a tree-based aggregation protocol. Simulation results show that, within the parameter ranges of the simulation scenarios, the tree-based protocol consistently outperforms the gossip-based protocol.

The rest of the paper is organized as follows. Section II reviews related work. Sections III-VI present our protocol, starting out with push-synopses, which is developed first into a synchronous robust protocol, then into an asynchronous robust protocol, namely G-GAP. Section VII presents results from our experimental evaluation. Finally Section VIII concludes the paper and outlines future work.

II. Related Work

Distributed averaging, where the average of a set of local variables is computed, and distributed agreement, where nodes agree on a common value for a variable, are special cases of distributed aggregation. Many approaches to distributed aggregation, including the gossip protocols presented in this paper, can be seen as instances of the following *iterative update scheme*, where a state vector $s(t) = (s_1(t), \ldots, s_n(t))$ is iteratively updated for each node i by the equation

$$s_i(t+1) = \sum_{j=1}^{n} \alpha_{i,j}(t)\, s_i(t) \qquad (1)$$

(cf. [23], [24]). $t = 0, 1, 2, , \ldots$ represents time, and $\alpha_{i,j}(t) > 0$ are the elements of the, generally time-dependent, matrix $A(t)$.

In case $A(t) = A$ is time-invariant, [23] shows that, under mild assumptions on $A(t)$, any solution to the agreement problem can be used for distributed averaging, with a time complexity that is polynomial in network size. Consensus propagation in [25] uses a variant of the iterative update scheme for which strong relations to belief propagation (cf. [30]) have been established.

General convergence properties of the iterative update scheme have been examined in [24]. Karp et al. in [26] obtain a general lower bound for averaging using gossiping on an arbitrary graph. The polynomial-time upper bound of [23] can be lowered 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 on complete graphs and under assumption of uniform gossip (see Section IV). Boyd et al. in [6] minimize the convergence time for in-network computation of a time-invariant matrix A, and they obtain convergence bounds for several graph topologies.

Several approaches that address node and link failures in gossip protocols can be found in the recent literature. Link failures, investigated for instance in [23], are generally easier to deal with, since nodes can preserve their state and hence avoid mass loss. Gossip-based aggregation protocols that address node failures are proposed in the context of the Astrolabe monitoring system (cf. [7], [34]). There, a hierarchical scheme is used for the aggregation of the local values, while a gossip protocol disseminates partial aggregates among the peers on the same level of the aggregation tree. When a node fails, any peer on the same level can take its place in the aggregation tree. Note that mass loss does not occur in this scheme, as the gossip protocol is only used for data dissemination, but not for aggregation.

Jelasity et al. discuss in [2] two approaches that address mass loss in gossip-based aggregation protocols. The first approach is based upon restarting the protocol periodically, and the second involves running multiple instances of the protocol concurrently and computing an overall estimate of the aggregate from the estimates produced by the multiple instantiations. While these approaches do not prevent mass loss from occurring, they reduce the extent to which mass loss affects the estimation error.

Mehyar et al. present in [20] a gossip protocol for averaging local values on a dynamically changing network graph. Convergence of the protocol is proved using the asynchronous framework of Bertsekas and Tsitsiklis presented in [22]. No analytical upper bound on convergence time is reported in the paper. The protocol in [20] is robust to node failures, and there are similarities to the protocol presented in this paper (i.e., G-GAP), particularly regarding the use of recovery information. Our protocol G-GAP differs from the protocol in [20] in that the recovery mechanism can be easily separated from an underlying, non-robust 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.

Most works on gossip-based aggregation protocols consider the aggregation functions AVERAGE or SUM for protocol description and evaluation. (e.g., [2], [20], [3], [31]). (As explained in [2], using the aggregation function SUM, one can compute the average of local values; conversely, with the

aggregation function AVERAGE, one can compute the SUM of local values.) Using SUM, one can compute aggregation functions of the form

$$f\left(\sum_i g_1(x_i) + \sum_i g_2(x_i) + \ldots + \sum_i g_k(x_i)\right) \quad (2)$$

and thus also the aggregation function PRODUCT, namely as $e^{\sum_i \log(x_i)}$. In [2] the authors show how various types of means (e.g., geometric mean and harmonic mean) and moments (e.g., variance and skew) can be computed using AVERAGE. The minimum or maximum of values, i.e., the aggregation functions MIN or MAX, can in many cases be approximated by $\log\left(\sum_i e^{x_i}\right)$. In [3] the authors outline how a class of database queries that can be approximated by *linear synopses* (i.e, functions f on multisets with $f(S_1 \cup S_2) = f(S_1) + f(S_2)$) can be computed using SUM. Several gossip protocols have been proposed to compute aggregates other than the ones mentioned above. These include protocols for computing quantiles (cf. [2], [3]), the most frequent elements among a set of values (cf. [33]), the top-k elements (cf. [31]) and the top-k eigenvectors of a distributed adjacency matrix (cf. [32]).

The protocols presented in this paper, i.e., push-synopses and G-GAP compute AVERAGE. They can be modified in a straightforward way to compute functions of the form (2.2). The convergence results given in this paper can be extended to functions of the form (2.2), provided both f and g_i are continuous.

III. Architecture and Protocol Design Goals

The management architecture. The protocols discussed in this paper are designed for a management architecture shown in Fig. 1, in which each network device participates in the protocol execution, by running a management process, either internally or on an external, associated device. (A monitoring node in Fig. 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. A management station can access any node to initiate the monitoring protocol and retrieve estimates of the aggregate.

Each node of the network graph has an associated local (management) variable $x_i(t) > 0$. We write x_i if the variable does not depend on time. The variable can represent a MIB variable or a device counter (The term MIB stands for Management Information Base.) 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 an aggregation function over local variables in a scalable and robust manner. Estimates of the aggregate should be available on all network nodes. The design goals for the protocol, which we call G-GAP (for Gossip-based Generic Aggregation Protocol) 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.

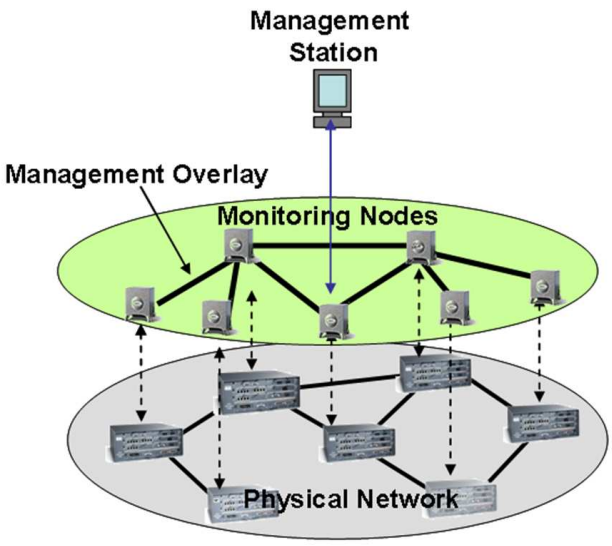

Fig. 1. Architecture of the decentralized monitoring system. Gossip protocols run in the management overlay (middle layer).

- *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 increase sub-linearly with the network size.
- *Robustness:* the protocol should be robust to node crash failures and should allow for nodes dynamically joining and leaving the network. During transient periods, the estimation error due to reconfiguration should be small.

IV. Push-Synopses

The Protocol. This section contains background information on the push-synopses protocol, which will be needed to understand design rationale, proofs, and evaluation of the extensions that form the contribution of this paper. The pseudo-code of push-synopses is given in Fig. 2 and follows the presentation in [3]. The protocol estimates the average of local state variables x_i of nodes i of a given network graph. The protocol implements two parallel passes of the iterative update scheme (1). One pass computes the sum s_i, while the other pass computes the weight w_i. At the end of each iteration (or round), shares of s_i and w_i are distributed to nodes according to the matrix elements $\alpha_{i,j}$, whereby $\alpha_{i,j}$ is chosen in such a way that $\alpha_{i,j} \geq 0$ and $\sum_j \alpha_{i,j} = 1$. (The elements $\alpha_{i,j}$ can be chosen differently from round to round, in which case matrix $A(t)$ is time-dependent.) After each round, the local estimate α_i of the global average at node i can be obtained as $\alpha_i = s_i/w_i$.

The push-synopses protocol can be instantiated in different ways. Initializing s_i to x_i and w_i to 1, as shown in Fig. 2, causes s_i/w_i to converge towards the average $\frac{1}{n}\left(\sum_i x_i\right)$, where n is the number of nodes. Instead, initializing s_i to x_i for all nodes i and w_i to 0, except for a selected node j for which w_j is initialized to 1, results in s_i/w_i converging to the sum $\sum_i x_i$.

```
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_j*, w_j*)} 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 }
```

Fig. 2. Push-synopses - pseudo code for node i.

Uniform gossip. The main setting studied in [3] is uniform gossip on a complete, i.e., fully connected, network graph. *Uniform gossip* refers to the strategy of selecting, in each round, the recipient of a message uniformly at random among all nodes of the graph (cf. [27]). For the case of push-synopses, this means that for all nodes i: $\alpha_{i,i} = 0.5$, for exactly one $j \neq 1$ $\alpha_{i,j} = 0.5$, and for all other k with $k \neq i$ and $k \neq j$ $\alpha_{i,k} = 0$.

To formally state the convergence properties of the protocol, we define the *relative error* after round r on node i as

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

where $\alpha_{r,i}$ denotes the value α_i of at the end of round r. (In the following, we use the same notation for the variables $s_{r,i}$, $w_{r,i}$, $\alpha_{r,i,j}$, etc.) Note that the management variables x_i are for now assumed to be constant.

Without proof we give the result of Kempe et al from [3], in slightly simplified form, as follows:

Theorem 1 (Convergence, push-synopses)

Assuming uniform gossip and a complete network graph, for a fixed error $\varepsilon > 0$ and a probability δ, the probability that there is a round $r' = O(\log n)$ for which $e(r,i) \leq \varepsilon$ for all $r \geq r'$ is at least $1 - \delta$.

In other words: the number of rounds r that is sufficient for estimates α_i to have converged to within a given error margin ε with given probability δ grows logarithmic with the network size n. In the remainder of the paper, when we say that some protocol P is $\varepsilon - \delta$ convergent, we mean that Theorem 1 holds for P.

In the above theorem, rounds are assumed to be of fixed duration. The theorem 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 VII.

Theorem 1 rests crucially on the following "mass invariant," which states that, under the assumption of a static network graph, the sums of the variables $s_{r,i}$ and $w_{r,i}$ remain constant during the evolution of the system.

Proposition 1 (Mass Invariant, push-synopses)

For all rounds $r \geq 0$,

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

Proof by induction: 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 on all nodes, then it holds after execution as well. This is straightforward.

We follow the literature (see, e.g., [3]) and interpret the sums in Proposition 1 as the *mass* of the network. Proposition 1 thus states that the protocol preserves the mass in the absence of failures. If a node experiences a crash failure, local information in the form of the state variables s_{ri} and w_{ri} are lost, which we refer to as *mass loss*. The usual consequence of mass loss is that the mass invariant is violated and that the local estimates of the aggregate on all nodes converge to the same erroneous value.

Other communication strategies. Communication strategies other than uniform gossip and network topologies other than complete graphs have been considered for push-synopses. For instance, the protocol is easily adapted to general network graphs where only adjacent nodes communicate directly with each other. This case is relevant in practice, as for large networks the maximum node degree is expected to be small in relation to the network size. The network graph push-synopses executes on can be expressed through a condition on the matrix A in (1), namely, $\alpha_{i,j} = 0$ if $i \neq j$ and j is not adjacent to i on the graph. Note that if $\alpha_{i,j} = 0$ then no message needs to be sent to j in step 4 of Fig. 2.

Under the assumption of an arbitrary, connected network graph and a time-invariant matrix A with evenly distributed local shares $\alpha_{i,j}$ (i.e., $\alpha_{i,j} = \alpha_{i,j'}$ whenever both j and j' are adjacent to i), [23] gives an upper convergence bound of $O\left(n^3 \log(n/\varepsilon)\right)$ and a lower bound of $\Omega\left(n^3 \log(1/\varepsilon)\right)$ for push-synopses. Note that the convergence bounds provided in [3] and [23] are different, as they relate to different graph topologies. The former applies to complete graphs only, whereas the latter covers connected graphs of arbitrary, e.g., linear, topology.

Variants. The push-synopses protocol executes in synchronized rounds and assumes reliable and timely communication in the sense that a message sent within a given round is guaranteed to be delivered within that round. The assumption of round synchronization can be lifted by adding round identifiers to messages and by buffering them in input queues, 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 remains valid under these changes. To prove convergence of the modified protocol, we assume that all messages sent during a round are also received during that round. Theorem 1 is easily adapted to this asynchronous setting and we obtain:

Corollary 1 (Convergence, Asynchronous push-synopses)

Asynchronous push-synopses is $\varepsilon - \delta$ convergent.

Push-synopses (and asynchronous push-synopses) can be made robust to message loss in a straightforward way, provided that the underlying transport mechanism detects the loss of a message. In such a case, the sender node retransmits the lost message to itself. The mass invariant holds, if message loss is always detected 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 messages that were sent but have–so far–not been reported lost.

So far, push-synopses has been discussed for the case of *polling*, i.e., for the case where the variables x_i remain constant. It is easily adapted to support *continuous monitoring*, by sampling the value of x_i at the beginning of each round and by adding the change in x_i to s_i in step 2 of Fig. 2. Step 2 for round r must then be replaced by

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

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

V. Synchronous G-GAP

In this section, we present a first adaptation of push-synopses to become robust against crash failures under rather strict assumptions. (In Section VI, we show how these assumptions can be partially lifted at the expense of a somewhat more complex protocol.) We call this adaptation the *Synchronous G-GAP* protocol or simply *SG-GAP*.

In case of a crash failure (i.e., a failure where the local node state is lost), the push-synopses protocol is no longer guaranteed to converge to the true value, since Proposition 1 does not hold anymore.

In order to be robust against node failures, the protocol design must address two issues, namely, how to detect a node failure and how to reinstate the mass invariant in Proposition 1 after such a failure. First, based on the assumptions for this protocol (see below), a node can detect in round r whether a neighbor has failed in round $r-2$. Second, to reinstate the mass invariant after a node failure, the following approach is taken. During each round, a node sends information to its neighbors about its state. In addition, the node keeps track of state information it has sent out during the last two rounds.

The SG-GAP protocol. Following the approach outlined above, SG-GAP extends push-synopses by maintaining additional variables that allow lost mass to be recovered by neighboring nodes in the event of crash failures. Besides the sum variables s_i and the weight variables w_i used in push-synopses, SG-GAP maintains the following state variables on each node i: the variables $rs_{i,j}$ and $rw_{i,j}$, which we call recovery shares and which contain information about the state of node j that is stored on node i, as well as the variables $s1s_{i,j}$, $s2s_{i,j}$, $s1w_{i,j}$ and $s2w_{i,j}$, which contain shares of the local state that have been sent from node i to node j during the last two rounds.

In the case of push-synopses, a node sends, during each round, a message of the form (s, w) to each of its neighbors. In SG-GAP, this message format is extended to include recovery shares, and a node thus sends a message of the form (s, w, rs, rw) to each of its neighbors during each round. Depending on the communication strategy used (see Section III), the set of neighbors of a node can include all nodes of the system (as in the case of uniform gossip) or only a small subset.

Fig. 3 shows the pseudo code for SG-GAP for a node i. The upper part (round 0) describes the initialization of the protocol, and the lower part describes the protocol cycle that is executed for each round $r > 0$. N denotes the set of (live) neighbors of node i at round 0.

```
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 > 0 {
  1. let L = {l ∈ N | l sent a message (s*_l, w*_l, rs*_l, rw*_l)
     to i during round r−1}.
  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 neighbors 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 α_{i,j} ≥ 0 such that
     Σ_j α_{i,j} = 1 and α_{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}) = (α_{i,j} s_i, α_{i,j} w_i) }
  6. for all j ∈ N choose shares β_{i,j} ≥ 0 such that
     Σ_j β_{i,j} = 1 and β_{i,i} = β_{i,j} = 0 whenever j ∉ L
  7. for all j ∈ N
     send (s1s_{i,j}, s1w_{i,j}, β_{i,j}(α_{i,j} s_i − x_i), β_{i,j}(α_{i,j} w_i − 1)) to j }
```

Fig. 3. Synchronous G-GAP - pseudo code for node i, N is the set of live neighbors of node i at round 0.

The initialization of s_i and w_i is identical to push-synopses, and all other local variables are initialized with value 0 (steps 1–3). As in push-synopses, node i sends the values s_i and w_i to itself (step 4). In addition, it sends a message with 0-values to its neighbors to indicate that it is alive (step 5).

At the beginning of each protocol cycle (round $r > 0$), a node processes the messages it has received during the last round and updates its sum and weight variables, as well as the recovery shares of its neighbors (steps 1-2).

Step 3 handles failure detection and the recovery from crash failures. The fact that node i failed to receive a message from a neighbor k in round $r-1$ (which is expressed as $k \notin L$) indicates that node k has failed during round $r-2$ (see protocol assumptions below). In this case, the local sum and weight variables are updated, to compensate for lost mass and to reinstate the mass invariant, which is achieved in the following way.

If node k failed during round $r-2$, the messages sent from node i to k during rounds $r-1$ and $r-2$ have both not been processed by k. The values of the sums and weights contained in these messages are available on node i as $s1s_k$, $s1w_k$, $s2s_k$

and $s2w_k$. These values, together with the recovery share for node k on node i, are added to the local sums and weight variables. (The fact that this step correctly reinstates the mass invariant will be proved in Proposition 2.)

In step 4, node i selects the shares $\alpha_{i,j}$, according to which the local sum and the weight variables will be distributed at the end of round r. In step 5, the variables $s2s_{i,j}$ and $s2w_{i,j}$ are set to the values sent to node j in round $r-1$, and the variables $s1s_{i,j}$ and $s1w_{i,j}$ are set to the values that will be sent out to node j at the end of round r.

In step 6, node i selects the shares $\beta_{i,j}$, according to which its state information will be distributed at the end of round r for the purpose of failure recovery. This information accounts for the shares of the sum and weight that a node sends to itself and the value of the local variable x_i.

Finally, in step 7, node i sends out a message to each of its neighbors, which concludes the protocol cycle for round r.

SG-GAP protocol assumptions. The protocol in Fig. 3 relies on five assumptions:

1) *Reliable and timely message delivery:* There is a maximum communication delay t_d, strictly smaller than the round duration t_r, 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 $t_{\Delta r} < t_r$. 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 operation. (Actually, it is sufficient that the send function in step 7 is executed as an atomic operation.)
4) *Discontiguous crash failures:* At most one node can fail within any period of two rounds. When running the protocol on a general network graph, this assumption is reduced 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 on the network graph.

The above assumption of coarse round synchronicity allows us to unambiguously determine the value of a variable during each (global) round r. As a result, the value of s_i during round 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 (cf. [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. Therefore, if node i detects in round r that it has not received a message from node k in round $r-1$, it can conclude that node k has failed in round $r-2$.

The assumption of discontiguous crash failures is needed, as otherwise recovery information exchanged between neighboring nodes may be lost, potentially invalidating the mass invariant.

Correctness and convergence. For the statement of the mass invariant, we assume that round 0 refers to the initialization phase and that all nodes are alive during this round.

Proposition 2 (Mass Invariant, 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) $\sum_{i\in L_r} s_{r,i} + \sum_{i\in L_r, j\notin L_r} s2s_{r,i,j} + \sum_{i\in L_r, j\in L_{r-1}-L_r} rs_{r,i,j} = \sum_{i\in L_r} x_{ri}$
2) $\sum_{i\in L_r} w_{r,i} + \sum_{i\in L_r, j\notin L_r} s2w_{r,i,j} + \sum_{i\in L_r, j\in L_{r-1}-L_r} rw_{r,i,j} = |L_r|$

A detailed proof of Proposition 2 is given in [19]. The proof follows the same idea as that of Proposition 3, which is given in the appendix. (Note that the first terms in both formulas include the contributions from the state variables $s1s_{i,j}$ or $s1w_{i,j}$, respectively.)

The invariant property 1 in Proposition 2 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 $rs_{r,i,j}$ which were sent by currently dead nodes j to a currently live nodes i in round $r-1$.

For the convergence analysis, we prove a key lemma relating executions of SG-GAP to executions of push-synopses for the case when the network is eventually stable. Formally, say that a node i is *stable* from some round r' onwards, if i stays either live or failed throughout all rounds $r \geq r'$. In order to extend convergence results for push-synopses, such as those of [3] and [23], to SG-GAP, we restrict attention to networks graphs for which all nodes are stable from some round r' onwards. We define an execution of the protocol as a sequence $\sigma = \sigma_1\sigma_2\ldots\sigma_r\ldots$ that records, for each round r, the assignment σ_r of values to local variables (such as s_i, $rs_{i,j}$, etc.) that applies at the end of round r. The *observable trace*, $tr(\sigma)$, of the execution σ is the restriction of σ so that each σ_r assigns values only to the variables s_i and w_i. Finally, the m-th suffix of the trace $\sigma = \sigma_0\sigma_1\ldots$ is the trace $\sigma^m = \sigma_m\sigma_{m+1}\ldots$.

The key observation is that, if a node is stable, then all recovery information is processed and reset within one round.

Lemma 1 Suppose the network graph is stable from round r' onwards. Let σ be an execution of SG-GAP. Then $tr(\sigma)^{r'+1}$ is a trace of push-synopses.

Proof: One round after r' the value of the failure handling variables $s1s_{i,j}$, $s2s_{i,j}$, $rs_{i,j}$, etc. will be 0 and remain so for all rounds $r \geq r'$. Since the graph remains stable after round r', $tr(\sigma)^{r'+1}$ can be extended to an execution of push-synopses.

As an example, we show how Lemma 1 allows to extend the convergence result of [3] to SG-GAP.

Theorem 2 (Convergence, SG-GAP)
SG-GAP is $\varepsilon-\delta$-convergent on graphs that are eventually stable.

Proof: By Lemma 1, $\varepsilon-\delta$-convergence for SG-GAP from round r' on follows from Theorem 1, given the assumptions.

We then use Proposition 2 to conclude that $\varepsilon - \delta$-convergence holds from the initial state as well.

VI. Asynchronous G-GAP

In this section, we present an adaptation of the SG-GAP protocol to an asynchronous setting. We call this protocol *Asynchronous G-GAP*, or refer to it simply as G-GAP. As in the description of push-synopses and SG-GAP, the protocol is given for polling. The application of the protocol for continuous monitoring is described in Section IV.

Compared to SG-GAP, the assumption of round synchronization is removed for this protocol, as is the assumption of timely message delivery. With these modifications, nodes have less precise knowledge of each others' state, which has consequences on how failures can be detected and how the mass invariant can be reinstated.

Regarding failure detection, G-GAP relies on an external failure detector, which signals to a node when one of its neighbors has crashed. With respect to reinstating the mass invariant, the protocol design follows a similar approach to that taken for SG-GAP, whereby a node maintains information about the state of its neighbors and about the messages it has sent. However, unlike in SG-GAP, acknowledgements are used to notify a sender about the receipt of a message.

The G-GAP protocol. As in the case of push-synopses and SG-GAP, the G-GAP protocol maintains the sum variables s_i and the weight variables w_i on a node i. G-GAP also maintains recovery shares $rs_{i,j}$ and $rw_{i,j}$, although with a slightly different semantic. Each node also maintains the variables $srs_{i,j}$ and $srw_{i,j}$, which contain the recovery shares sent by node i to j in the previous round. All nodes also maintain L_i, which is the set of neighbors of node i that the node believes to be alive.

For the G-GAP protocol, the message format used in SG-GAP is extended to include acknowledgements for received messages and has the new format $(s, w, rs, rw, acks, ackw)$.

Fig. 4 shows the pseudo code for G-GAP for a node i. The upper part (round 0) describes the initialization of the protocol, and the lower part describes the protocol cycle that is executed for each round $r > 0$. N denotes the set of (live) neighbors of node i at round 0.

The protocol starts by initializing s_i and w_i in a same way as in push-synopses (steps 1-2). The set L_i is initialized with the id of node i (step 3). The recovery shares $rs_{i,j}$ and $rw_{i,j}$, as well as the sent recovery shares $srs_{i,j}$ and $srw_{i,j}$ are initialized with value 0 for each neighbor j (steps 4–5). The initialization completes with the node sending s_i and w_i to itself and values of 0 to its neighbors (steps 6–7).

At the beginning of each protocol cycle (round $r > 0$), a node processes the messages it has received during the last round and updates its sum and weight variables, as well as acknowledgements are set to 0 (steps 1-3). For a message m of the format $(s, w, rs, rw, acks, ackw)$, we use $s(m)$, $w(m)$, etc. to extract components of m and use $orig(M)$ to denote the set of senders of messages $m \in M$. All nodes that sent a message to node i during the previous round are added to its set L of live neighbors (step 4).

In step 5.a, the recovery shares $rs_{i,j}$ and $rw_{i,j}$ held by node i for node j are incremented by the recovery shares

```
Round 0 {
  1. s_i = x_i ;
  2. w_i = 1 ;
  3. L_i = {self};
  4. for each node j∈N (rs_{i,j}, rw_{i,j}) = (0,0) ;
  5. for each node j∈N (srs_{i,j}, srw_{i,j}) = (0,0) ;
  6. send (s_i, w_i, 0,0,0,0) to self;
  7. for all j∈N send (0,0,0,0,0,0) to j }
Round r>0 {
  1. Let M be all messages received by
     i during round r−1
  2. s_i = Σ_{m∈M} s(m) ; w_i = Σ_{m∈M} w(m)
  3. for all neighbors j (acks_{i,j}, ackw_{i,j}) = (0,0)
  4. L_i = L_i ∪ orig(M)
  5. for all j∈L_i {
     a. (rs_{i,j}, rw_{i,j}) = (rs_{i,j}, rw_{i,j}) +
            Σ_{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}) + Σ_{m:orig(m)=j} (s(m), w(m))
     }
  6. for all j∈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 = L_i − {j}
     }
  7. for all j∈L_i {
     a. choose α_{i,j} ≥ 0 such that Σ_j α_{i,j} = 1
     b. choose β_{i,j} ≥ 0 such that Σ_j β_{i,j} = 1 and β_{i,i} = 0
     c. compute (srs_{i,j}, srw_{i,j}) = (β_{i,j}(α_{i,i}s_i − x_i), β_{i,j}(α_{i,i}w_i − 1))
     d. send (α_{i,j}s_i, α_{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} + α_{i,j}s_i, rw_{i,j} + α_{i,j}w_i)
     }
}
```

Fig. 4. (Asynchronous) G-GAP - Pseudo code for node i. N is the set of live neighbors of node i at round 0.

received from node j during the previous round, and they are decremented by the acknowledgements received from j. In step 5.b, node i computes the acknowledgements that will be sent to all neighbors j at the end of the round. For a neighbor j, the acknowledgement is the sum of the recovery shares $srs_{i,j}$ and $srw_{i,j}$ sent to j during the previous round, plus the mass received during the previous round from j. Note that the pair $(acks_{i,j}, ackw_{ij})$ is computed as what node i believes to be j's recovery information on i. For the synchronous case, this would be the pair $(rs_{j,i}, rw_{j,i})$ in the previous round.

Step 6 handles failure detection and the recovery from crash failures. If the external failure detector determines that a neighbor has failed, then the recovery shares associated with the failed node are added to the sum and weight variables (step 6.a). Also, the failed node is removed from the set of (live) neighbors (step 6.c).

Finally, in step 7, the node computes the shares of its sum and weight variables, as well as the shares for recovery information (steps 7.a and 7.b), sends out a message to each of its live neighbors (step 7.d) and updates the recovery shares it maintains for its neighbors (step 7.e).

For the further discussion of G-GAP, the asynchronous setting makes some changes in modeling and 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 in Section VII is performed by using G-GAP with such an extension.)

G-GAP protocol assumptions. For 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 an 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. This 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_{detect}, 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 $\lfloor t_{fail}, t_{fail} + \Delta t_{detect} \rfloor$. 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.

Correctness and convergence. 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 $t \in \{t_n | n \in \omega\}$.

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* $n \in \omega$,

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

We use the notation

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

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

The statement on mass invariant now needs to take into account both received and pending messages.

Proposition 3 (Mass invariant, 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 $t_n > 0$:

$$\begin{aligned} & 1.\ \sum_{i \in L_{t_n}} x_i \\ = \ & \sum_{j \in L, i \in L_{t_n}} s_{pending,t_n,j,i} + \sum_{i \in L_{t_n}, j \notin L_{t_n}} rs_{t_n,i,j} + \\ & \sum_{i \in L_{t_n}, j \notin L_{t_n}} (rs_{pending,t_n,j,i} - acks_{pending,t_n,j,i}) \end{aligned}$$

$$\begin{aligned} & 2.\ |L_{t_n}| \\ = \ & \sum_{j \in L, i \in L_{t_n}} w_{pending,t_n,j,i} + \sum_{i \in L_{t_n}, j \notin L_{t_n}} rw_{t_n,i,j} + \\ & \sum_{i \in L_{t_n}, j \notin L_{t_n}} (rw_{pending,t_n,j,i} - ackw_{pending,t_n,j,i}) \end{aligned}$$

Proof: See the Appendix.

Proposition 3 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 at 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 3 allows us to derive a convergence result only under timely message delivery. That is, we assume that there is some time $t_d \leq t_r$ such that, if a message is sent at time t and read at time then $t' - t \leq t_d$. 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 events will have been read by the receiving nodes, and Proposition 3 reduces to:

$$1.\ \sum_{i \in L_{t_d}} x_i = \sum_{i,j \in L_{t_d}} s_{pending,t_d,i,j}$$

$$2.\ |L_{t_d}| = \sum\nolimits_{i,j \in L_{t_d}} w_{pending,t_d,i,j}$$

Moreover, since the failure-recovery variables rs, srs, $acks$, etc. can only affect the push-synopses variables s and w during the processing of failure events, the behavior of G-GAP after time $t + t_d$ reduces to that of asynchronous push-synopses, at least up to the assignments to the s and w variables. The development is similar to that for SG-GAP, taking into account that executions and traces now refer to times instead of rounds. We have thus shown:

Lemma 2 Suppose the network graph is stable from time t' onwards. Let σ be an execution of G-GAP. Then $tr\left(\sigma\right)^{t'+t_d}$ is a trace of push-synopses.

Convergence, then, follows, as it did for SG-GAP:

Theorem 3 (Convergence, G-GAP)

G-GAP is $\varepsilon - \delta$ convergent on graphs that are eventually stable.

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.

VII. Experimental Evaluation

We have evaluated G-GAP through extensive simulations using the SIMPSON simulator (available at [5]), a discrete event simulator that allows us to simulate message exchanges between nodes and processing on nodes for large network topologies. In various scenarios, we measure the estimation error by G-GAP on the network nodes in function of the message rate, the network size, and the failure rate, in order to evaluate the protocol against the design goals given in Section III.

In addition to G-GAP, we run most simulation scenarios also with GAP, a tree-based aggregation protocol described in [4], which 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 90$^{\text{th}}$ 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 (available at [10]). 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

$$x'_{t,i} = \left\lfloor \left(1 + u \cos\left(\frac{2\pi t}{30}\right)\right) x_{t,i} \right\rfloor$$

where $u \in [0, 1]$ is chosen uniformly at random for each t and i. In the second trace, 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, a gossip protocol presented in [17] 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 in the sense that the overlay topology does not change during a simulation run. Unless stated otherwise, the overlay topology used in the simulations has 654 nodes. It is generated with target connectivity of 10, which produces an average distance of 3.1 hops 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+# of neighbors) and $\beta_{t,i,j}$ =1/# 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.

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 message rates of 1, 2, 4, 6, 8 and 10 msg/sec. (For G-GAP, a message rate of 1 means that the protocol executes one round per second, i.e., one protocol cycle per second. For GAP a message rate of 1 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 the behavior of the local variables. Fig. 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 message rate results in the decreasing of the estimation error. Therefore, the message rate controls the tradeoff between estimation accuracy and protocol overhead. In addition, for comparable overhead (i.e. the same message 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. Fig. 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 GAP and G-GAP 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 the Randomized Periodic UT 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 message rates and by a ratio of 5 for high message rates. This ratio is smaller than that for the UT trace.

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

B. Scalability

In this scenario, we measure the estimation accuracy of G-GAP and GAP in function of the network size. The message rate is set to 4 rounds/sec. We run simulations with GoCast-generated overlays for networks of size 82, 164, 327, 654, 1308, 2626, 5232 and 10464 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.

Fig. 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 of the network size, which confirms our expectations. in fact, for the case of periodic synthetic traces generated a random process, we would expect such a result for both GAP and G-GAP. Note also that [2] includes a scenario (for a gossip protocol that aggregates static local variables), which suggests that the variance of the estimates of the global average across all nodes is independent of the network size.

Also in this scenario, GAP clearly outperforms G-GAP in terms of accuracy, for comparable protocol overhead.

C. Robustness Against Node Failures

In this section, we evaluate the robustness properties of G-GAP in three scenarios. In the first scenario, we validate

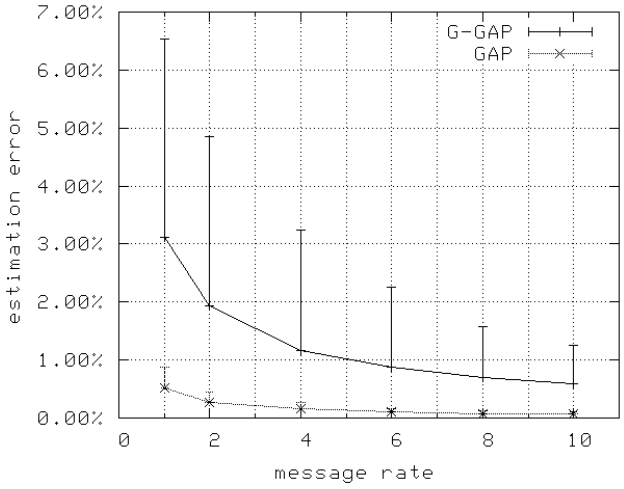

Fig. 5. Estimation error vs. protocol overhead for UT trace.

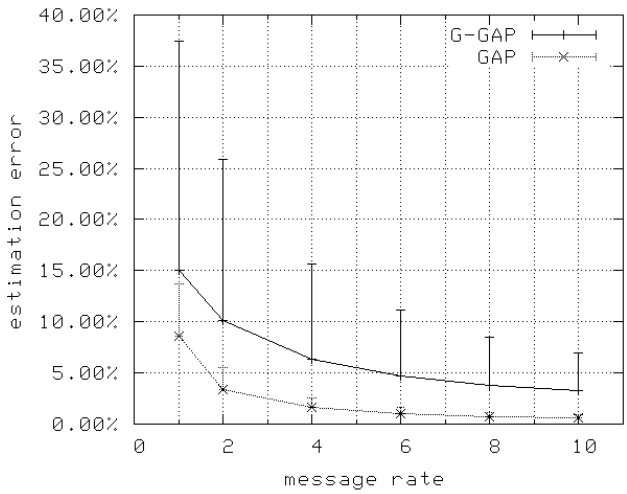

Fig. 6. Estimation error vs. protocol overhead for randomized periodic UT trace.

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 as a 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 VII-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 both G-GAP and push-synopses (which was extended to support continuous monitoring as described at the bottom of Section IV). During the simulation run, we measure the mass loss, computed as,

$$\sum\nolimits_{i \in L_{t_n}} x_i - \sum\nolimits_{j \in L, i \in L_{t_n}} s_{pending,t_n,j,i}$$

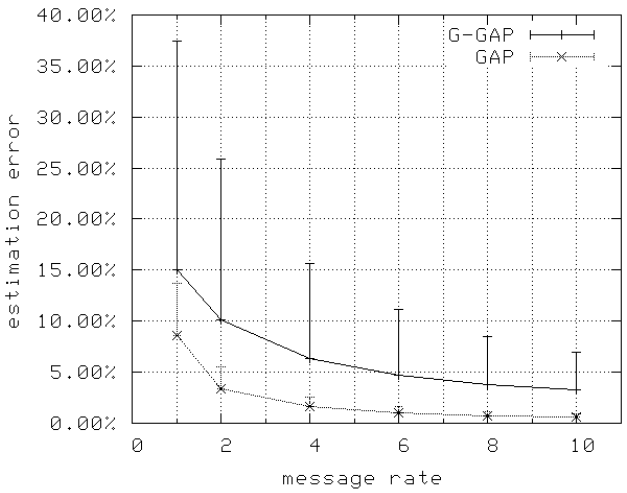

Fig. 7. Estimation error vs. network size.

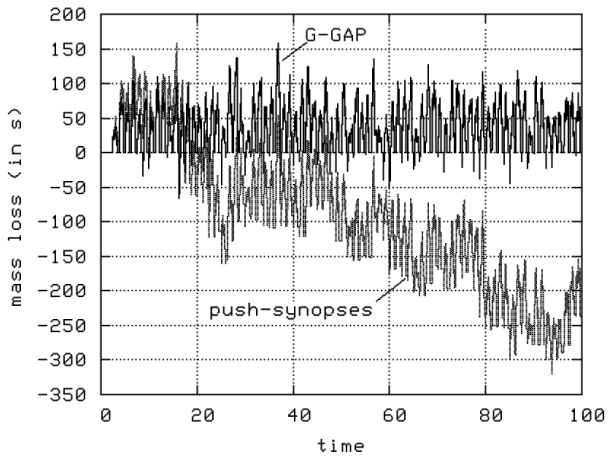

Fig. 8. Mass loss in push-synopses and G-GAP for discontiguous failures in a simulation run.

TABLE I
(TOPOLOGICAL PROPERTIES OF GOCAST-GENERATED OVERLAYS

# 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
10464	6	4.4

for s and, similarly, for w. The simulation is run for 100sec and Fig. 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 push-synopses however, we observe that mass loss accumulates over time, as it is not corrected by the protocol. 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 push-synopses, we compute the drift in estimating the aggregate, which is the estimation error due to mass loss, and the overall estimation error. We obtain two curves per protocol, which are shown in Fig. 9. Each measurement point on a curve corresponds to one simulation run of 150sec (which includes a 25sec warm-up period).

As expected, the overall estimation error by push-synopses is much larger than that by G-GAP in case of failures, because mass loss occurs at a much faster rate by push-synopses than by G-GAP. The effect of mass loss is directly visible in the two curves that show the drift. They show that the drift of push-synopses 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 the 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.

Fig. 10 shows the result obtained. Each measurement point corresponds to one simulation run. The top of the bars indicate the 90$^{\text{th}}$ 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 to us. We would have expected a gossip protocol to perform better, compared to a tree-based protocol, under high node-failure rates.

VIII. DISCUSSION AND FUTURE WORK

This paper includes two main contributions. First, we present a new 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. in [20].

The simulation studies suggest that we have achieved the design goals for G-GAP set out in Section III. First, we have shown that the tradeoff between estimation accuracy and the

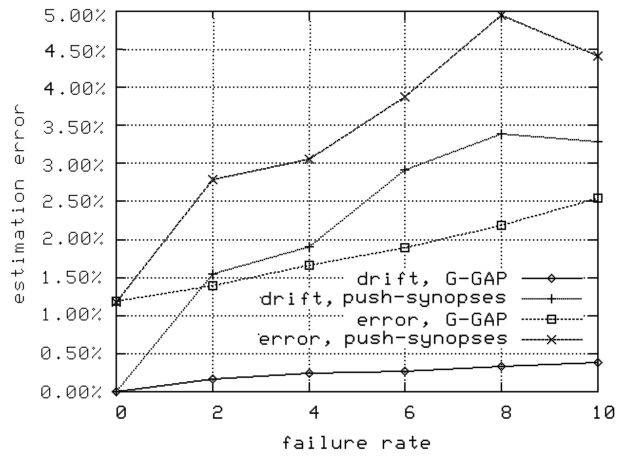

Fig. 9. Drift and overall estimation error vs. failure rate by G-GAP and push-synopses.

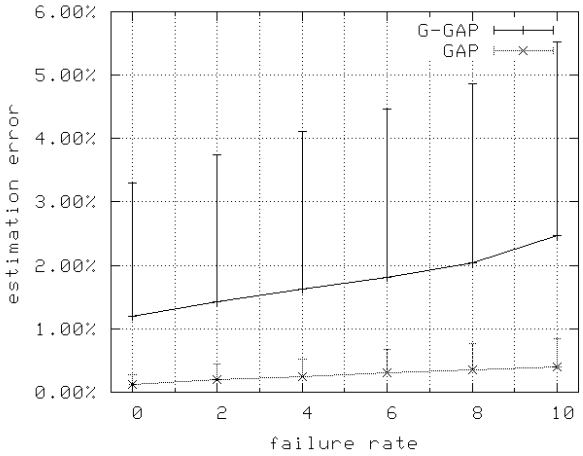

Fig. 10. Estimation error vs. failure rate by GAP and G-GAP.

protocol overhead can be controlled by varying the message rate. Second, for traces based on traffic measurements, 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 (for the trace we used), 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 vs. GAP, a fairly standard tree-based aggregation protocol. This assessment is significant, as it compares gossip-based and tree-based monitoring—the first such experimental comparison to our knowledge. 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 protocol, independent of network size and independent of frequency of failures that occur in the network. A more recent study by us presented in [29] suggests 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. While more work is needed to evaluate the relative advantages and disadvantages of tree-based vs. gossip-based monitoring, this paper contributes 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 message 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 the aggregation function AVERAGE. 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 would like to understand the convergence properties of gossip protocols better. As we have shown, the convergence analysis for G-GAP reduces to that of push-synopses quite easily, as we can establish bounds after which, in stable state, the behavior of G-GAP and push-synopses is identical. This result 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.

ACKNOWLEDGEMENT

This work has been supported by a grant from Cisco Systems, a personal grant from the Swedish Research Council (VR), the EC IST-EMANICS Network of Excellence (#26854), the ACCESS Linnaeus Center at KTH, and the EU FP7 project 4WARD.

REFERENCES

[1] S Voulgaris, D Gavidia, and M Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *J. Network Syst. Manage.*, vol. 13, no. 2, p. 197, 2005.

[2] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Computer Syst.*, vol. 23, no. 3, pp. 219–252, Aug. 2005.

[3] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. 44th Annual IEEE Symposium Foundations Computer Science (FOCS)*, Oct. 2003.

[4] M. Dam and R. Stadler, "A generic protocol for network state aggregation," in *Proc. Radiovetenskap och Kommunikation (RVK)*, June 2005.

[5] K. S. Lim and R. Stadler, "SIMPSON — A simple pattern simulator for networks," Aug. 2006 [Online]. Available: http://www.s3.kth.se/lcn/software/simpson.htm

[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Trans. Networking*, vol. 14, no. SI, pp. 2508–2530, June 2006.

[7] R. van Renesse, K. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring," *ACM Trans. Computer Syst.*, vol 21, no. 2, pp. 164–206, May 2003.

[8] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *Proc. 4th IEEE International Symposium Network Computing Applications (NCA)*, July 2005.

[9] A. Demers, D. Green, C. Hauser, W. Irish, and J. Larson, "Epidemic algorithms for replicated database maintenance," in *Proc. 6th Annual ACM Symposium Principles Distributed Computing*, Aug. 1987.

[10] University of Twente - Traffic Measurement Data Repository, Aug. 2006 [Online]. Available: http://m2c-a.cs.utwente.nl/repository/

[11] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Hierarchical in-network data aggregation with quality guarantees," in *Proc. 9th International Conf. Extending Database Technology (EDBT)*, Mar. 2004.

[12] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: A tiny aggregation service for ad-hoc sensor networks," in *Proc. 5th Symposium Operating Systems Design Implementation (USENIX - OSDI)*, Dec. 2002.

[13] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *International J. Very Large Data Bases*, vol. 13, no. 4, pp. 384–403, Dec. 2004.

[14] K. S. Lim and R. Stadler, "Real-time views of network traffic using decentralized management," in *Proc. 9th IFIP/IEEE International Symposium Integrated Network Management (IM)*, May 2005.

[15] P. C. Roth, D. C. Arnold, and B. P. Miller, "MRNet: A software-based multicast/reduction network for scalable tools," in *Proc ACM/IEEE Conf. Supercomputing (SC)*, Nov. 2003.

[16] F. Wuhib, A. Clemm, M. Dam, and R. Stadler, "Decentralized computation of threshold crossing slerts," in *Proc 16th IFIP/IEEE Distributed Systems Operations Manage. (DSOM)*, Oct. 2005.

[17] C. Tang and C. Ward, "GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication," in *Proc. International Conf. Dependable Systems Networks (DSN)*, June–July 2005.

[18] A. G. Prieto and R.Stadler "Adaptive distributed monitoring with accuracy objectives," to be published.

[19] F. Wuhib, M. Dam, R. Stadler, and A. Clemm, "Robust monitoring of network-wide aggregates through gossiping," in *Proc. Integrated Manage. (IM)*, 2007, pp. 226–235. [Online]. Available: http://www.ee.kth.se/php /index.php?action=publications

[20] M. Mehyar, D. Spanos, J. Pongsajapan, S. Low, and R. Murray, "Asynchronous sistributed averaging on communication networks," *IEEE/ACM Trans. Networking*, Aug. 2007.

[21] F. Cristian, R. de Beijer, and S. Mishra, "A performance comparison of asynchronous atomic broadcast protocols," *Distributed Syst. Engineering*, 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," in *Proc. 45th IEEE Conf. Decision Control (CDC)*, Dec. 2006.

[24] J. N. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. diss., 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 Trans. Inform. Theory*, vol. 52, no. 11, pp. 4753–4766, 2006.

[26] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, "Randomized rumor spreading," in *Proc. 41st Annual Symposium Foundations Computer Science (FOCS)*, Nov. 2000.

[27] D. Kempe and J. Kleinberg, "Protocols and impossibility results for gossip-based communication mechanisms," in *Proc. 43rd Annual IEEE Symposium Foundations Computer Science (FOCS)*, Nov. 2002, p. 471.

[28] K. Birman, "The promise, and limitations, of gossip protocols," *ACM SIGOPS Operating Syst. Review Archive*, vol. 41 , no. 5, Oct. 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] J. S. Yedidia,W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Exploring Artificial Intelligence in the New Millennium*, Chap. 8, pp. 239-269. Morgan Kaufman Publishers Inc., Jan. 2003.

[31] D. Mosk-Aoyama and D. Shah, "Computing separable functions via gossip," in *Proc. 25th Annual ACM Symposium Principles Distributed Computing (PODC)*, July 2006.

[32] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analysis," in *Proc. 36th Annual ACM Symposium Theory Computing (STOC)*, June 2004.

[33] B. Lahiri and S. Tirthapura, "Computing frequent elements using gossip," in *Proc. 15th International Colloquium Structural Inform. Commun. Complexity (SIROCCO)*, June 2008.

[34] I. Gupta, R. V. Renesse, and K. P. Birman, "Scalable fault-tolerant aggregation in large process groups," in *Proc. International Conf. Dependable Syst. Networks (DSN)*, July 2001.

APPENDIX
PROOF OF PROPOSITION 3

Proposition 3 (Mass conservation, G-GAP)

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

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

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

Proof (sketch). We prove only 1 here. The proof of 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 1 holds at all time instants up to and not including $fail_1$ (Lemma 3), and that if 1 holds at time $fail_n - 1$ then it holds at all times $t \in [fail_n, fail_{n+1} - 1]$ (Lemma 5). Note that we may assume $fail_1 > 0$. The result then follows by induction.

Lemma 3 (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,

$$\begin{aligned}
&\sum_{j,i\in L} s_{pending,t+1,j,i} \\
=\ &\sum_{j,i\in L} s_{pending,t,j,i} - \sum_{j\in L} s_{read,t+1,j,a} \\
&+\sum_{j\in L} s_{write,t+1,a,j} \\
&\text{(by axiom 1)} \\
=\ &\sum_{j,i\in L} s_{pending,t,j,i} - \sum_{j\in L} s_{read,t+1,j,a} \\
&+\sum_{j\in L} \alpha_{a,j} s_{t+1,a} \\
&\text{(by the protocol)} \\
=\ &\sum_{j,i\in L} s_{pending,t,j,i} \\
&-\sum_{j\in L} s_{read,t+1,j,a} + s_{t+1,a} \\
&\text{(by the definition of } \alpha) \\
=\ &\sum_{j,i\in L} s_{pending,t,j,i} = \sum_{i\in L} x_i \\
&\text{(by the protocol)}
\end{aligned}$$

QED.

Note that Proposition 3 reduces to Lemma 3 for time instants prior to $fail_1$.

Lemma 4 (Recovery information)

For any node α and any time $t \geq 0$,

$$\begin{aligned}
&\sum_{j\in L_t} s_{pending,t,j,a} - x_a \\
=\ &\sum_{j\in L_t\setminus a} rs_{t,j,a} \\
&+\sum_{j\in L_t\setminus a} \left(rs_{pending,t,a,j} - acks_{pending,t,a,j}\right).
\end{aligned}$$

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

$$\begin{aligned}
&\beta_{t,a,b}\left(\alpha_{t,a,a} s_{t,a} - x_a\right) + s_{pending,t,b,a} = \\
&rs_{t,b,a} + \left(rs_{pending,t,a,b} - acks_{pending,t,a,b}\right) \qquad (1)
\end{aligned}$$

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 5 (Induction step)

Assume that node $z \in L$ fails at time $fail_n$. If

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

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

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

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

$$\begin{aligned}
&\sum_{j\in L_{fail_{n-1}} i\in L_{fail_n}} s_{pending,fail,,j,i} \\
=\ &\sum_{i,j\in L_{fail_{n-1}}} s_{pending,fail_{n-1},j,i} \\
&-\sum_{i\in L_{fail_{n-1}}} s_{pending,fail,n-1,i,z}
\end{aligned}$$

Therefore

$$\begin{aligned}
&\sum_{i\in L_{fail_n}} x_i \\
=\ &\sum_{j\in L_{fail_{n-1}}, i\in L_{fail_n}} s_{pending,fail_n,j,i} \\
&+\sum_{i\in L_{fail_{n-1}}} s_{pending,fail_{n-1},i,z} - x_2 \\
=\ &\sum_{j\in L_{fail_{n-1}}, i\in L_{fail_n}} s_{pending,fail_n j,i} \\
&+\sum_{i\in L_{fail_{n-1/z}}} rs_{fail,n-1,j,z} \\
&+\sum_{i\in L_{fail_{n-1/z}}} \\
&\left(rs_{pending,fail_{n-1},z,j} - acks_{pending,fail_{n-1},z,j}\right)
\end{aligned}$$

(by Lemma 4)

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

as desired. For the induction step assume that (2) holds at some time $t : fail_n \leq t < fail_{n+1} - 1$. We show it also holds at time $t+1$. Let

$$\begin{aligned}
M_t \ =\ &\sum_{j\in L, i\in L_t} s_{pending,t,,j,i} + \sum_{i\in L_t, j\notin L_t} rs_{t,i,j} + \\
&\sum_{i\in L_t, j\notin L_t} (rs_{pending,t,j,i} - acks_{pending,t,j,i})
\end{aligned}$$

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

$$\begin{aligned}
M_{t+1} \ =\ &\sum_{j\in L, i\in L_t} s_{pending,t,,j,i} \\
&-\sum_{j\in L} s_{read,t+1,j,\alpha} + \\
&\sum_{j\in L_{fail_{n-1}}} \alpha_{a,j} s_{t+1,a} \\
&+\sum_{j\in L_t\setminus a} rs_{t,j,z} + rs_{t+1,a,z} + \\
&\sum_{j\in L_t} (rs_{pending,t,z,j} - acks_{pending,t,,j,i}) - \\
&\sum_{j\in L_t} (rs_{read,t+1,z,j} - acks_{read,t+1,,j,i})
\end{aligned}$$

(since only z's message is in the network

$$\begin{aligned} = & \sum_{j\in L, i\in L_t} s_{pending,t,,j,i} - \sum_{j\in L} rs_{read,t+1,j,a} + \\ & \sum_{j\in L} s_{read,t+1,j,a} + \sum_{j\in L_t \setminus a} rs_{t,j,z} + rs_{t,a,z} + \\ & \sum_{j\in L_t} (rs_{pending,t,z,j} - acks_{pending,t,j,i}) \end{aligned}$$

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

$$\begin{aligned} & \sum_{j\in L_{fail_{n-1}}} \alpha_{a,j} s_{t+1,a} \\ = & \sum_{j\in L} s_{read,t+1,j,a} \\ + & \quad rs_{t,a,z} + rs_{read,t+1,z,\alpha} - acks_{read,t+1,z,a} \end{aligned}$$

and

$$rs_{t+1,a,z} = 0$$

and if a did not learn about the failure of z then

$$\sum_{j\in L_{fail_{n-1}}} \alpha_{a,j} s_{t+1,a} = \sum_{j\in L} s_{read,t+1,j,a}$$

and

$$\begin{aligned} rs_{t+1,a,z} &= rs_{t,a,z} + rs_{read,t+1,z,a} - acks_{read,t+1,z,a}) \\ &= \sum_{j\in L, i\in L_t} s_{pending,t,,j,i} + \sum_{j\in L_t} rs_{t,j,z} + \\ &\quad \sum_{j\in L_t} (rs_{pending,t,z,j} - acks_{pending,t,j,i}) \\ &= \sum_{j\in L_t} x_i \end{aligned}$$

as was to be proved.