A Gossiping Protocol for Detecting Global Threshold Crossings

Fetahi Wuhib, Mads Dam, Rolf Stadler

***Abstract*—We investigate the use of gossip protocols for the detection of network-wide threshold crossings. Our design goals are low protocol overhead, small detection delay, low probability of false positives and negatives, scalability, robustness to node failures and controllability of the trade-off between overhead and detection delay. Based on push-synopses, a gossip protocol introduced by Kempe et al., we present a protocol that indicates whether a global aggregate of static local values is above or below a given threshold. For this protocol, we prove correctness and show that it converges to a state with no overhead when the aggregate is sufficiently far from the threshold. Then, we introduce an extension we call TG-GAP, a protocol that (1) executes in a dynamic network environment where local values change and (2) implements hysteresis behavior with upper and lower thresholds. Key elements of its design are the construction of snapshots of the global aggregate for threshold detection and a mechanism for synchronizing local states, both of which are realized through the underlying gossip protocol. Simulation studies suggest that TG-GAP is efficient in that the protocol overhead is minimal when the aggregate is sufficiently far from the threshold, that its overhead and the detection delay are largely independent on the system size, and that the trade-off between overhead and detection quality can be effectively controlled. Lastly, we perform a comparative evaluation of TG-GAP against a tree-based protocol. We conclude that, for detecting global threshold crossings in the type of scenarios investigated, the tree-based protocol incurs a significantly lower overhead and a smaller detection delay than a gossip protocol such as TG-GAP.**

I. Introduction

The work in this paper has been conducted as part of a research agenda where we investigate the use of gossip protocols for the purpose of 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 [1]. To date, however, no gossip-based monitoring systems have been built, and our recent work on gossip-based aggregation for real-time monitoring [2] is the first to evaluate gossip-based monitoring against the traditional alternative of tree-based monitoring.

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 with, whereby the selection function is often probabilistic. Nodes interact via "small" messages, which are processed and trigger local state changes (cf. [1], [3], [4]). Gossip protocols were originally proposed in [3] 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., [5]), network slicing (e.g., [6]), estimating the network size (e.g., [7]) and monitoring of network-wide aggregates (e.g., [4], [2]).

Manuscript received August 3, 2009, revised December 22, 2009, accepted January 13, 2010. Associate editor: Adarsh Sethi.

The authors are members of the ACCESS Linnaeus Center at KTH, Royal Institute of Technology, Sweden.

Digital Object Identifier: 10.1109/T-WC.2008.I9P0329.

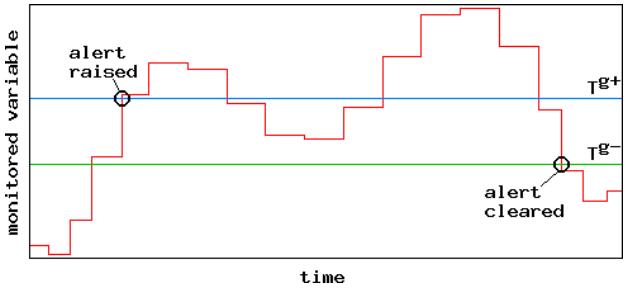

Fig. 1. Threshold Crossing Alerts: an alert is raised when a monitored variable crosses a given threshold T^{g+} from below. The alert is cleared when the variable crosses a lower threshold T^{g-} from above.

This paper focuses on gossip protocols for detecting threshold crossings. Threshold crossing alerts (TCAs) indicate to a management system that a monitored management variable, for instance a device counter, has crossed a preconfigured value—the threshold. Variables that are monitored for threshold crossing typically contain performance-related data, such as link utilization or packet drop rates. In order to avoid repeated TCAs in case the monitored variable oscillates, a threshold T^{g+} is typically accompanied by a second threshold T^{g-} called the hysteresis threshold, set to a lower value. The hysteresis threshold must be crossed, in order to clear the TCA and allow a new TCA to be triggered when the threshold is crossed again (see Figure 1).

More specifically, this work centers around raising and clearing TCAs when the monitored variable is a global aggregate of local variables across the network. Such aggregates are computed from local variables using aggregation functions, including SUM, MAX and AVERAGE. Use cases in this context are (1) raising an alert when the average load across all network links exceeds 50% and clearing the alert when it falls below 40%, and (2) raising an alert when the number of active voice calls in a given domain exceeds 20,000 and clearing the alert when that number falls below 18,000. Specific management tasks that require such functionality include network surveillance, service assurance and traffic control.

In deployed monitoring systems today, TCAs are raised and cleared in a centralized fashion, whereby a management station collects device-level data, performs aggregation on this data and evaluates a predicate (i.e., the condition for threshold crossing) that determines whether to raise or clear an alert.

To allow for scalable solutions, decentralized approaches to the TCA problem have been developed, which are based on a distributed spanning tree in the network over which aggregates are computed [8], [9]. The predicate is then evaluated at the root of the tree.

In this paper, we propose a gossip protocol to detect threshold crossings of aggregates in a decentralized way. The work is based on the push-synopses protocol by Kempe et al., a gossip protocol for computing aggregates in a network [4]. We first introduce a simple extension of push-synopses that determines, in an efficient way, whether an aggregate is above or below a threshold, under the assumption that the local values do not change. For this protocol, we prove two properties related to correctness and efficiency. Then, we extend the simple protocol to a protocol that (1) executes in a dynamic network environment where local values change and (2) implements the hysteresis behavior with an upper and lower threshold (See Figure 1). We refer to this protocol as TG-GAP for TCAs using a Gossip-based Generic Aggregation Protocol. We evaluate TG-GAP through simulation, focusing on the protocol efficiency and the quality of detecting threshold crossing (for different network sizes, frequency of threshold crossings and churn rates). For the sake of comparison, we run scenarios with a tree-based protocol for detecting threshold crossings, which provides us with insight into the performance of gossip-based vs. tree-based monitoring.

This paper is a revised and extended version of [10]. While in [10] we map out the design space for a gossip-based protocol for detecting threshold crossings, this work focuses on proposing two specific protocols, prove properties and provide simulation results. The simulation studies in this work are much more thorough than those presented in [10]. Most importantly they include a comparative evaluation of gossip-based vs. tree-based threshold detection.

The rest of the paper is organized as follows. Section II reviews related work. Section III describes the architectural assumptions and the design goals for our protocol TG-GAP. Section IV provides the background for gossip-based aggregation. Section V presents the simple protocol and provides proofs for some of its properties. Section VI introduces TG-GAP. The evaluation of TG-GAP and the comparison to tree-based protocol for detecting threshold crossings is presented in Section VII. Finally, Section VIII concludes the paper.

II. Related work

The naïve approach to detect threshold crossings of global aggregates is to continuously poll the network devices, aggregate this data, and evaluate the aggregate for threshold crossings. This approach is clearly not scalable, specifically given the fact that, in general, many variables are monitored for threshold crossing at the same time. Several works that address this problem have been recently reported. They follow either a centralized approach (e.g., [11], [12], [13], [14]) or a distributed approach (e.g., [8], [9]). A common property of the proposed solutions is that they exhibit a low management overhead when the aggregate is far from the threshold. For all these solutions, local predicates define conditions under which nodes send updates as part of the aggregation process, with the goal of achieving a low protocol overhead. [8] contains a detailed overview of the state of the art in threshold detection.

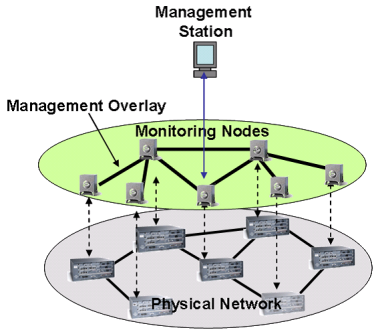

Fig. 2. Architecture of the decentralized monitoring system. Gossip protocols run in the management overlay.

In the context of introducing a protocol design methodology, Gupta et al. present a gossip protocol for detecting threshold crossing of the aggregation function AVERAGE [15]. This protocol has similarities with the simple protocol presented in Section V. Both protocols converge to a state where nodes do not exchange messages if the aggregate is below the threshold, and they converge to a state where all nodes exchange messages if the aggregate is above the threshold. Nodes have state variables that indicate whether the aggregate is above or below the threshold.

In our earlier work [8], we proposed TCA-GAP, a protocol that augments a tree-based aggregation protocol with a rate adaptation scheme for detecting threshold crossings. The key idea is to recursively assign local thresholds to subtrees and to introduce a scheme that dynamically adjusts these thresholds to network conditions and external events. In Section VII-H, we present a comparative evaluation of the protocol presented in this paper with TCA-GAP.

III. Architecture and protocol design goals

The protocols discussed in this paper are designed for a management architecture shown in Figure 2, 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 2 corresponds to a management process.) These management processes communicate via a management overlay for the purpose of monitoring the network. 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 a monitoring protocol and to receive updates of some monitored variable. For the case of monitoring threshold crossing, the node notifies the management station whenever a threshold crossing occurs.

The goal of this paper is to engineer a protocol for detecting network-wide threshold crossings with the following properties.

- **Efficiency**: The communication and processing overhead of the protocol should be small, specifically during peri-

ods where the aggregate is far (above or below) from the threshold .

- **Quality of detection**: The protocol should achieve small delays for detecting threshold crossings, and false positives and false negatives should be extremely rare.
- **Scalability**: The protocol should allow for efficient operation with high quality of detection in large networks with at least thousands of nodes.
- **Robustness**: The protocol should allow for a continuous operation after node or link failures, as well as after addition and removal of nodes.
- **Controllability**: The protocol should allow for controlling the trade-off between quality of detection and protocol overhead through management parameters that can be adjusted at runtime.

In this work, we assume that the protocol is executed in nodes of a partially synchronous network with bounded communication and processing delays, as well as non-synchronized local clocks. Each node of the network graph has an associated local (management) variable $x_i(t) > 0$ whose aggregate is monitored for threshold crossing. The pseudo-code descriptions of the protocols are given for the case where no message is lost and nodes do not fail.

IV. Background: Gossip-based aggregation

This section contains background information on the push-synopses protocol [4], which will be needed to understand the design rationale and the evaluation of the protocols introduced in this paper. The pseudo-code of push-synopses is given in Algorithm 1 and follows the presentation in [4]. The protocol estimates the average of the local state variables $x_i > 0$ over all nodes i of a given network graph. At the end of each iteration (or round), shares of s_i and w_i are distributed to nodes according to the variables $\alpha_{i,j}$, whereby $\alpha_{i,j}$ is chosen in such a way that $\alpha_{i,j} \geq 0$ and $\sum_j \alpha_{i,j} = 1$. After each round, the local estimate a_i of the global average at node i can be obtained as $a_i = \frac{s_i}{w_i}$.

The push-synopses protocol can be instantiated in different ways. Initializing s_i to x_i and w_i to 1, as shown in Algorithm 1, causes $\frac{s_i}{w_i}$ to converge towards the average $\left(\frac{1}{n}\sum_i x_i\right)$, where n is the number of nodes. Alternatively, initializing s_i to x_i and w_i to 0 for all nodes i, except for a selected node j for which w_j is initialized to 1, results in $\frac{s_i}{w_i}$ converging to the sum $(\sum_i x_i)$. As shown in [2], the ability to compute SUM and AVERAGE allows computing many aggregation functions that can be written in the form of $f\left(\sum_i g_1(x_i) + \sum_i g_2(x_i) + ... \sum_i g_k(x_i)\right)$, such as MIN/MAX, PRODUCT, various means and moments, etc.

The choice of $\alpha_{i,j}$ defines the so-called *communication strategy* used by the protocol. The main strategy studied in [4] 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. [16]). For the case of push-synopses, this means that (1) $\alpha_{i,i} = 0.5$ for all nodes i, (2) $\alpha_{i,j} = 0.5$ for exactly one $j \neq i$ and (3) $\alpha_{i,k} = 0$ for all k with $k \neq i$ and $k \neq j$.

round 0
1: $s_i = x_{i,0}$; $w_i = 1$;
2: send (s_i, w_i) to self
round $r > 0$
3: let $M = \{(s_l^*, w_l^*)\}$ be all pairs sent to i during round $r-1$
4: $s_i = \sum_l s_l^*$; $w_i = \sum_l w_l^*$;
5: choose shares $\alpha_{i,j} \geq 0$ for all nodes $j \in N_i$ such that $\sum_j \alpha_{i,j} = 1$
6: for all $j \in N_i | \alpha_{i,j} > 0$ send $(\alpha_{i,j} s_i, \alpha_{i,j} w_i)$ to j

Algorithm 1: Push-synopses: pseudo-code for node i. N_i represents i's neighbors.

The convergence property of push-synopses under uniform gossip is such that the number of rounds sufficient for an estimate to converge to within a given error margin grows with the logarithm of the network size, for a given probability. The following theorem formalizes this statement. A proof can be found in [4].

Theorem 1 (Convergence, push-synopses)**.** *Assuming uniform gossip and a complete network graph, for a fixed error ϵ and a probability δ, the probability that there is a round $r' = O(\log n)$ for which the relative error $e(r,i) \leq \epsilon$ for all i and all $r \geq r'$ is at least $1 - \delta$.*

In the theorem, the relative error $e(r,i)$ after round r on node i is defined as $e(r,i) = |\frac{a_{r,i}}{\sum_j x_j} - \frac{1}{n}|$ where $a_{r,i}$ denotes the value of a_i at the end of round r.

For readability, Theorem 1 is stated for fixed error bounds ϵ and δ. More generally, arbitrary bounds ϵ and δ are allowed, and then r' becomes $O(\log n + \log \frac{1}{\epsilon} + \log \frac{1}{\delta})$.

A communication strategy different from uniform gossip is one we call *deterministic gossip*, whereby a node i chooses $\alpha_{i,i} = \alpha_{i,j}$ for all neighbors j and $\alpha_{i,k} = 0$ for all other nodes. Hence, a node i with degree d_i chooses $\alpha_{i,i} = \alpha_{i,j} = \frac{1}{d_i+1}$ for all neighbors j. Under this strategy, for an arbitrarily connected graph, Olshevsky et al. give an upper bound of $O(n^3 \log \frac{n}{\epsilon})$ and a lower bound of $O(n^3 \log \frac{1}{\epsilon})$ on the number of rounds needed for the relative estimation error to have fallen to within ϵ [17].

Note that while the above convergence properties for deterministic gossip are much worse than that for uniform gossip, one has to take into account that uniform gossip assumes a fully connected graph while deterministic gossip assumes only that the graph is connected. In fact, simulation results show that deterministic gossip on connected graphs with small diameter seem to converge with the logarithm of the system size, just as uniform gossip.

For the protocols presented in this paper, we assume that they apply either uniform gossip or deterministic gossip as their communication strategy. For the purpose of real-time monitoring, we envision overlay topologies that are not fully connected, for scalability reasons. Therefore, we believe that deterministic gossip is the appropriate strategy for many practical settings.

V. A SIMPLE PROTOCOL FOR THRESHOLD DETECTION

In this section, we present a protocol for threshold detection, which we refer to as the 'simple protocol'. It is simple in the sense that it only indicates whether the aggregate is above or below a given threshold $T = T^{g+} > 0$, and it assumes that local values do not change over time. In the next section, we extend this protocol into TG-GAP, which allows local values to change and which implements hysteresis behavior using two thresholds (see Figure 1).

The simple protocol is an extension of the push-synopses protocol introduced in Section IV. In the simple protocol, a node is either in *active* state where it actively participates in the computation of the aggregate, or it is in *passive* state and refrains from actively participating in the computation. The switch between active and passive state is controlled by the local estimate of the aggregate. Specifically, a node with a local estimate larger or equal to kT for some $0 \leq k < 1$ is active and passive otherwise. Note that if $k = 0$ for all nodes, all nodes are active, and the execution of the simple protocol reduces to that of push-synopses (more precisely, to push-synopses with an additional state variable that indicates whether or not the aggregate is above the threshold).

A key idea behind the protocol design is that, if the aggregate is close to or above the threshold, all nodes will eventually become active, allowing the local value of the estimate to converge to its true value. Similarly, if the aggregate is below the threshold, all nodes will eventually become passive and the overhead will become 0. We prove these properties in subsection V-B.

A. Pseudo-code

Similar to push-synopses, the simple protocol maintains sum and weight variables s_i and w_i on a node i. In addition, a node also maintains a variable $crossed_i$ that indicates its belief whether the threshold is crossed or not. As in push-synopses, at the end of each protocol cycle, a node sends out messages of the format (s, w) to its neighbors.

The pseudo-code for the simple protocol for a node i is given in Algorithm 2. The upper part (**round** 0) describes the initialization of the protocol, and the lower part (**round** $r > 0$) describes the protocol cycle that is executed for each round r. N_i denotes the set of neighbors of node i and $T = T^{g+}$ denotes the threshold.

The initialization of the simple protocol starts, in a similar way as push-synopses, by initializing s_i and w_i for the desired aggregation function (step 1). The $crossed_i$ variable is initialized to $true$ if $(\frac{s_i}{w_i} > T)$ and to $false$ otherwise (step 1). The initialization phase of the protocol completes by the node sending the pair (s_i, w_i) to itself (step 2).

At the start of each protocol cycle (**round** $r > 0$), a node processes the messages it has received during the previous round and updates its local state. In step 4, s_i and w_i are set to the sum of the s and w components, respectively, of all messages received during the previous round. The variable $crossed_i$ is updated to reflect whether the local estimate of the aggregate, $\frac{s_i}{w_i}$, is above or below the threshold.

round 0
1: $s_i = x_{i,0}$; $w_i = 1$; $crossed_i = (\frac{s_i}{w_i} > T)$
2: send (s_i, w_i) to self
round $r > 0$
3: let $M = \{(s_l^*, w_l^*)\}$ be all pairs sent to i during round $r-1$
4: $s_i = \sum_l s_l^*$; $w_i = \sum_l w_l^*$; $crossed_i = (\frac{s_i}{w_i} > T)$
5: **if** $(s_i/w_i \geq kT)$ **then**
6: choose shares $\alpha_{i,j} \geq 0$ for all nodes $j \in N_i$ such that $\sum_j \alpha_{i,j} = 1$
7: **else**
8: choose shares $\alpha_{i,j} \geq 0$ for all nodes $j \in \{i\} \cup \{m | \frac{s_m^*}{w_m^*} \geq kT\}$ and $\alpha_{i,j} = 0$ for all others in N_i such that $\sum_j \alpha_{i,j} = 1$
9: **end if**
10: for all $j \in N_i | \alpha_{i,j} > 0$ send $(\alpha_{i,j} s_i, \alpha_{i,j} w_i)$ to j

Algorithm 2: The simple protocol: pseudo-code for node i. N_i represents the set of i's neighbors.

In step 5, the node checks whether it is active (i.e., $\frac{s_i}{w_i} \geq kT$ holds) or passive (i.e., $\frac{s_i}{w_i} < kT$ holds). If it is active, it chooses in step 6 the shares $\alpha_{i,j}$ for all of its neighbors j according to the choice of the communication strategy. If the node is passive, it chooses shares of 0 for all nodes except for itself and its active neighbors, for whom shares are chosen according to the communication strategy (step 8). Finally, in step 10, the node completes the protocol cycle by sending out shares $(\alpha_{i,j} s_i, \alpha_{i,j} w_i)$ to all nodes $\{j \in N | \alpha_{i,j} > 0\}$.

Note that the pseudo-code of the simple protocol reduces to that of push-synopses if all references to state variable $crossed_i$ are removed and steps 5-9 are replaced by step 6.

B. Analysis of the simple protocol

In this subsection, we prove two properties of the simple protocol that relate to protocol efficiency and correctness. The proofs are given for the communication strategy of deterministic gossip (see Section IV), as this strategy will likely be relevant in practical settings.

Let $G = (V, E)$ be a connected graph with n nodes that defines the network topology on which the simple protocol executes. Each node $i \in V$ has an associated local variable $x_i > 0$, over which the protocol computes the aggregate. The global threshold is $T > 0$. The parameter $k \in [0, 1]$ controls the switching of a node between active and passive state. $r \geq 0$ denotes the round of protocol execution. The set of active nodes at the end of round r is denoted by $\mathcal{A}_r$, and, similarly, the set of passive nodes by $\mathcal{P}_r$. Obviously, $\mathcal{A}_r \cup \mathcal{P}_r = V$.

We want to prove the following property of the protocol. If the global aggregate is above kT, then all nodes will eventually become active; if the global aggregate is below kT, then all nodes will eventually become passive. Regarding protocol overhead, this means that, in the first case, when all nodes are active, the overhead is the same as that of push-synopses. In the second case, when all nodes are passive, the overhead is zero. We define here the protocol overhead as the number of messages exchanged by the protocol. Formally, we state this property as:

Theorem 2 (Convergence)**.** *For the simple protocol, there exists a round $r_e < \infty$ such that*

1) *If $\frac{\sum_i x_i}{n} > kT$ then $i \in \mathcal{A}_r, \forall i \wedge \forall r \geq r_e$*
2) *If $\frac{\sum_i x_i}{n} < kT$ then $i \in \mathcal{P}_r, \forall i \wedge \forall r \geq r_e$*

Based on Theorem 2, we will prove a correctness result for the simple protocol. It states that, if the global aggregate is above the threshold, then the local variable $crossed_i$ will eventually hold the value $true$ on all nodes; if the global aggregate is below the threshold, then $crossed_i$ will eventually hold the value $false$ on all nodes. Formally, we state this result as:

Theorem 3 (Protocol Correctness)**.** *For the simple protocol, there exists a round $r_c < \infty$ such that*

1) *If $\frac{\sum_i x_i}{n} > T$ then $crossed_{i,r} = true, \forall i \wedge \forall r \geq r_c$*
2) *If $\frac{\sum_i x_i}{n} < T$ then $crossed_{i,r} = false, \forall i \wedge \forall r \geq r_c$*

Here is the idea behind the proofs of the two theorems. First, we define a quantity we call 'potential', and show that active nodes have positive potential while passive nodes have negative potential. Using properties of this quantity, we then prove Theorem 2 by showing that, if the global aggregate is above kT, then the sum of the potentials of passive nodes will eventually increase to zero; if it is below kT, then the sum of the potentials of active nodes will eventually decrease to zero. Finally, using Theorem 2, we prove Theorem 3 in a straightforward manner.

Definition 1 (Potential)**.** *The potential of node i at round r is a function of the value of the local variables of node i at round r and is given by $\Phi_{i,r} = s_{i,r} - kTw_{i,r}$. The potential of a set of nodes S is given by the sum of the potentials of the individual nodes: $\Phi_r^S = \sum_{i \in S} \Phi_{i,r}$.*

The potential has several properties that are formulated as Lemmas 1 through 5.

Lemma 1 (Positive and negative potential)**.** *At any round $r \geq 0$*

1) $\Phi_{i,r} \geq 0, \forall i \in \mathcal{A}_r$
2) $\Phi_{i,r} < 0, \forall i \in \mathcal{P}_r$

Proof: We can write $\Phi_{i,r} = w_{i,r}(\frac{s_{i,r}}{w_{i,r}} - kT)$. By definition a node is active in round r if $(\frac{s_{i,r}}{w_{i,r}} \geq kT)$ and passive otherwise. The statement thus follows, since $w_{i,r}$ is always greater than 0. ∎

Lemma 2 (Conservation of potential)**.** *The potential of the system is conserved: $\Phi_r^V = \sum_i x_i - nkT, \forall r \geq 0$.*

Therefore, we can write Φ^V, instead of Φ_r^V.

Proof:

$$\begin{aligned} \sum_i \Phi_{i,r} &= \sum_i (s_{i,r} - kTw_{i,r})) \\ &= \sum_i s_{i,r} - kT \sum_i w_{i,r} \\ &= \sum_i x_i - nkT \end{aligned}$$

For the last step, we apply the mass conservation property of push-synopses, namely, $\sum_i s_{i,r} = \sum_i x_i$ and $\sum_i w_{i,r} = n, \forall r$ [4]. ∎

Lemma 3 (Transfer of potential)**.** *In the simple protocol, when node i sends a message to node j in round r, node i's potential is decreased by $\alpha_{i,j}\Phi_{i,r}$ and node j's potential is increased by the same amount, at round $r+1$.*

Proof: This follows from the definition of potential and the protocol. ∎

A direct consequence of this lemma is the following monotonicity property.

Lemma 4 (Monotonicity)**.** *The potential of the set of active nodes $\Phi_r^{\mathcal{A}}$ decreases monotonically with r, and the potential of passive nodes $\Phi_r^{\mathcal{P}}$ increases monotonically with r.*

Proof: Let M_r be the set of all (s, w)–pairs sent in round r. Let $M_{i,r}$ be the set of messages sent to i in round r. Obviously, $M_{i,r} \subseteq M_r$ and $M_{i,r} \cap M_{j,r} = \emptyset$ for $i \neq j$. Let $M_r^{\mathcal{A}}$ and $M_r^{\mathcal{P}}$ be the set of messages sent from active nodes and passive nodes, respectively, in round r, with $M_r^{\mathcal{A}} \cup M_r^{\mathcal{P}} = M_r$. Similarly, let $M_{i,r}^{\mathcal{A}}$ and $M_{i,r}^{\mathcal{P}}$ be the sets of messages received by node i, which are sent from active and passive nodes, respectively, in round r, with $M_{i,r}^{\mathcal{A}} \cup M_{i,r}^{\mathcal{P}} = M_{i,r}$. For a given set of messages M, we define $\Phi(M) = \sum_{(s_l, w_l) \in M}(s_l - kTw_l)$. Then, because of Lemma 3 and the protocol, we can write $\Phi_{r-1}^{\mathcal{A}} = \sum_i \Phi(M_{i,r-1}^{\mathcal{A}})$. By definition, $\Phi_{i,r} = \Phi(M_{i,r-1}^{\mathcal{A}}) + \Phi(M_{i,r-1}^{\mathcal{P}})$. Hence

$$\begin{aligned} \Phi_r^{\mathcal{A}} &= \sum_{i \in \mathcal{A}_r} \Phi_{i,r} \\ &= \sum_{i \in \mathcal{A}_r} \Phi(M_{i,r-1}^{\mathcal{A}}) + \sum_{i \in \mathcal{A}_r} \Phi(M_{i,r-1}^{\mathcal{P}}) \\ &\leq \Phi_{r-1}^{\mathcal{A}} \end{aligned}$$

The last inequality holds since $\sum_{i \in \mathcal{A}_r} \Phi(M_{i,r-1}^{\mathcal{A}}) \leq \sum_i \Phi(M_{i,r-1}^{\mathcal{A}})$ and $\sum_{i \in \mathcal{A}_r} \Phi(M_{i,r-1}^{\mathcal{P}}) \leq 0$. We have thus proved that $\Phi_r^{\mathcal{A}}$ decreases monotonically with r. From this statement and Lemma 2, it follows that $\Phi_r^{\mathcal{P}}$ increases monotonically with r. ∎

The following Lemma shows that the positive potential decreases exponentially fast over time, assuming at least one passive node in the system. The proof builds upon the following idea. Imagine one tags and follows the contribution of an active node to the positive potential. Let D be the diameter of the network graph G and d be the degree of G (i.e., the maximum degree of any node in G), incremented by 1. Since the distance between two nodes is bounded by D and since at least $\frac{1}{d}$ times a node's contribution is transferred to the receiver upon sending a message, at least $\frac{1}{d^D}$ of a node's potential at round r would have reached some passive node by round $r + D$. Summing all these contributions up over all active nodes at round r and using Lemma 3, we arrive at the following lemma.

Lemma 5 (Decrease in positive potential)**.** *Let D be the diameter of the network graph G and d be the degree of G, incremented by 1. For the simple protocol, $\Phi_{r+D}^{\mathcal{A}} \leq$*

$(1-d^{-D})\Phi_r^{\mathcal{A}}$ *holds for all rounds* $r \geq 0$*, if there is at least one passive node at round* $r+D$.

Proof: Recall that by definition $\mathcal{P}_r = V \setminus \mathcal{A}_r$. Also, by Lemma 4, $\mathcal{P}_{r'+D} \neq \emptyset$ implies that $\mathcal{P}_r \neq \emptyset$ for all $r \leq r'+D$. For simplicity, we consider the case where $r'=0$. Extension to any $r'>0$ is straightforward.

For some $r : 0 \leq r < D$ we say that a node $v \in \mathcal{A}_r$ is $(D-r)$*-close*, if there is a sequence of nodes and messages $(v=v_r), m_r, v_{r+1}, \cdots, v_{r'-1}, m_{r'-1}, v_{r'}$ such that $r' \leq D$, $v_{r'} \in \mathcal{P}_{r'}$, for all $r'' : r \leq r'' < r'$, $v_{r''} \in \mathcal{A}_{r''}$, and, in round r'', the node $v_{r''}$ sends the message $m_{r''}$ to $v_{r''+1}$. Let $\hat{V}_r$ be the set of nodes that are $(D-r)$-close. It follows that $\hat{V}_0 = \mathcal{A}_0$, $\hat{V}_D = \emptyset$, $\hat{V}_r \subseteq \mathcal{A}_r$, and that, if $v \in \hat{V}_r$, $r < D-1$, then there is at least one node $v' \in \hat{V}_{r+1}$ such that v sends a message to v' at round r. Vice versa, if v is $(D-(r+1))$-close, $(v',v) \in E$, and $v' \in \mathcal{A}_r$ then v' is $(D-r)$-close

Let $\hat{\Phi}_r = \Sigma_{v \in \hat{V}_r} \Phi_{v,r}$ and

$$fan(v,r) = \begin{cases} 1 + card(\{v' \mid (v,v') \in E\} \\ \qquad \text{if } \frac{s_{v,r}}{w_{v,r}} \geq kT, \\ 1 + card(\{v' \mid (v,v') \in E \wedge \frac{s_{v',r}}{w_{v',r}} \geq kT\} \\ \qquad \text{otherwise.} \end{cases}$$

We obtain:

$$\begin{aligned} \hat{\Phi}_{r+1} &= \Sigma_{v \in \hat{V}_{r+1}} \Phi_{v,r+1} \\ &= \Sigma_{v \in \hat{V}_{r+1}} \Sigma_{v':(v',v) \in E} fan(v')^{-1} \Phi_{v',r} \\ &\geq d^{-1} \Sigma_{v \in \hat{V}_{r+1}} \Sigma_{v' \in V_{A,r}:(v',v) \in E} \Phi_{v',r} + \\ &\quad \Sigma_{v \in \hat{V}_{r+1}} \Sigma_{v' \in V_{P,r}:(v',v) \in E} fan(v')^{-1} \Phi_{v',r} \\ &= d^{-1} \Sigma_{v \in \hat{V}_r} \Phi_{v,r} + \\ &\quad \Sigma_{v \in \hat{V}_{r+1}} \Sigma_{v' \in V_{P,r}:(v',v) \in E} fan(v')^{-1} \Phi_{v',r} \\ &\geq d^{-1} \hat{\Phi}_r - (\Phi^{\mathcal{P}}_{r+1} - \Phi^{\mathcal{P}}_r) \ . \end{aligned}$$

The latter step is justified since the (negative) potential transferred from a passive node v' at round r to an active node v at round $r+1$ is, by Lemma 2, limited by the decrease in total potential of passive nodes from round r to round $r+1$ in absolute terms.

It follows that the mass transferred from active to passive at round $D-1$ is

$$\begin{aligned} d^{-1}\hat{\Phi}_{D-1} &\geq d^{-2}\hat{\Phi}_{D-2} - (\Phi^{\mathcal{P}}_{D-1} - \Phi^{\mathcal{P}}_{D-2}) \\ &\geq d^{-D}\hat{\Phi}_0 - (\Phi^{\mathcal{P}}_{D-1} - \Phi^{\mathcal{P}}_0) \\ &= d^{-D}\Phi^{\mathcal{A}}_0 - (\Phi^{\mathcal{P}}_{D-1} - \Phi^{\mathcal{P}}_0) \end{aligned}$$

and hence $\Phi^{\mathcal{P}}_D - \Phi^{\mathcal{P}}_0 \geq d^{-D}\Phi^{\mathcal{A}}_0$ which is sufficient to conclude the result. ∎

Now we are ready to prove Theorem 2. Let us assume that $\frac{\sum_i x_i}{n} > kT$ holds. Then, $\Phi^V > 0$ by Lemma 2. By the definition of potential, $\Phi^V = \Phi^{\mathcal{A}}_r + \Phi^{\mathcal{P}}_r$. Applying Lemma 1, we conclude that

$$\Phi^{\mathcal{A}}_r \geq \Phi^V > 0 \tag{1}$$

From Lemma 5, it follows that

$$\Phi^{\mathcal{A}}_r \leq (1-d^{-D})^{\lfloor \frac{r}{D} \rfloor} \Phi^{\mathcal{A}}_0 \tag{2}$$

for rounds r with $\mathcal{P}_r \neq \emptyset$. If there are passive nodes for all rounds r, it follows that eventually $\Phi^{\mathcal{A}}_r < \Phi^V$, a contradiction to (1). This concludes the proof of the first statement in Theorem 2.

To prove the second statement, let a *border node* be a passive node that has an active neighbor. Let $\mathcal{B}_r$ be the set of border nodes at round r and $\Phi^{\mathcal{B}}_r$ be the potential of nodes in $\mathcal{B}_r$. Note that by Lemma 2, Φ^V is a negative quantity.

Consider an infinite run of the protocol starting in a configuration such that $\Phi^V < 0$. Let V^{lim} be the nodes that are infinitely often either active or border nodes during the run, and let Φ^{lim} be the potential, eventually constant, of V^{lim}. Let r_0 be a round after which all active or border nodes are in V^{lim}.

First we observe that, if $\mathcal{A}_r$ is inhabited for all r then so is $\mathcal{B}_r$. This is so, since the network is connected, and we assume that $\Phi^V < 0$ and no node not in V^{lim} can be a border node. We want to show that a round $r \geq r_0$ exists such that

$$\Phi^{\mathcal{A}}_r \leq \Phi^{\mathcal{A}}_{r_0} + d^{-1}\Phi^{\mathcal{P},lim}_{r_0} \tag{3}$$

where $V^{\mathcal{P},lim}_{r_0} = V^{lim} \cap \mathcal{P}_{r_0}$ and $\Phi^{\mathcal{P},lim}_{r_0} = \Sigma\{\Phi_{v,r_0} \mid v \in V^{\mathcal{P},lim}_{r_0}\}$.

Let

$$\begin{aligned} W_i = \{v \in V^{\mathcal{P},lim}_{r_0} \mid v \in \mathcal{B}_{r_0+i} \text{ and} \\ \forall j : 0 \leq j < i . v \in \mathcal{P}_{r_0+j} - \mathcal{B}_{r_0+j}\} \ . \end{aligned}$$

Clearly, $\{W_i\}$ partitions $V^{\mathcal{P},lim}_{r_0}$, and $\Phi^{\mathcal{P},lim}_{r_0} = \bigcup\{\Phi_{v,i} \mid v \in W_i\}$. Also, there is some $r_1 \geq r_0$ such that $W_r = \emptyset$ whenever $r \geq r_1$. Moreover, at each round $r \geq r_0$,

$$\begin{aligned} \Phi^{\mathcal{A}}_{r+1} &\leq \Phi^{\mathcal{A}}_r + \Sigma\{\Phi_{v,r} \mid v \in W_r\} \\ &= \Phi^{\mathcal{A}}_r + \Sigma\{\Phi_{v,r_0} \mid v \in W_r\} \ , \end{aligned}$$

since all $v \in W_r$ are passive, non-border, and hence constant, between r_0 and $r-1$ inclusive. This is sufficient to conclude (3).

Since Φ^{lim} is constant, this construction can be iterated, starting now at round r_1 instead of r_0. Each such construction reduces the potential of active nodes by a constant positive amount $-d^{-1}\Phi^{lim}$, provided Φ^{lim} is negative. Therefore, it follows that, eventually, for some round r, $\Phi^{\mathcal{A}}_r = 0$ as desired.

But Φ^{lim} must be negative, since otherwise the construction of Lemma 5 can be used to show that, eventually, all nodes in V^{lim} must be active, a contradiction. This concludes the proof of statement 2) in Theorem 2 and thus of the theorem itself. ∎.

Given Theorem 2, the proof of Theorem 3 becomes straightforward. Let us assume that $\frac{\sum_i x_i}{n} > T$. Due to Theorem 2, all nodes become active after a round $r_e < \infty$. If all nodes are active, then the convergence properties of the simple protocol are the same as those of push-synopses, which means that $\frac{s_i}{w_i}$ converges to $\frac{\sum_i x_i}{n}$ [17]. Therefore, the local variable $crossed_i$ on all nodes will eventually hold the value $true$. This concludes the proof of statement 1) in Theorem 3.

To prove statement 2), assume that $\frac{\sum_i x_i}{n} < T$. There are two cases to consider. If $\frac{\sum_i x_i}{n} > kT$, then all nodes become active after a finite number of rounds. Therefore, the local variable $crossed_i$ on all nodes will eventually hold the value

$false$. In the other case, where $\frac{\sum_i x_i}{n} < kT$, Theorem 2 implies that all nodes will eventually become passive. By definition, a passive node has $\frac{s_i}{w_i} < kT$ and hence its local variable $crossed_i$ has the value $false$ as $0 \leq k \leq 1$. This concludes the proof of statement 2). ■

Note that our proof of statement 2) of Theorem 2 above does not allow to bound the convergence time. We leave the identification of such a bound for future work.

VI. TG-GAP: AN EXTENSION OF THE SIMPLE PROTOCOL

The simple protocol presented in Algorithm 2 does not provide the functionality we require, namely that of triggering threshold crossing alerts for continuously evolving global aggregates. However, it contains the core concepts of efficiently and correctly detecting whether or not an aggregate is above a given threshold. In this section, we extend the simple protocol to TG-GAP, a protocol that meets the design goals set out in Section III. To design such an extension, we need to address issues which we identified in [10]. The issues relate to dynamic rate adjustment, triggering of TCAs and implementing hysteresis behavior using an upper and a lower threshold.

Dynamic rate adjustment reduces the protocol overhead when the aggregate is far from the threshold. As proven in Theorem 2, all nodes become passive and the overhead of the simple protocol reduces to zero when the aggregate is far from the threshold. Hence, the simple protocol implements dynamic rate adjustment through switching from active to passive states and vice versa. We discuss design choices for the last two issues in the following subsections.

A. Triggering TCAs

The push-synopses protocol introduced in Section IV and the simple protocol have been presented for the case where the local variables x_i remain constant. As shown in [2], we can easily adapt the protocols to support continuous monitoring in the case where the local values change, by sampling the value of x_i at the beginning of each round and by adding the change in x_i to the sum variable s_i (see step 4 in Algorithm 3).

For a protocol adapted in such a way, triggering a TCA using the current estimate of the aggregate on a node can result in a high probability of false positives for the following reasons. First, the estimate has a local bias, in the sense that a sudden change in the local variable produces, for a short duration, a significant error in the local estimate of the aggregate. Such a bias is an intrinsic property of gossip-based aggregation protocols. Second, it is generally difficult to determine how close the current value of an estimate is to the true value of the aggregate. This is because the theoretical bounds on convergence rates given in the literature are often loose, they require global knowledge (e.g., system size, maximum of all local values, etc.), and they are given under the assumption that the local values do not change (cf. [4], [17]). Finally, for the case of the simple protocol, it is difficult for a node to determine whether its estimate of the aggregate contains up-to-date contributions from all nodes, or whether it is computed only over a subset of the active nodes.

In [10], we put forth two approaches for triggering and clearing TCAs. The first approach involves low-pass filtering, such that a node raises an alert only if the local estimate of the aggregate is above the threshold and remains so for a predetermined number of rounds denoted by r_{wait}. This reduces the effect of local bias (i.e., reducing false positives) at the risk of missing threshold crossings (i.e., increasing false negatives). The second approach extends this scheme with a global snapshot algorithm. If a node locally determines a threshold crossing, it initiates a snapshot protocol to compute a more accurate estimate of the global aggregate. Note that the result of this computation is not affected by changes in the local values during the computation of the snapshot, and that the snapshot is computed over all nodes in the network, both passive and active.

For TG-GAP we follow the second approach and use push-synopses to compute a snapshot. The snapshot is run at the protocol rate, piggy-backing the snapshot messages onto the regular protocol messages. To decrease convergence time, instead of computing a snapshot from the local values directly, TG-GAP seeds the snapshot computation on a node by initializing the local state variables with the (partially converged) estimates already held by the node. The initialization phase of the snapshot is also accelerated by allowing a node to execute the first two rounds of push-synopses as soon as it receives a snapshot request from a neighbor. The duration of the snapshot which we denote by r_{poll} is dependent on the specific networking scenarios and is determined experimentally.

B. Implementing the hysteresis behavior

In order to avoid repeated TCAs in case the monitored variable oscillates, the monitored threshold is typically accompanied by a second hysteresis threshold, set to a lower value. The hysteresis threshold must be crossed, in order to clear the TCA and allow a new TCA to be triggered when the threshold is crossed again (see Figure 1). TG-GAP implements such a hysteresis behavior.

TG-GAP maintains a local state variable dir, which has value up when the protocol is detecting the upward crossing of the upper threshold, and has value down when the protocol is detecting the downward crossing of the lower threshold.

A key component of implementing hysteresis behavior is a mode switching scheme, which switches the values of dir on all nodes, from up to down or vice versa, after TCA has been raised or cleared by the protocol.

In TG-GAP, for the case of deterministic gossiping, mode switching is realized as follows. Nodes exchange, in addition to the s and w variables, a TCA number tn and the direction of detection dir (1 for up and -1 for down). All nodes are initialized with TCA number 0 and direction 1. Whenever a node detects a threshold crossing, it increases its local TCA number by 1, switches the direction of detection and sends out a message with the updated tn and dir to its neighbors. Whenever a node receives a TCA number larger than its own, it sets its TCA number and direction to the values it received, and sends them out to its neighbors. This mechanism ensures that all even TCA numbers are associated with direction 1 and all odd numbers are associated with direction -1.

f	π
1	0%
2	79.68%
3	94.05%
4	98.02%
5	99.30%
10	99.995%

TABLE I
THE RATIO π OF INFECTED INDIVIDUALS TO THE TOTAL NUMBER INDIVIDUALS IN A POPULATION FOR THE INFECT-AND-DIE MODEL.

This mode switching scheme ensures that, after any sequence of TCAs raised by nodes of the system, the system eventually converges to a consistent state where the the TCA number tn (and thus the mode variable dir) agree on all nodes. This consistent state is reached after the number of rounds that is equal to the diameter of the network.

The mode switching scheme presented above has to be adapted for the case of uniform gossip for the TCA number and direction to spread successfully to all nodes. Specifically, if the majority of the nodes are passive, then the spread of the new state may stop before it is received by all nodes. To ensure the spread of the new state (in a probabilistic sense), nodes have to forward the state for more than a single round, even if the distance between their estimate of the aggregate and the threshold dictates that they be passive. The relationship between the number of rounds a node communicates its new state and how far the new state spreads is captured by the infect-and-die model for the spread of epidemics [18].

In the infect–and–die model, for a large population, it has been shown that the ratio π of the infected individuals to the size of a given population satisfies the fixed point equation $\pi = 1 - e^{-\pi f}$, where f is the number of rounds an infected individual remains infectious before dying. To draw a parallel with our protocol, a node that detects threshold crossing corresponds to the individual that starts the epidemic, a node receiving a new state corresponds to individuals getting infected, and a node forwarding the state corresponds to an infectious individual. (Note that in our protocol, a node will continue to 'infect' other nodes after f rounds if it becomes active due to its aggregate being close to the threshold.)

Table I shows the values of the ratio π for different values of f. From this table, one can see that, for example, using the uniform gossip communication strategy, if a node that receives a new TCA number (and direction) forwards it for at least 5 rounds, at least 99.3% of the nodes in the network are expected to have received the new TCA number.

C. Pseudo-code

The pseudo-code of TG-GAP for a node i is given in Algorithm 3. The upper part (**round** 0) describes the initialization of the protocol, and the lower part (**round** $r > 0$) describes the protocol cycle that is executed for each round $r > 0$. N_i denotes the set of neighbors of node i on the network graph.

As mentioned before, TG-GAP is an extension of the simple protocol. As does the simple protocol, TG-GAP maintains the s_i and w_i variables. Instead of the $crossed_i$ variable in the simple protocol, TG-GAP maintains a TCA number variable

```
round 0
 1: s_i = x_{i,0}; w_i = 1; tn_i = 0; dir_i = 1; cnt_i = 0;
 2: send (s_i, w_i, tn_i, dir_i) to self
round r > 0
 3: let M = {(s_l^*, w_l^*, tn_l^*, dir_l^*)} be all messages
    sent to i during round r − 1
 4: s_i = (x_{i,r} − x_{i,r−1}) + Σ_l s_l^*; w_i = Σ_l w_l^*;
 5: active = crossed(s_i/(k*w_i), dir_i);
 6: if MAX_l(tn_l^*) > tn_i or pcrossed(tn_i + 1, dir_i)
    then
 7:   raiseTCA(dir_i); active = true; cnt_i = 0
 8:   if pcrossed(tn_i + 1, dir_i) then
 9:     tn_i = tn_i + 1; dir_i = dir_i * −1;
10:   else
11:     j = argmax_l tn_l^*; tn_i = tn_j^*; dir_i = dir_j^*
12:   end if
13: end if
14: if crossed(s_i/w_i, dir_i) then
15:   if cnt_i ≥ r_wait then
16:     startpoll(tn_i + 1);
17:   else
18:     cnt_i = cnt_i + 1
19:   end if
20: else
21:   cnt_i = 0
22: end if
23: if active then
24:   choose shares α_{i,j} ≥ 0 for all nodes j ∈ N_i
      such that Σ_j α_{i,j} = 1
25: else
26:   choose shares α_{i,j} ≥ 0 for all nodes j ∈ {i} ∪
      {m | s_m^*/w_m^* ≥ kT} and α_{i,j} = 0 for all others in
      N_i such that Σ_j α_{i,j} = 1
27: end if
28: for all j ∈ N_i | α_{i,j} > 0 send (α_{i,j} s_i, α_{i,j} w_i) to j
```

Algorithm 3: TG-GAP: pseudo-code for node i.

tn_i, a direction variable dir_i, and a variable cnt_i, which stores the number of rounds that passed since the local estimate of the aggregate crossed the current threshold. TG-GAP extends the message format of the simple protocol to (s, w, tn, dir) by adding the node's TCA number and its direction to all messages that are exchanged.

For the sake of better readability, the description of taking a snapshot is simplified in the pseudo-code and encapsulated in function calls.

The pseudo-code uses the following ancillary functions:

- `crossed(`agg`,`dir`)`: Returns true if $dir = 1$ and $agg \geq T^{g+}$ or if $dir = -1$ and $agg < T^{g-}$. Otherwise it returns false.
- `startpoll(`tn`)`: Initiates computing the snapshot of the aggregate if there does not already exist an ongoing snapshot. The snapshot is given a local id of tn, overwriting any previous id.
- `pcrossed(`tn`,` dir`)`: Returns true if the snapshot with id tn has completed and the result of the snapshot is such that the threshold in the direction dir is crossed. Otherwise it returns false.
- `raiseTCA(`dir`)`: In case a management station is connected to the node, it is notified of the threshold crossing.

The protocol starts by initializing s_i and w_i in a same way as in the simple protocol. The TCA number tn_i is initialized to

0, the direction dir_i is initialized to 1 (i.e., upward crossing of a threshold), and the count variable cnt_i variable is initialized to 0 (step 1). Similar to the simple protocol, the initialization of the protocol ends with the node sending a message to itself (step 2).

At the start of each protocol cycle, a node processes the messages it has received during the previous round and updates its local state variables. The w_i variable is updated in the same way as in the simple protocol, while the s_i variable now takes into account the change to the local variable x_i (step 4). The variable $active$ is set to true, if the aggregate is close to the threshold and false otherwise (step 5).

Steps 6-13 are executed, if a node detects that a global threshold crossing has occurred. A node determines that a global threshold crossing has occurred, if a neighbor sends a message with a tn value larger than tn_i, or if a snapshot has completed and the result indicates that a global threshold crossing has occurred. In case of a global threshold crossing, the node raises a TCA, sets the $active$ variable to true, and resets the cnt_i counter (step 7). In addition, if the threshold crossing has been detected through the snapshot, then the node increments its TCA number and changes the direction of detection (step 9). Otherwise, if the crossing has been detected through information obtained from a neighbor, then the TCA number and direction variables are set to those sent by the neighbor (step 11).

Steps 14-20 are executed, if the local estimate of the aggregate is above (if $dir = 1$, or below, if $dir = -1$) the threshold. If the aggregate has been continuously above (or below) the threshold for the last r_{wait} rounds, the node initiates a snapshot of the aggregate. Otherwise, it increments cnt_i by 1. If the aggregate is below (or above) the threshold, then cnt_i is set to 0 (step 21).

In steps 23-27, node i computes, in the same way as the simple protocol, the shares $\alpha_{i,j}$ for its neighbors. If the node is active, it computes the shares according to the communication strategy (step 24). If the node is passive, it sets the shares of all passive neighbors to zero and computes the shares of its active neighbors according to the communication strategy (step 26). In step 28, the node sends a message to each node with a none-zero share.

The pseudo-code given in Algorithm 3 applies to the communication strategy of deterministic gossip. For the case of uniform gossip, a straightforward extension is required, keeping a node to remain active for f rounds before it switches to passive state (see subsection VI-B).

VII. Experimental evaluation

We have evaluated TG-GAP through extensive simulations using a Java version of the SIMPSON simulator (available at [19]), 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 protocol overhead and the quality of detection by TG-GAP, the network size, the failure rate, frequency of threshold crossings and protocol rate, in order to evaluate the protocol against the design goals given in Section III.

In addition to TG-GAP, some simulation scenarios are run also with TCA-GAP, a tree-based protocol for detecting threshold crossing of aggregates [8]. 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 threshold crossing of network-wide aggregates.

A. Evaluation against design goals

Evaluation metrics. TG-GAP is evaluated using the following metrics. First, protocol overhead is measured as the average number of messages processed per second per node. Second, we evaluate the quality of detection by measuring the correctness of the detection and the detection delay. The correctness of the detection is determined by a) the ratio of false negatives (i.e., cases where the protocol fails to raise an alert although a threshold crossing has occurred) to the total number of threshold crossings and b) the ratio of false positives (i.e., cases where the protocol raises an alert even though no threshold crossing has occurred) to the total number of alerts raised by the protocol. Finally, the detection delay is measured as the difference between the time TG-GAP reports a crossing and the time the actual crossing occurs.

Local variables. In all scenarios, a local variable represents the number of HTTP flows that enter the network at a specific router, and the aggregate of those variables represents the total number of such flows in the network. We simulate the behavior of the local variables using packet traces captured at the University of Twente [20]. Some 20 hours of traces captured at two links were used. In order to obtain simulation traces for local variables to a large number of nodes, we used the following method. The first simulation trace which we call the *UT trace* has been created as follows. The number of HTTP flows from those original traces were sampled each minute. This produced traces that give the evolution of the number of HTTP flows over time. Then, we divided the obtained traces into segments of 150sec each, which provided unique segments for a network of size of 654 nodes. From those segments, we constructed traces of 1500sec for each node in the simulation, by randomly selecting and concatenating ten of those segments. Across all traces, the average value of the local variables is about 45 flows, and the standard deviation of the change between two consecutive samples is about 3.4 flows.

The aggregate values for the traces constructed above show small changes over time, which makes it difficult to evaluate the performance of our protocol in scenarios with high number of threshold crossings. To evaluate the performance of our protocol in such scenarios, we produced a second trace, which we call *Periodic UT trace*. It is obtained by adding a sinusoidal bias to the UT trace $w_i(t)$ on a node i as follows: $w_i(t)^* = \lfloor w_i(t) + 23 * \left(1 + \sin(\frac{2\pi t}{30} - \frac{\pi}{2})\right) \rfloor$. The period of the sinusoid is chosen such that we have about 100 threshold crossings over the simulation period of 1500 seconds.

Overlay topology. The topologies used for the network graphs in our simulations are generated by GoCast [5], a gossip protocol that builds topologies with bidirectional edges and small diameters. The protocol allows setting the (target)

connectivity of the graph. For the measurements reported in this paper, we do not simulate the dynamics of GoCast. This means that the topology does not change during a simulation run. Unless stated otherwise, the topology used in the simulations has 654 nodes (this is the size of Abovenet, an ISP[21]). All topologies are generated with a target connectivity of 10, which, for the 654 node topology we use, produces an average inter-node distance of 4.3 hops and a diameter of 7 hops in the graph.

Failures. For our simulations, we assume that failure arrivals follow a Poisson process and that a failed node recovers after 30sec. We also assume a failure detection service is available to TCA-GAP, allowing a node to detect the failure of a neighbor in the network graph.

Protocol parameters. The following protocol parameters are used for our simulations. They are determined experimentally for the scenarios we consider in this work, by looking at the distribution of the estimation error and the rate of convergence of the underlying aggregation protocol, as well as the maximum rate of change of the aggregate. The parameter $k \leq 1$ specifies how close the aggregate needs to be to the threshold before a node turns active. If the aggregate changes slowly, larger values of k reduce the protocol overhead. For our simulations we use $k = 0.9$. The r_{wait} parameter specifies the interval (in number of rounds) between the crossing of the current threshold and the start of polling. Large values reduce the protocol overhead while increasing the detection delay of threshold crossings. For our evaluation, we use $r_{wait} = 4$. Finally, r_{poll} gives the duration (in number of rounds) of the polling phase. Larger values allow the estimate to converge closer to the true value at the expense of higher overhead. For the simulations, we use $r_{poll} = 6$.

Other Simulation Parameters. We run the simulations with the following parameters unless stated otherwise. The choices for the particular values are based on the configuration and measurements on our testbed, internet measurements, and the need for a sufficient number of measurement events to obtain statistically significant simulation results.

- protocol rate r: 4 rounds/sec
- processing delay: 1ms/message
- communication delay: 20ms
- length of a simulation: 1500sec, with an initialization period of 30sec
- threshold values: T_g^+ set at 1.05 times the average value of the aggregate and T_g^- set at the average value of the aggregate.

B. Protocol efficiency

We assess the efficiency of TG-GAP by measuring the protocol overhead in two scenarios, one in which several threshold crossings occur, and a second scenario in which the threshold is not crossed. In the first scenario, we run the protocol on the 654-node network graph where the local variables change according to the Periodic UT trace. The simulation is run for 75 sec, and Figure 3 shows the trace of the simulation after the initialization period.

Figure 3 shows the evolution of the aggregate and the protocol overhead. During the simulation run, three threshold

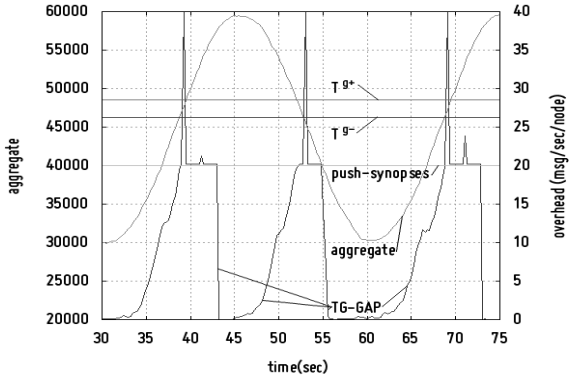

Fig. 3. TG-GAP and push-synopses: protocol overhead over time.

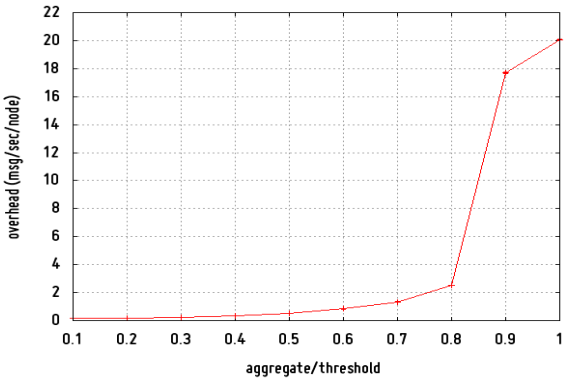

Fig. 4. Protocol overhead in function of the ratio of aggregate to threshold.

crossings occur: at around t=39sec (upper threshold crossing), t=53.5sec (lower threshold crossing) and t=69sec (upper threshold crossing). When the aggregate is close to the threshold (e.g., between 38sec and 43sec) we observe that the overhead of TG-GAP becomes equal to that of push-synopses, except for the spikes. The observation that both protocols have the same overhead can be explained by the fact that TG-GAP and push-synopses exchange the same number of messages when all nodes are active (which occurs when the aggregate is close to the threshold). Also, during this period, we observe spikes that are the results of the initialization of the snapshot to determine the global aggregate. During all other times, we see that the overhead is much lower than that of push-synopses because of the dynamic rate control feature described in section VI.

In the second scenario, we study the effect of the relative distance between aggregate and threshold on the protocol overhead. To do this, we use the UT trace and vary the threshold such that the ratio of (the average of) the aggregate to the (upper) threshold ranges from 10%-100%. Each simulation is run for 1500sec, and the average protocol overhead over this time is measured. Figure 4 shows the result.

As can be seen from Figure 4, the protocol overhead decreases with decreasing ratio. For ratios of 0.1 to 0.8, the overhead increases almost linearly with a small rate and stays below some 15% of the maximum. Above 0.8, the overhead quickly increases to that of push-synopses when the ratio reaches 1. (The overhead of push-synopses is 20.1

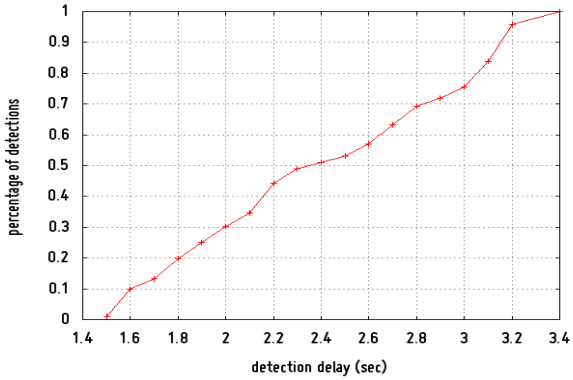

Fig. 5. Cumulative distribution of detection delays.

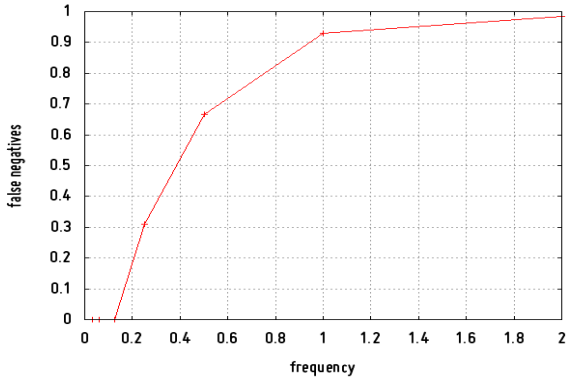

Fig. 6. False negatives (as the ratio of the number of missed crossings to the total number of crossings) in function of the frequency of threshold crossing (ν).

msg/sec/node for this scenario.)

The two scenarios above demonstrate that TG-GAP is efficient in the sense that the protocol overhead is small when the aggregate is far from the threshold. In our specific experiment, an aggregate of below 25% of the threshold results in a protocol overhead that is two orders of magnitude lower than that of push-synopses. Further, the results show that the overhead of TG-GAP is always below that of push-synopses (ignoring the overhead associated with the initialization of the snapshot).

C. Latency for threshold detection

In this scenario, we study the delays between the actual threshold crossing and its detection by TG-GAP. We simulate the protocol on the 654-nodes topology with the periodic UT trace. The protocol is run for 1500sec, resulting in 98 (49 upward and 49 downward) threshold crossings. The resulting delay distribution is shown in Figure 5.

The figure shows that, for this particular scenario, threshold crossings are detected within 1.5-3.4 sec. Note that since computing a snapshot takes 6 rounds (∼1.5sec), TCAs can not, in general, be correctly detected before this time. In this scenario no false positives or false negatived occurred.

The result from this scenario shows that all threshold crossings can be detected within 3.4sec in this network of 654 nodes, with a protocol rate of 4/sec.

D. Correctness of TG-GAP

We assess the correctness of TG-GAP by measuring the ratio of false positives to the total number of TCAs raised by the protocol and the ratio of false negatives to the total number of actual threshold crossings. We measure these metrics in function of the frequency of threshold crossings.

For this scenario, we run the protocol on the 654-node network graph. The local variables change according to the Periodic UT trace, with the frequency as the control parameter. For a node i, the local value changes as $w_i^*(t) = int\left(w_i(t) + 23\left(1 + \sin(2\pi t\nu)\right)\right)$, where values of ν are chosen from {1/32, 1/16, 1/8, 1/4, 1/2, 1, 2}. Note that small values of ν result in a low rate of threshold crossings, while large values result in a high rate of threshold crossings.

Size	diameter	Avg. inter-node distance
82	5	2.9
164	5	3.6
327	6	3.9
654	7	4.3
1308	7	4.8
2616	8	5.3
5232	8	5.8
10464	9	6.3
20928	9	6.9

TABLE II
TOPOLOGICAL PROPERTIES OF GOCAST-GENERATED TOPOLOGIES USED IN THE EXPERIMENTS (WITH CONNECTIVITY 10).

Figure 6 shows the result. Each point on the graph is the outcome of a simulation run. The plot shows only false negatives, as no false positives occurred for all values of ν.

As expected, the protocol correctly detects all threshold crossings for small frequencies (i.e., $\nu \leq 1/8$). However, above $\nu = 1/4$, the protocol misses threshold crossings. We know from the protocol design that, for a correct detection, there is a minimum detection delay of $\Delta = round_length * (r_{wait} + r_{poll}) = 2.5$sec (see Section VI). For an aggregate of the form used in this scenario, we expect that false negatives occur when $\Delta > \frac{1}{2\nu}$, which means for our experiment $\nu > 0.2$. This reasoning is consistent with the graph in Figure 6.

Regarding false negatives, the results show that false negatives can be avoided, if the frequency of threshold crossings is below a certain limit, which is dependent on the minimum detection delay of the protocol. Regarding false positives, the results here show that for the correct choice of the protocol parameter r_{poll}, false positives can be avoided. (Recall that false positives occur only due to an estimation error in the computed snapshot.)

E. Scalability

We study the protocol in two scenarios, where we measure the protocol overhead and the detection delay as a function of the network size.

In the first scenario, we use GoCast-generated graphs with a target connectivity of 10 for networks of size 82, 164, 327, 654, 1308, 2626, 5232, 10464 and 20928. Table II shows

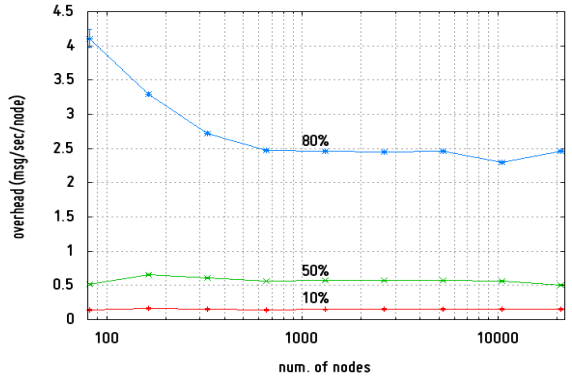

Fig. 7. Protocol overhead in function of network size for different ratios of aggregate/threshold.

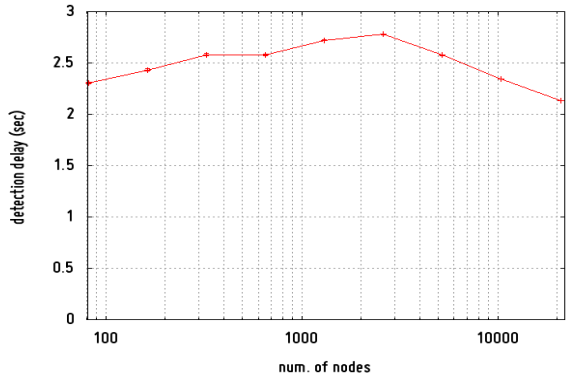

Fig. 8. Detection delay in function of network size.

topological properties of these graphs. We use the UT trace to simulate the behavior of the local variables. For each topology, the threshold is selected such that the ratio of the average of the aggregate to the threshold is 10%, 50% and 80%, respectively. The result is shown in Figure 7. Each point on the graph is the outcome of a simulation run.

The figure shows no visible dependence of the protocol overhead on the system size when the aggregate is far from the threshold. When the aggregate is close to the threshold, the protocol overhead seems to decrease with the system size. (This is an interesting effect that we did not expect and cannot explain.)

In the second scenario, we measure the average detection delay as a function of the network size. Again, we use the topologies given in Table II. The local variables are simulated using the periodic UT trace. The result is shown in Figure 8. Each point on the graph is the outcome of a simulation run.

The figure suggests that the detection delay in TG-GAP is by and large independent of the system size. (Note that in all these experiments, no false positives and false negatives occurred.)

We explain this behavior by the way we constructed the local traces for the simulation which results in all parts of the network exhibiting similar statistical behavior. (We experienced a similar behavior in our earlier work where we evaluated the accuracy of a gossip-based aggregation protocol as the function of the system size and used the same way of constructing traces [2].)

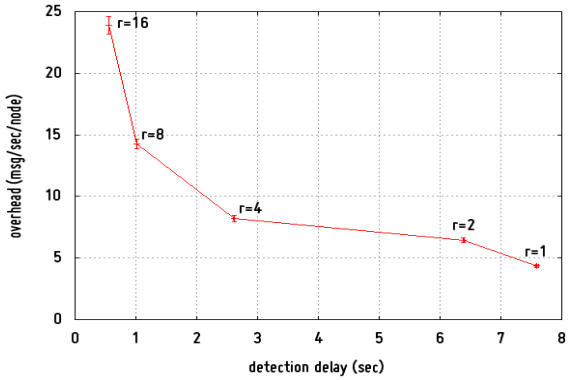

Fig. 9. r controls the trade-off between protocol overhead and detection delay.

In summary, for the scenarios considered here, TG-GAP is scalable in the sense that its overhead and detection delay are largely independent on the system size.

F. Controlling trade-off between overhead and detection delay

In this subsection, we study the controllability of TG-GAP. Specifically, we are interested in the extent to which we can control the trade-off between overhead and detection delay. We use the protocol rate r as the control parameter.

We measure the average protocol overhead and the average detection delay as functions of r, varying r from 1 to 16. (Values of r smaller than 1 would result in false negatives as explained above.) We use the default parameters for the simulations and the result is shown in Figure 9.

As expected, the protocol overhead increases with r, while the detection delay decreases. The figure shows that r is an effective control parameter, because the changes in the overhead are significant for the range within which r was varied. (In [10], we demonstrated that the parameter k, which determines the switch between active and passive state of a node, can also be used as a control parameter for the same purpose.)

G. Robustness

Robustness of TG-GAP is studied in three scenarios: first, the protocol overhead is measured as a function of the rate of node failures. Then, for a fixed failure rate, we measure the distribution of the delay in detecting threshold crossings. Finally, we evaluate the accuracy of the protocol under node failures. For the scenarios in this section, TG-GAP is extended, in a similar way as described in [2], to ensure that the underlying aggregation process performs correctly under failures.

For the first scenario, we use the default 654-node topology, and we simulate local variables with the UT trace. The threshold is chosen five times the average value of the aggregate during the run. For failure rates of 0.039, 0.156, 0.625, 2.5 and 10 failure/sec/network, we measure the protocol overhead. The result is shown in Figure 10. Each point on the graph is the outcome of a simulation run.

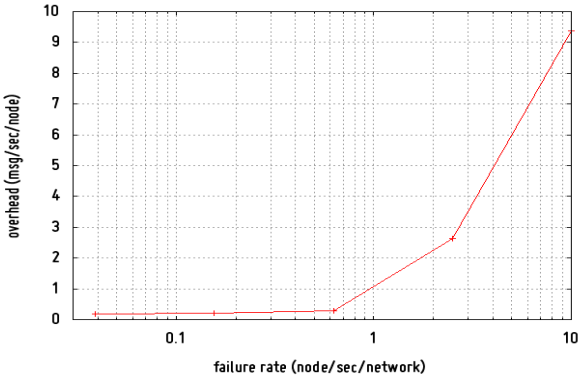

Fig. 10. Protocol overhead in function of the failure rate for a 654-node network.

Fig. 11. Cumulative distribution of detection delays for 0.625 failure/sec/network and no failures.

As can be seen from the figure, the protocol overhead increases with the failure rate. For rates less than 0.625/sec, the overhead of TG-GAP is similar to that for the case where no failures occur. However, for larger failure rates, the protocol overhead quickly increases. We explain this behavior by the fact that failure recovery introduces a temporary local bias, which in turn, increases probability of nodes starting a snapshot.

For the second scenario, we use the default parameters with the Periodic UT trace. We produce a simulation run with 0.625 failure/sec/network and one with no failures. Figure 11 shows distribution of the detection delays for these two runs.

As can be seen from the figure, the scenario with failures shows a higher variance of detection delays compared to that with no failures.

In the last scenario, we run simulations with failure rates of 0.039, 0.156, 0.625, 1, 2.5 and 10 failure/sec in the network, with the same setting as in the experiment above. For each simulation run, we assess the correctness of the protocol by determining the number of false negatives and false positives. The result is shown in Figure 12.

The figure shows that the protocol exhibits no false positives for failure rates of less than 0.125/sec. For larger failure rates, the rate of false positives increase to about 50% for a failure rate of 10/sec. During the simulation run the protocol exhibited no false negatives. Note that, in practice, false

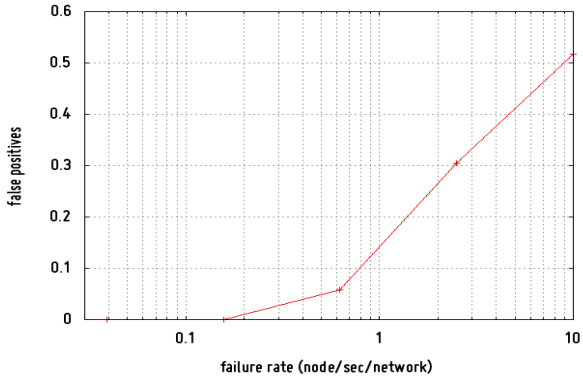

Fig. 12. False positives (as a ratio to the total number of raised alarms) in function of the failure rate for a 654-node network.

negatives are more serious than false positives, since with false positives, further checks can be performed to verify the threshold crossing, while with false negatives this is not possible.

The results in this subsection suggest that TG-GAP is robust to node failures in the following sense. There is a failure-rate limit up to which the protocol (1) exhibits an overhead comparable to when no failures occur and (2) exhibits neither false positives nor false negatives. For the scenarios investigated, this limit is 0.125 failures/sec/network.

H. Comparison against TCA-GAP

In this subsection, we compare the performance of TG-GAP with that of TCA-GAP, a tree-based protocol for detection of threshold crossings that we developed in earlier work [8]. For the same maximum number of protocol messages per link, we compare both protocols with regard to protocol efficiency, quality of detection, scalability and robustness. An in-depth description and evaluation of TCA-GAP can be found in [8]. For TG-GAP, we take the same measurement results as discussed in the subsections above, and we run TCA-GAP for the same scenarios and with the same settings described above.

1) Efficiency: Figure 13 shows the average protocol overhead for different ratios of aggregate to threshold, for the scenario of Figure 4. The qualitative behavior of both protocols is similar: they exhibit a low overhead when the aggregate is far from the threshold, and the overhead becomes comparable to that of the underlying aggregation protocol when the aggregate is close to the threshold.

Quantitatively however, the overhead of the two protocols is significantly different, since they are based on different aggregation protocols with very different performance characteristics. In fact, the overhead of TG-GAP is larger than that of TCA-GAP by a factor of at least 10 in most cases. TCA-GAP generates a lower overhead than TG-GAP, since TCA-GAP exchanges messages reactively and only on a subset of overlay links, namely on those links that are part of the aggregation tree, while in TG-GAP messages are exchanged on all overlay links.

2) Quality of detection: The distribution of the detection delays, for the same scenario as that of Figure 5, is shown

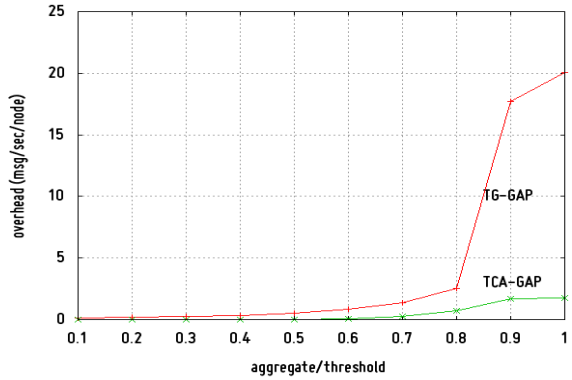

Fig. 13. Protocol overhead in function of average ratio of aggregate to threshold for TG-GAP and TCA-GAP.

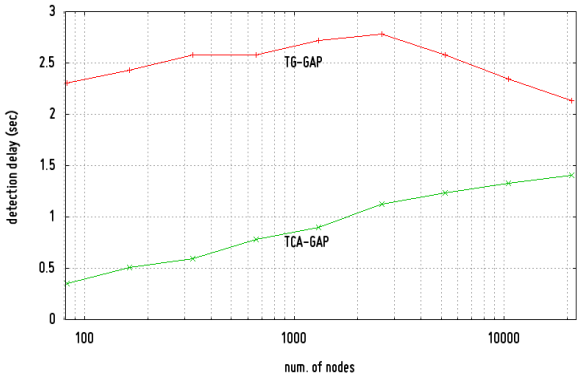

Fig. 15. Detection delay in function of network size for TG-GAP and TCA-GAP.

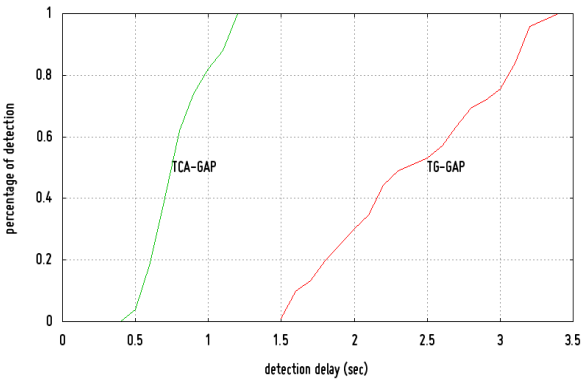

Fig. 14. Cumulative distribution of detection delays for TG-GAP and TCA-GAP.

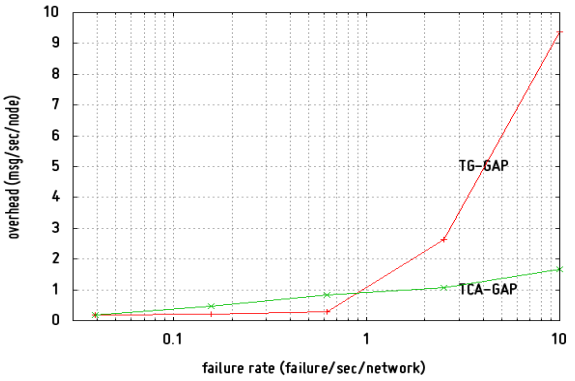

Fig. 16. Protocol overhead in function of failure rate for TG-GAP and TCA-GAP.

in Figure 14. With regards to detection delay, TCA-GAP performs generally better than TG-GAP, although the differences are not as drastic as in the above experiment. First, TCA-GAP has clearly a smaller detection delay than TG-GAP. The larger delay of TG-GAP is the consequence of the protocol design where nodes must wait for the completion of a snapshot before they can issue an alert. Second, TG-GAP has a larger variance in detection delays than TCA-GAP.

3) Scalability: As the discussion above shows for TG-GAP and the measurement results in [8] demonstrate for TCA-GAP, both protocols exhibit an overhead that is seemingly independent of the system size, for the scenario settings and the parameter space investigated.

However, as investigated in the scenario of Figure 8, detection delays seem to increase with the logarithm of the system size in the case of TCA-GAP, while the delays seem to be independent of the system size in the case of TG-GAP (see Figure 15). This figure further suggests that, in terms of detection delay, TG-GAP potentially outperforms TCA-GAP for very large networks. (Obtaining measurement results for network sizes much larger than those reported in this work would have been difficult for us, due to the limited computational resources at our disposal.)

4) Robustness: Figure 16 shows the protocol overhead in TCA-GAP and TG-GAP for the scenarios of Figure 10. The protocol overhead for TCA-GAP seems to increase with the logarithm of the failure rate, while for TG-GAP the increase seems to be linear (note that the x-axis is in log-scale). As further measurements show, for failure rates above 2.5/sec, both protocols exhibit large rates of false positives which would render them unsuitable for practical settings. Below that rate, the overhead of both protocols is roughly comparable.

Based on the above experiments, our comparative assessment of TG-GAP vs TCA-GAP is as follows. For scenarios with no failures and up to 20,000 nodes, TG-GAP incurs an overhead of at least one order of magnitude larger than TCA-GAP. In failure scenarios, the overheads of both protocols are roughly comparable. TG-GAP has a somewhat larger detection delay than TCA-GAP for networks below 20,000 nodes.

VIII. Discussion and Conclusion

In this work, we investigate the use of gossip protocols for the detection of network-wide threshold crossings. Our design goals for such a protocol are efficiency with respect to protocol overhead, high quality for detection of threshold crossings (i.e., small detection delay and low probability of false positives and negatives), scalability of key metrics with the system size, robustness to node failures and controllability of the trade-off between overhead and quality of detection. Based on push-synopses, we introduce a simple protocol that indicates whether a global aggregate of static local values is above or below a given threshold. For this protocol, we prove

correctness and we prove that the protocol converges to a state with no overhead when the aggregate is sufficiently far from the threshold. Second, we introduce TG-GAP, an extension of the simple protocol that executes in a dynamic network environment and implements the hysteresis behavior. Key elements of the design are the construction of a snapshot of the global aggregate for threshold detection and a mechanism for synchronizing the local direction of threshold detection, both of which are realized using the underlying gossip protocol. Through simulations, we evaluate the performance of TG-GAP against our design goals, and performed a comparative evaluation against a tree-based protocol for detecting global threshold crossings.

The evaluation results indicate that TG-GAP meets our design goals. Regarding efficiency, the protocol has a low overhead when the aggregate is far from the threshold, and, when the aggregate is close to the threshold, the overhead becomes comparable to that of the underlying aggregation protocol. In our experiment, an aggregate that averages below 25% of the threshold results in a protocol overhead that is two orders of magnitude lower than that of push-synopses.

Regarding correctness, for a scenario with 654 nodes where no failures occur, all threshold crossings are correctly detected, for all cases where less than 7 threshold crossings occur per minute. For all the scenarios considered with the default protocol rate, for network sizes of up to 20,000 nodes, threshold crossings are detected within 3sec of their occurrence.

With respect to scalability, the protocol overhead and detection delays seem to be independent on the system size for the specific overlay and traces used in the simulations.

Regarding robustness, the protocol, based on a robust gossip-based aggregation protocol, correctly detects all threshold crossings in scenarios where the failure rates are below 8 failures/minute for a 654 node network. Finally, regarding controllability, the trade-off between protocol overhead and detection delay can be effectively controlled through the protocol rate.

Lastly, we perform a comparative evaluation of TG-GAP against TCA-GAP, a tree-based protocol for detecting global threshold crossings. For scenarios with no failures and up to 20,000 nodes, TG-GAP incurs an overhead of at least one order of magnitude larger than TCA-GAP. In failure scenarios, the overheads of both protocols are roughly comparable. TG-GAP has a somewhat larger detection delay than TCA-GAP for networks below 20,000 nodes. For systems of much larger size, TG-GAP may have a significantly smaller detection delay than TCA-GAP.

In conclusion, we learned that, for detecting global threshold crossings in the type of scenarios investigated in this work, a tree-based protocol incurs a significantly lower overhead and a smaller detection delay than a gossip protocol such as TG-GAP. This finding is consistent with our earlier result in [2], where we conclude from extensive simulation studies that a tree-based aggregation protocol consistently shows a higher estimation accuracy than a gossip protocol (a robust extension of push-synopses), for a comparable protocol overhead.

REFERENCES

[1] K. Birman, "The promise, and limitations, of gossip protocols," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 8–13, 2007.

[2] F. Wuhib, M. Dam, R. Stadler, and A. Clemm, "Robust monitoring of network-wide aggregates through gossiping," *Network and Service Management, IEEE Transactions on*, vol. 6, June 2009.

[3] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *PODC '87: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, (New York, NY, USA), pp. 1–12, ACM Press, 1987.

[4] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *FOCS '03: Proceedings of the 44^{th} Annual IEEE Symposium on Foundations of Computer Science*, p. 482, IEEE Computer Society, 2003.

[5] C. Tang and C. Ward, "Gocast: Gossip-enhanced overlay multicast for fast and dependable group communication," in *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, (Washington, DC, USA), pp. 140–149, IEEE Computer Society, 2005.

[6] A. Fernandez, V. Gramoli, E. Jimenez, A.-M. Kermarrec, and M. Raynal, "Distributed slicing in dynamic systems," in *ICDCS '07: 27th International Conference on Distributed Computing Systems*, (Los Alamitos, CA, USA), p. 66, IEEE Computer Society, 2007.

[7] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *NCA '05: Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pp. 41–48, IEEE Computer Society, 2005.

[8] F. Wuhib, M. Dam, and R. Stadler, "Decentralized detection of global threshold crossings using aggregation trees," *Computer Networks*, vol. 52, no. 9, pp. 1745–1761, 2008.

[9] D. Breitgand, D. Dolev, and D. Raz, "Accounting mechanism for membership size-dependent pricing of multicast traffic.," in *NGC '03: Networked Group Communication*, pp. 276–286, 2003.

[10] F. Wuhib, M. Dam, and R. Stadler, "Gossiping for threshold detection," in *IM '09: 11^{th} IFIP/IEEE International Symposium on Integrated Network Management*, (Long Island, USA), 2009.

[11] M. Dilman and D. Raz, "Efficient reactive monitoring," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 20, no. 4, pp. 668–676, 2002.

[12] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pp. 289–300, ACM Press, 2006.

[13] L. Huang, M. Garofalakis, J. Hellerstein, A. Joseph, and N. Taft, "Toward sophisticated detection with distributed triggers," in *MineNet '06: Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, pp. 311–316, ACM Press, 2006.

[14] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," in *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pp. 301–312, ACM Press, 2006.

[15] S. Y. Ko, I. Gupta, and Y. Jo, "A new class of nature-inspired algorithms for self-adaptive peer-to-peer computing," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, no. 3, pp. 1–34, 2008.

[16] D. Kempe and J. M. Kleinberg, "Protocols and impossibility results for gossip-based communication mechanisms," in *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, (Washington, DC, USA), pp. 471–480, IEEE Computer Society, 2002.

[17] A. Olshevsky and J. N. Tsitsiklis, "Convergence rates in distributed consensus averaging," in *CDC '06: Proceedings of the 45^{th} IEEE Conference on Decision and Control*, (Washington, DC, USA), pp. 3387–3392, IEEE Computer Society, 2006.

[18] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulieacute;, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[19] K. S. Lim and R. Stadler, "SIMPSON — a SIMple Pattern Simulator fOr Networks." http://www.s3.kth.se/lcn/software/, 2009.

[20] U. of Twente, "Traffic measurement data repository." http://m2c-a.cs.utwente.nl/repository/, 2006.

[21] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *SIGCOMM '02: Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 133–145, ACM Press, 2002.

Fetahi Wuhib (http://www.ee.kth.se/˜fzwuhib) is a Ph.D. candidate at the Royal Institute of Technology (KTH), Stockholm, Sweden. He received his B.Sc. degree in electrical engineering from Addis Ababa University, Ethiopia, his M.Sc. degree in internetworking and his Licentiate of Technology degree in telecommunications from KTH. His current research focuses on distributed protocols for network monitoring and configuration.

Mads Dam (http://www.csc.kth.se/˜mfd) is associate professor at the School of Computer Science and Communication, the Royal Institute of Technology (KTH), in Stockholm, Sweden, since 1998. Dam received his B.Sc. in information technology in 1983, and his M.Sc. in computer engineering in 1985 from Aalborg University, Denmark. In 1990 Dam received his Ph.D. in computer science from the University of Edinburgh, U.K., where he remained as a postdoctoral researcher until 1992 when he moved to the Swedish Institute of Computer Science (SICS) where he became head of the Formal Design Techniques Laboratory. Dam has made significant contributions in areas such as modal and temporal logics, process algebra, programming languages and program logics, computer security, and, most recently, autonomic computing and network management.

Rolf Stadler (http://www.ee.kth.se/˜stadler) is a professor at the Royal Institute of Technology (KTH), Stockholm, Sweden. He holds an M.Sc. degree in mathematics and a Ph.D. in computer science, both from the University of Zurich. In 1991 he was a researcher at the IBM Zurich Research Laboratory. From 1992 to 1994 he was a visiting scholar at Columbia University, which he joined in 1994 as a research scientist. From 1998 to 1999 he was a visiting professor at ETH Zurich. He joined the faculty of KTH in 2001.