# Newscast Computing

Márk Jelasity, Wojtek Kowalczyk and Maarten van Steen

***Abstract*— Monitoring large computer networks often involves *aggregation* of various sorts of data that are distributed across network components. Finding extreme values, counting discrete observations or computing an average or a sum of some parameter values are typical examples of such "background" activities that provide input to monitoring systems. Another aspect of network management is fast and reliable *information dissemination*, like propagation of alarm signals.**

**We present a novel approach to information aggregation and dissemination. It is based on a concept of a highly distributed, anonymous, democratic and non-deterministic form of collaborative information processing: *newscast computing*. The main properties of this approach are scalability, robustness, adaptivity, and speed. The underlying protocol is very simple and can be implemented and run on huge networks of small computing devices, such as mobile phones, PDA's, sensors, etc.**

**The usefulness of the newscast approach is illustrated by two algorithms for finding the maximum and the average of values that are distributed along the nodes of a network. The algorithms are formally proven to converge exponentially fast and numerous simulation experiments provide additional insights into their behavior. Finally, we demonstrate their applicability to several network monitoring tasks: finding the size of a network, counting nodes that left or joined the network, system load estimation, and fast propagation of alarms.**

## I. Introduction

The Internet has been exponentially growing from the start. This trend is not likely to stop soon, partly due to recent developments in wireless technology and the IPv6 standard, which will enable virtually all devices with a digital heartbeat to go online. The Internet is now forming a platform for a wide variety of applications ranging from entertainment to business. It can be expected that the number and importance of such distributed applications will grow with time as the Internet is penetrating more and more into our lives.

Managing and controlling such applications poses new challenges for computer science. The Internet is heterogeneous, unreliable, huge, and dynamic. Controlling applications by human administrators will gradually become impossible. The situation is not unlike the difference between flying a small plane and the space shuttle. The latter cannot be flown by humans at all anymore due to the complexity of the task.

Solving the problem of control and monitoring (analysis) in such an environment is a grand scientific challenge. Working towards this goal, in this paper we are proposing a method for a specific task: aggregating information in a *large* and *highly dynamic* distributed environment in a robust, dependable manner. By aggregation we mean finding statistics over a set of numeric values that are distributed over a wide-area application, like extremal values, average, sum, count, variance, etc. In particular, in this paper we will emphasize the applicability of these aggregated statistics for network monitoring.

In the context of network monitoring, our application model is the following. The distributed environment is a network consisting of a possibly huge number of *nodes*. All nodes have *attributes*, like memory size, disk size, CPU speed, network bandwidth, current load, etc. The network is dynamic. Nodes can join and leave, and furthermore their attribute values can also change.

Our solution to finding aggregates of attribute values of nodes is based on an epidemic protocol we have recently developed. The protocol is called *newscast*. Newscast is a so-called peer-to-peer protocol. All nodes are equivalent and run the same algorithm. The purpose of the protocol is to maintain and disseminate *up-to-date* information in a robust, self-organizing way, without any central intervention, in a possibly dynamically changing and large-scale environment. The basic underlying idea is that all nodes periodically exchange information with each other including membership information (addresses of other nodes) and application specific information. Furthermore, each piece of information gets a timestamp when created which is used to remove outdated items.

In this paper we will describe the newscast protocol and apply it to calculate aggregates that are useful for network monitoring. In Section II we discuss related work. The purpose of Section III is to give an intuitive introduction to the key ideas of the paper, in particular the possibility of the application of *epidemics* and *diffusion* for calculating aggregates. These ideas are elaborated upon in other sections. Section IV introduces the newscast protocol from an algorithmic point of view, followed by Section V which demonstrates the robustness and scalability of the protocol. Building on newscast, Sections VII and VIII elaborate on the ideas introduced in Section III with theoretical analysis and simulation results. Various applications of newscast computing to network monitoring and control are presented in Section IX. Section X concludes the paper.

## II. Related Work

Our approach to network monitoring is related to several sub-areas of distributed computing. They include the following.

*a) Epidemic protocols:* Epidemic protocols are becoming more and more popular since the publication of the seminal paper by Demers et al. [1]. A recently completed survey by Eugster et al. provides an excellent introduction to the

This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923)

M. Jelasity is with the University of Bologna

W. Kowalczyk and M. van Steen are with the Vrije Universiteit Amsterdam

field [2]. Epidemic algorithms have been applied to solving several practical problems like database replication [1], failure detection [3] and resource monitoring [4]. A large body of theoretical work is also available due to the general importance of understanding epidemics [5] and its close relation to random graph theory [6].

*b) Self-organizing topology management:* There are countless protocols for managing different kinds of topologies in an adaptive fashion. Here we focus only on those that maintain a random (or unstructured) topology. The most well-know examples are SCAMP [7], lpbcast [8], a method to build random expander graphs [9], [10]. Newscast is targeted at extremely large and dynamic environments. In its approach and scope, the closest to newscast is lpbcast. The lpbcast protocol is also based on epidemic-style information dissemination, it is also proactive and maintains a regular random topology. However, the resulting communication graph shows rather significant differences from that of newscast. This difference is important because the communication graph defines crucial constraints on possible applications.

*c) Aggregation in distributed environments:* The field of distributed computation of aggregates is less established than epidemic protocols. An overview of the problem can be found in [11].

A prominent approach is Astrolabe [4] which is a hierarchical architecture for aggregation in large distributed systems. Our approach is substantially different in that it is extremely simple, lightweight, and aimed at unstructured, highly dynamic environments. In the case of our protocol the overhead of installation and maintenance is virtually negligible. A related work is [12] which is also based on a hierarchical approach. While building hierarchies indeed reduces the cost of finding the aggregates, it introduces additional overhead having to maintain this hierarchical topology in a dynamic distributed environment. Moreover, due to being hierarchical, it also needs extra effort and protocols to broadcast the result continuously over the network if all nodes need to know the result continuously.

Another recent work [13] discusses many approaches, based on spanning tree induction and using other, more redundant topologies. While being the closest to our approach, a main difference is that the protocols described there are *reactive*: aggregation is initialized from a certain point and the result is known by only that node. This makes it hard to adopt for solving our present research problem, continuous network monitoring, similarly to the other approaches mentioned above.

*d) Network Monitoring:* Network and systems monitoring has since long been part of management architectures that tend to be large, complex, and difficult to scale across wide-area systems (see, for example, [14]). Recently, new insights have led to completely decentralized monitoring solutions, often involving mobile agents [15]. Also the research into scalable event-notification systems that deploy peer-to-peer technology is highly relevant to our work.

Siena is arguably one of the first large-scale event-notification systems, which effectively applies a combination of multicasting and content-based routing to efficiently notify events to interested parties [16]. However, approaches such as followed in Siena require that a network of servers is first installed before application-level multicasting can be deployed.

In this respect, more interesting is Scribe [17]. Scribe is an application-level multicasting system that is built on top of Pastry, a structured peer-to-peer network [18]. Scribe allows the formation of topic-based publish/subscribe groups in a fully dynamic and decentralized fashion.

A step further in this direction is taken in PeerCQ [19]. It is a general-purpose information monitoring system in which a client can formulate continual queries, that is, queries related to possibly continuous changes of data over time. The key idea behind PeerCQ is to let an arbitrary peer monitor the data and machines involved in a single query. Because each query has a unique identifier, this scheme fits nicely with the identifier-based routing protocols inherent to structured peer-to-peer systems. A drawback of this approach is that there may be many peers monitoring the same data or machines, in turn introducing a potential scalability bottleneck.

Scalable liveness detection has very recently been proposed in [20]. The essence of the proposed protocol is to offload the number of probes for a single device to its probing processes. The latter are dynamically organized into an overlay network by letting the device, on each probe, return the addresses of the last $N$ processes that probed it as well. This approach will permit a probing process to reduce its probing frequency, because it will be informed by one its peers whenever the status of the monitored device changes.

```
do forever {
    e = waitForEvent();
    if e is TIMEOUT {
        n_j = randomPeer()
        send x_i to n_j
        receive x_j from n_j
        x_i = aggregate(x_i, x_j)
    }
    if e is message x_j from n_j {
        send x_i to n_j
        x_i = aggregate(x_i, x_j)
    }
}
```

Fig. 1. The protocol run by each peer. It is assumed that a timer generates a `TIMEOUT` event regularly in every $\Delta T$ time units.

## III. Distributed Aggregation: the Basic Idea

As already mentioned, we will base our aggregation algorithms on the newscast protocol. In this section, however, we focus only on the basic idea of our approach to aggregation, abstracting away from newscast.

Consider a huge collection of nodes, where each node $n_i$ maintains a single number $x_i$. The goal is to let nodes compute some aggregate of these values in a fully decentralized way. The algorithm in Figure 1 is at the heart of our aggregation approach and is executed by each peer. We assume that a timer generates a `TIMEOUT` event every $\Delta T$ time units. When this happens, a peer selects another peer at random,

sends its current value, and waits for the response. Its own value is then updated by aggregating it with the response. In this light, whenever a node is contacted by another peer, it simply computes the aggregate of its own and the incoming value. We will focus on two implementations of the function `aggregate()`: average and maximum.

*e) Average:* In this case $\texttt{aggregate}(x_i, x_j) = (x_i + x_j)/2$. We will show that for each node $n_i$, $x_i$ converges exponentially fast to the overall average, provided we guarantee that the peers are indeed selected (more or less) randomly. Note that the algorithm results in *diffusion-like* dynamics over a random topology. Thinking of the values as being the concentration of some substance, the elementary averaging step is indeed equivalent to the equalization of the concentrations at the two locations. Motivated by this analogy, we will call this approach *diffusion-based aggregation*.

Being able to compute averages in a fully decentralized manner is by itself interesting. However, with an appropriate choice of the semantics of the values $x_i$, it is possible to calculate the network size, the sum of values, the variance, etc. For example, if initially exactly one node holds the value 1 and all the others hold 0, then it is obvious that eventually each node will be able to estimate the size of the network which is $1/\overline{x}$.

*f) Maximum:* In this case $\texttt{aggregate}(x_i, x_j) = \max\{x_i, x_j\}$. We will show that for each node, $x_i$ converges super-exponentially fast to the overall maximum, again, provided we guarantee that the peers are selected randomly.

To understand the dynamics better, consider that no matter which node holds the value which turns out to be the maximum, the speed of dissemination of the true maximum value is exactly the same as if it was effectively broadcast via a push-pull epidemic protocol. Motivated by this observation, we will call this approach *epidemic-based aggregation*.

## IV. The Newscast Protocol

We shall call the basic entities which run the newscast protocol at each network node *correspondents*. The motivation for this choice will be clarified later.

The newscast protocol is responsible for two functions at the same time. The first is maintaining a possibly very large group of correspondents taking care of joining and leaving members and failures (membership management). Second, it is responsible for a special form of information dissemination among the group members, which we call *newscasting*. The protocol is extremely simple: each correspondent knows only a (continuously changing) small set of peers of which one is randomly chosen to exchange information. The newscast protocol is fully distributed and symmetric: the algorithm run by all correspondents is completely identical.

The newscast protocol is not an application by itself, it provides only membership and information dissemination services to applications. It is crucial to understand how an application and newscast cooperate, in other words, understand the interface between an application and newscast.

Obviously, newscast supports distributed applications. From now on we will refer to the component of the application running at a fixed network node as an *agent*. Note that it is not assumed that all agents of an application run identical algorithms. We consider the interface between the correspondent and the agent running at the same node. Without loss of generality we assume that each agent has exactly one correspondent attached to it.

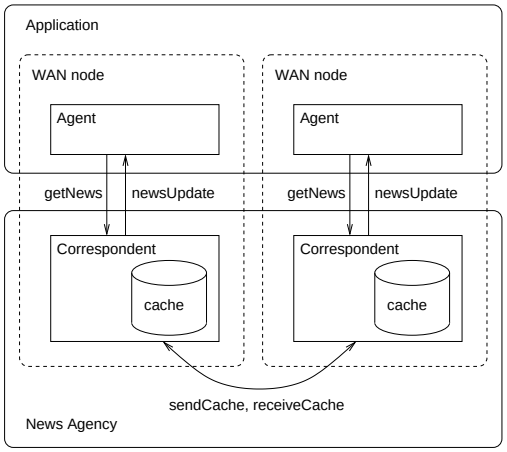



Fig. 2. The conceptual organization of a newscast application.

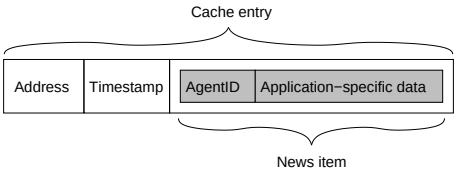



Fig. 3. The format of news items and cache entries.

The interface consists of two callback functions that have to be implemented by all agents. The correspondent asks the agent regularly for *news* by means of the callback function `getNews()`. In addition, the correspondent provides the agent with news collected by peer correspondents from other agents of the application through the callback function `newsUpdate(news[])`. The architecture is illustrated in Figure 2.

The definition of what counts as news is application dependent. The agents simply live their lives (perform computations, listen to sensors and the news, etc.) and based on the computations they have completed and the information they have collected they must provide the correspondent with news when asked.

### A. Principal Operation

Each correspondent maintains a *fixed-sized* cache of $c$ news items, where $c$ is a parameter. Whenever an agent passes a news item to its correspondent, the latter timestamps the item, adds its own network address, and subsequently caches the item. A news item itself consists of an agent identifier and the actual news as provided by the agent, as shown in Figure 3.

Correspondents regularly exchange caches as follows. Each correspondent executes the following five steps once every $\Delta T$ time units.

1) Request a fresh news item from the local agent by calling `getNews()`. Create a new cache entry and insert it into the cache.
2) Randomly select an accessible peer correspondent using the network addresses of other correspondents as found in the cache and ask this peer to exchange information.
3) Send all cache entries to the selected peer, and, in turn, receive all the peer's cache entries (at most $c$ items).
4) Pass the received cache entries from the peer agent to the local agent by calling `newsUpdate()`.
5) Merge the received entries into the local cache.
6) The correspondent now has at most $2c+1$ cache entries; it subsequently throws away the oldest ones to keep the $c$ freshest ones.

The merge and insert operations of the cache ensure that from one agent there is at most one item in the cache after the operation, the one with the freshest timestamp.

The selected peer correspondent (the passive party) executes the same algorithm except, of course, the peer selection step.

We call $\Delta T$ the *cycle length*. Even though the system is not synchronized, it is often convenient to talk about *cycles* of the protocol, which are simply consecutive wall clock time intervals of length $\Delta T$ counted from some convenient starting point.

The protocol does not require that the clocks of correspondents are synchronized, but only that the timestamps of news items in a single cache are mutually consistent. This can be achieved as follows. When a correspondent $A$ passes its cache to $B$, it also sends along its current local time, $T_A$. When $B$ receives the cache entries, it subsequently adds to the timestamp of each entry the value $T_B - T_A$, effectively normalizing the time of each new entry to those already cached. We assume that the communication time between two correspondents is smaller than $\Delta T$. This method introduces errors but exact synchronization is not necessary, so this solution fully suffices.

Note that if $\Delta T$ is chosen to be so small that this assumption is violated then newscast will not be able to work correctly anyway since in each cycle at least one communication has to be completed according to the protocol.

### B. Membership Management

The newscasting protocol disseminates correspondent addresses together with news items submitted by the agents. This automatically provides us with membership management functionality.

Subscriptions do not need any special sequence of communications, the new correspondent simply has to initialize its cache with at least one known correspondent which is already a member of the group, and start to execute the protocol. In Section V we will see that the system is not sensitive to subscription patterns and tolerates the worst case when each new member subscribes through the same fixed correspondent.

Unsubscriptions are treated as failures. An unsubscribing correspondent simply has to stop communicating. Outdated information is quickly removed from the system so if a correspondent does not keep communicating, it will be forgotten.

### C. Newscasting is not Broadcasting

It is important to note that newscasting is different from epidemic broadcasting or flooding, a difference that is easily overlooked and which may lead to confusion. A first observation is that newscasting is proactive, that is, it is not initiated by a single node, nor will it ever end. Second, unlike broadcasting, newscasting disseminates a given news item only to a random, relatively small group of peers. This limited dissemination is caused by the fact that, eventually, a news item is removed from a cache in favor of a fresher item.

These observations do not imply that newscasting cannot be deployed for broadcasting purposes. For example, a naive broadcasting scheme is to have an agent repeatedly return the same news when it is called back through `getNews()`. This scheme, however, will only slowly propagate news to all agents. A much better scheme is to let other agents store, and subsequently forward an incoming news item when requested for fresh news. This store-and-forward scheme effectively mimics a flooding algorithm. A more sophisticated solution is to deploy constrained forwarding in order to avoid that nodes receive too many duplicates. These alternatives are discussed later.

These examples illustrate the flexibility of newscasting which allows the implementation of a wide range of communication mechanisms and computations; a flexibility we use for computing aggregates as we explain later.

## V. Protocol Analysis

We would like the newscast protocol to have the following (slightly interrelated) properties:

1) *Self-organizing:* In the presence of very different patterns by which nodes join and (un)intentionally leave the system, it should continue to properly operate without manual intervention.
2) *Effective:* Information should be disseminated to each member in a fast and predictable manner.
3) *Scalable:* The same quality of service should be provided at acceptable costs even if the group of members is very large.
4) *Robust:* The system should tolerate severe damage, such as a massive failure of nodes.

In this section we will present empirical evidence that the newscast protocol indeed has the properties listed above.

It is important to stress that our approach to design was not developing a complex and sophisticated protocol that can provably achieve a pre-determined list of properties. Instead, we opted for keeping the design as simple as possible and analyzing its behavior afterwards as if it was a biological or physical system. In other words, our goal is to explore the power of *emergence*. There are no dedicated, specifically designed mechanisms to, for example, recover the network after damage, or to react to growing network size or increasing unreliability. The protocol performs the same simple operation irrespective of the circumstances and we will argue that it still reacts appropriately in a wide variety of settings.

In most experiments we will consider the communication graph that is defined by the set of participating correspondents

(the nodes of the graph) and the addresses of peer correspondents in their caches (targets of directed edges). In other words, there is a directed edge from correspondent $c_1$ to $c_2$ if and only if the address of $c_2$ is contained in the cache of $c_1$. This graph is extremely important since robustness and the effectivity of information dissemination depend on its properties. We will denote this graph in cycle $t$ by $D_t$. This graph could also be called the "knows-about" graph, because the directed edges are defined by peers the correspondents know about.

The graph $D_t$ is $c$-regular, that is, all correspondents have cache size $c$ and therefore $c$ outgoing edges (provided the number of correspondents is larger than $c$). In the following we will denote the number of correspondents by $N$.

Instead of examining the "knows-about" graph, we are more interested in the "can-communicate-with" graph, which we get by simply dropping the orientation of the edges in $D_t$. This is motivated by the fact that the information exchange performed by newscast is symmetric, it does not matter which party initiated the connection. We will denote this undirected version of the communication graph by $G_t$. Unlike $D_t$, $G_t$ is not necessarily regular but the degree of each node is at least $c$.

### A. Bootstrapping and Dynamic Environments

Newscast is a protocol designed for dynamic environments, and is itself dynamic even if the environment remains the same. The communication graph is constantly changing. We are interested in whether important properties of the graph converge to a fixed value from a wide variety of starting conditions, that is, is there a stable equilibrium with respect to that property? In other words, is *self-organization* observable? In this section we focus on *average path length*. This value is defined by averaging the minimal path lengths between all pairs of nodes. This property is important because it is directly related to the efficiency of information dissemination.

There are at least two cases when convergence is crucial. The first is bootstrapping, that is, the scenarios when many new nodes join according to a possibly artificial non-random pattern. The other case is recovery from damage. Figure 4 illustrates these two cases.

Figure 4(a) shows convergence after the network is initialized in three different ways: at random, as a lattice, and as a growing structure. Random initialization fills the cache with available peer addresses that are drawn at random. In the case of lattice initialization, the initial network is defined as follows. If the correspondents are $v_1, \ldots, v_n$ then there are edges from $v_i$ to $v_{(i+j) \mod n}$, where $j = -c/2, \ldots, c/2$ (assuming $c$ is even), and $i = 1, \ldots, n$. In the case of growing, the initialization process starts with only one correspondent, $v_0$. Then in each cycle 5% of all the nodes are added, until cycle 20. All the new nodes are initialized with a single connection to the oldest node $v_0$. This can be considered as a worst case scenario because the network growth is maximally unbalanced.

Figure 4(b) shows convergence after a radical fluctuation of membership. During 20 cycles, starting from cycle 20, in each cycle a random 10% of the nodes are replaced with new nodes. This way the number of node removals reaches $2N$ during the 20 cycles. The new nodes are linked to only a single fixed node, just like in the case of the worst case scenario of network growth described above. Each time this fixed node is removed from the network a new one is selected.

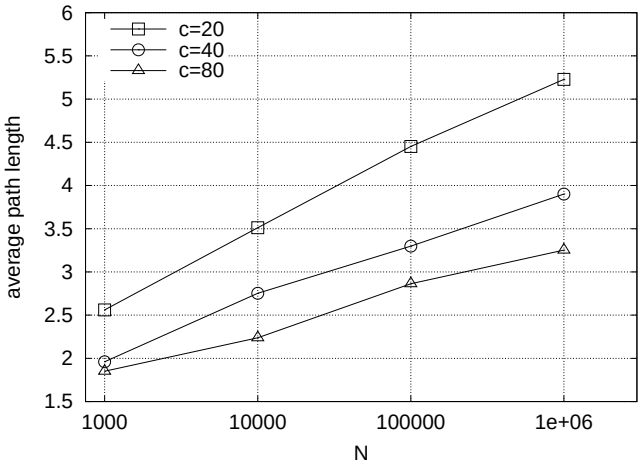



Fig. 5. Average path length as a function of network size and cache size. All data was obtained from $G_{50}$ starting with a random topology.

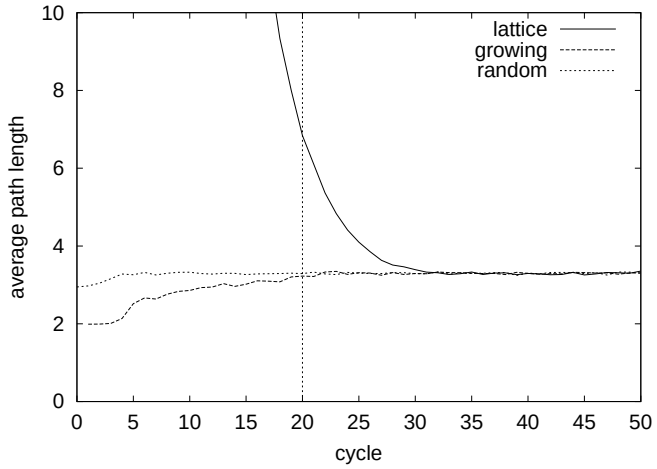
It can be observed that independently of the pattern of subscription or the initial graph topology, the average path length quickly converges to the same value.

### B. Information Dissemination

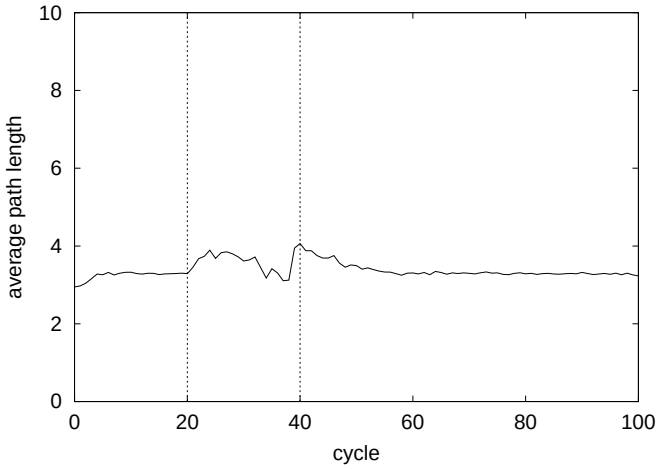
Having seen the convergent behavior of the newscast network, let us turn to the property of average path length in more detail, which is a key feature that determines the efficiency of information dissemination. All results presented here were obtained from the undirected communication graph after running the protocol for 50 cycles to allow the average path length to converge.

To allow efficient information dissemination, the average path length has to be small and it should preferably grow only logarithmically with network size. Figure 5 shows that we clearly have such a growth. Also, we can observe that, as expected, increasing the cache size decreases the path length.

The newscast network cannot be modeled as a random network, however. As Figure 6 shows, the average clustering of the network is very high. The clustering coefficient of a node is defined by the proportion of the number of edges between the neighbors of the node and all possible edges [21]. In other words, if a node has $k$ neighbors, and the subgraph induced by these neighbors has $m$ edges, then the clustering coefficient is $2m/k(k-1)$ since there could be at most $k(k-1)/2$ edges. The clustering coefficient of a graph is the average of the clustering coefficients of its nodes. For comparison, in a random graph where each pair of nodes is connected with probability $2c/n$, the clustering coefficient is exactly $2c/n$. This is much smaller than the values shown, while in such a random graph the average degree is $2c$, approximately the same as in the undirected newscast graphs.




(a) Convergence after bootstrapping. Bar indicates end of growth in the "growing" case.




(b) Response to severe damage. Bars indicate damage interval.

Fig. 4. Convergence of the average path length. In all experiments $c = 40$ and $N = 10^5$ (see text for further explanation).

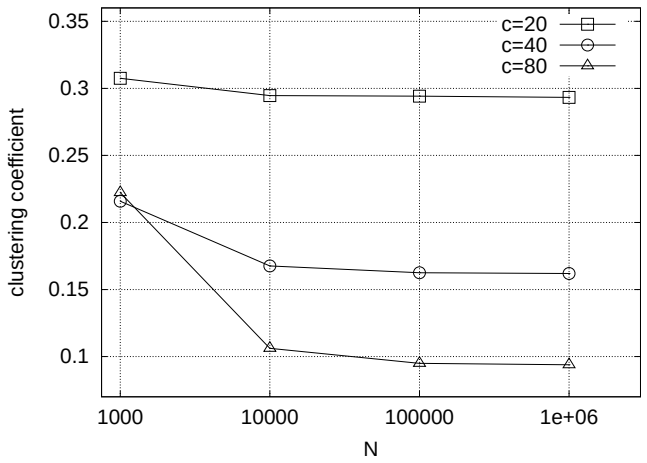



Fig. 6. The clustering coeffi cients a function of network size and cache size. All data was obtained from $G_{50}$ starting with a random topology.

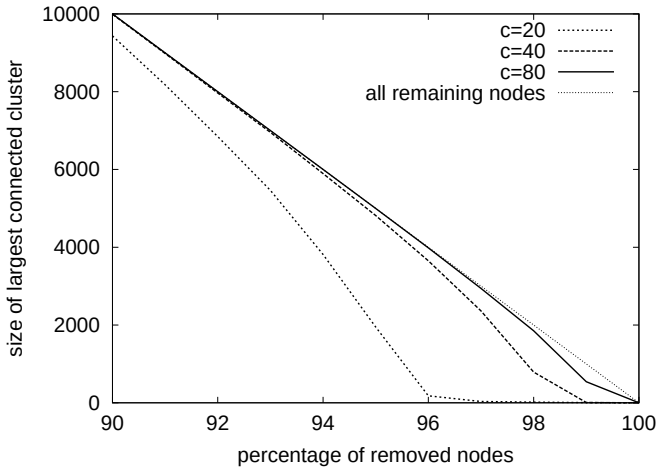



Fig. 7. The effect of random node removal. All data was obtained from $G_{50}$ with $n = 10^5$, starting with a random topology.

The large clustering coefficient is fortunately not problematic because the average path length is small. This means that information dissemination is still efficient because an arbitrary pair of nodes are separated only by a few links. Furthermore, in Section VI we will show that from the point of view of the applications presented here the newscast graph is a sufficiently good approximation of a random graph.

### C. Robustness

One important property of a communication network is robustness to random node removal. As before, we examine the undirected version of the communication graph. Figure 7 shows how the newscast network reacts to random node removal. It can be seen that with a cache size of 80 practically all the remaining nodes are connected.

Even though it can be seen that most of the network remains connected forming one large connected component, it is not clear from the figure when the first small components start to disconnect from this large component. For cache sizes 20, 40 and 80 this happens after removing 68%, 83% and 94% of the nodes, respectively.

### D. Communication Costs

Let us begin with the global communication costs. The cycle length, $\Delta T$, defines the wall clock time of one newscast cycle. The communication cost of one cycle for the overall system depends on the cache size $c$. In each $\Delta T$ time units each correspondent initiates exactly one information exchange session which involves the transfer of $2c$ cache entries. The size of a cache entry can be seen from Figure 3. It has a fixed-sized component and a news item, which is application dependent. In the case of aggregation, a news item will generally be either empty or a single floating point number. Clearly, the global communication costs of one cycle grows

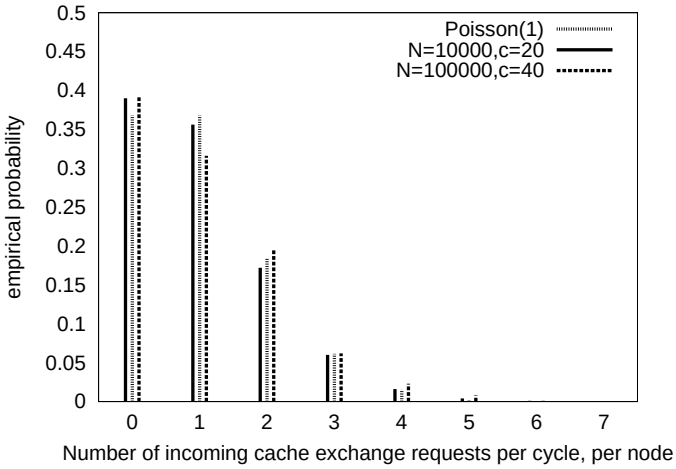



Fig. 8. Incoming number of cache exchange requests per cycle, per node. The empirical probabilities are calculated from a sample generated by recording the incoming connections for a fixed node in each cycle during 1000 cycles of the protocol.

linearly with the network size, but stays constant from the perspective of a single node.

For the communication costs at a single node, we observe the following. Each node initiates one connection in each cycle, and it is contacted by a random number $k$ of peers. Assuming unbiased random cache content, it can easily be shown that for large network size the distribution of $k$ is Poisson$(1)$. Figure 8 shows that the distribution of the incoming connections is close to the Poisson distribution.

An important consequence, which is valid for both local and global costs, is that there are no performance peaks. The communication costs are evenly distributed over the set of nodes and—more importantly—over time on one node. This feature is an important advantage from the point of view of scalability: independently of system size, each correspondent will experience the same predictable load without peaks.

## VI. Distributed Aggregation Revisited

In this section we introduce the basic framework for Sections VII and VIII. These sections will present theoretical results about the convergence speed of the algorithm presented in Section III for the two aggregation functions maximum and average. They also contain experimental results of running these algorithms on top of newscast.

In both cases theoretical analysis will assume that the topology of the communication network is random. We know from Section V that in the newscast network this assumption is not true in a strict sense due to clustering. The purpose of the empirical analysis therefore is to validate the theoretical results by showing that the newscast network is sufficiently random from the point of view of the aggregation algorithms.

### A. Theoretical framework

For the purpose of mathematical analysis we translate the networking terminology into mathematical structures and concepts. In this framework, we can formulate our approach

*// vector a is the input*
**do** $N$ times {
  $(i, j) =$ getPair()
  $a[i] = a[j] =$ aggregate$(a[i], a[j])$
}

Fig. 9. One cycle of the aggregation algorithm.

as follows. We are given an initial vector of numbers $\mathbf{a}_0 = (a_{0,1} \dots a_{0,N})$. We shall model this vector by assuming that $a_{0,1}, \dots, a_{0,N}$ are independent random variables with identical expected values and a finite variance.

The assumption of identical expected values is not so strong as it seems. The protocols are not sensitive to the ordering of values, so after any permutation of the initial values the statistical behavior remains the same. Starting with random variables $a_{0,1}, \dots, a_{0,N}$ with arbitrary expected values, after a random permutation the new value at index $i$, $b_i$ will have the distribution

$$P(b_i < x) = \frac{1}{N} \sum_{j=1}^{N} P(a_j < x). \tag{1}$$

That is, we obtain an equivalent probability model where the distribution of random variables $b_0, \dots, b_N$ is identical. Note that the assumption of independence is violated, but—in the case of large networks—only to an insignificant extent.

When considering the network as a whole, one cycle of the aggregation algorithm in Figure 1 (i.e. any wall-clock interval of $\Delta T$ time units) can be looked at as an algorithm which takes a vector as a parameter and produces a new vector of the same length ($N$). Furthermore, the consecutive cycles of the protocol result in a series of vectors $\mathbf{a}_1, \mathbf{a}_2, \dots$. The elements of vector $\mathbf{a}_i$ will be denoted by $\mathbf{a}_i = (a_{i,1} \dots a_{i,N})$

Figure 9 shows one cycle of the aggregation algorithm. Note that all the practical aspects of the overlay topology, synchronization (or the lack of it), distributedness and possible node failures can be modeled by properties and constraints of the method `getPair`. The distributed and local nature of the epidemic protocol underlying this model (Figure 1) can be expressed by the constraint that the returned pair cannot be determined (or affected) by some global property of the value vector, like the maximum of the values for instance.

### B. Implementation with newscast

Aggregation requires an accessible random peer in regular time intervals. Such a peer is what newscast has to provide. In Figure 1 this requirement is expressed by the method `randomPeer`; in the global version of the algorithm in Figure 9, it is expressed by the method `getPair`. There are various ways that newscast can implement these methods. An important observation is that newscast by itself performs two different functions. The first is handling membership management through the exchange of addresses of correspondents when exchanging caches. The second is disseminating information by means of the news items contained in the respective caches. The implementation of aggregation methods

can either be integrated with the dissemination function of newscast, or kept separate.

The weakest integration is to randomly select one or two peers from a correspondent's current cache, and return that as the result of `randomPeer` or `getPair`, respectively. Note that this implementation makes cycle lengths of newscast and aggregation independent, and that they can thus be different.

A stronger integration is to adopt the random peer as selected by the newscast protocol as the one to use for aggregation, but not to make use of newscast's functionality for information dissemination. In this case, the data that is required from the peer is piggybacked with the caches that are sent when the newscast exchange protocol is executed.

The strongest integration is to adopt newscast's random peer selection, and to use the information as stored in that peer's cache for computing the aggregation function itself (i.e. the function `aggregate`).

In the remaining part of the paper we will assume the simplest and weakest integration, unless explicitly stated otherwise. In other words, we assume that the peer is selected randomly from the cache. We also assume that the cycle length of newscast and aggregation is the same ($\Delta T$). However, we will show that there are ways of implementing aggregation as a newscast application so that its performance is significantly improved and therefore the news dissemination feature of newscast can also play an important role.

## VII. Diffusion-based Aggregation

Let us first discuss the case when $\texttt{aggregate}(x,y) = (x+y)/2$. We introduce the following notations for empirical statistics:

$$\mu_i = \mu_{\mathbf{a}_i} = \frac{1}{N}\sum_{k=1}^{N} a_{i,k} \tag{2}$$

$$\sigma_i^2 = \sigma_{\mathbf{a}_i}^2 = \frac{1}{N-1}\sum_{k=1}^{N}(a_{i,k}-\mu_i)^2 \tag{3}$$

Only linear operations are performed on the vector elements so without loss of generality we will assume that the common expected value of the elements of $\mathbf{a}_0$ is zero. The purpose of this choice is merely to simplify our expressions. In particular, for any vector $\mathbf{a}$, if the elements of $\mathbf{a}$ are independent random variables with zero expected value then

$$E(\sigma_{\mathbf{a}}^2) = \frac{1}{N}\sum_{k=1}^{N} E(a_k^2). \tag{4}$$

Furthermore, the elementary variance reduction step in which both selected elements are replaced by their average does not change the sum of the elements in the vector so $\mu_i \equiv \mu_0$ for all cycles $i = 1, 2, \ldots$. This property is very important because it guarantees that the algorithm does not introduce any errors into the approximation. This means that from now on we can focus on variance. Clearly, if the expected value of $\sigma_i^2$ tends to zero with $i$ tending to infinity then the expected value of all vector elements will tend to the correct average $\mu_0$.

Let us begin our analysis of the convergence of variance with some fundamental observations.

*Lemma 1:* Let $\mathbf{a}'$ be the vector that we get by replacing both $a_i$ and $a_j$ with $(a_i + a_j)/2$ in vector $\mathbf{a}$. If $a_i$ and $a_j$ are uncorrelated random variables with expected value 0 then the expected value of the resulting variance reduction is given by

$$E(\sigma_{\mathbf{a}}^2 - \sigma_{\mathbf{a}'}^2) = \frac{1}{2(N-1)}E(a_i^2) + \frac{1}{2(N-1)}E(a_j^2). \tag{5}$$

*Proof:* Simple calculation using the fact that if $a_i$ and $a_j$ are uncorrelated then $E(a_i a_j) = E(a_i)E(a_j) = 0$. ∎

Considering also (4), an intuitive interpretation of this lemma is that after an elementary variance reduction step both participating nodes will contribute only approximately the half of their original contribution to the overall expected variance, provided they are uncorrelated. In the extreme case of maximal correlation ($a_i \equiv a_j$) the variance reduction is zero. From this it can be seen that the assumption of uncorrelatedness is crucial.

In order to derive reduction rates for particular implementations of `getPair`, we introduce a simplifying assumption:

$$E(\sigma_{\mathbf{a}}^2 - \sigma_{\mathbf{a}'}^2) \approx \frac{1}{2N}E(a_i^2) + \frac{1}{2N}E(a_j^2). \tag{6}$$

This formula will be applied instead of (5). This is completely harmless because it reduces the reduction we consider and also because if $N$ is large, the difference is insignificant. This simplification serves the purpose of making our formulas more elegant.

First let us define random variable $\phi_k$ to be the number of times index $k$ was selected as a member of the pair returned by `getPair` during the calculation of $\mathbf{a}_{i+1}$ from the input $\mathbf{a}_i$ (i.e. during one cycle). In networking terms, $\phi_k$ is the number of peer communications node $k$ was involved in during cycle $i$.

*Theorem 1:* Let us assume that `getPair` has the following properties.

1) Each pair of values selected by the index pairs returned by each call to `getPair` are uncorrelated.
2) The random variables $\phi_1, \ldots, \phi_N$ are identically distributed. Let $\phi$ denote a random variable with this common distribution.
3) After $(i, j)$ is returned by `getPair` the number of times $i$ and $j$ will be selected by the remaining calls to `getPair` has identical distribution.

Then the expected value of variance reduction during one cycle is given by

$$E(\sigma_{i+1}^2) \approx E(2^{-\phi})\frac{1}{N}\sum_{k=0}^{N} E(a_{i,k}^2) = E(2^{-\phi})E(\sigma_i^2) \tag{7}$$

*Proof:* The proper formal proof is long and technical but the following informal description is sufficient to reproduce it. The basic idea is thinking of the value $E(a_{i,k}^2)/N$ as a quantity of some material. Equation (6) tells us that each time index $k$ is selected we loose half of the material, if the value of the selected peer is uncorrelated to that of $k$, and the remaining material will be divided among the locations. Using assumption 3 we can observe that it does not matter where a given piece of the original material ends up, it will have the same chance of loosing its half then the proportion that stays at the original location. That means that the original

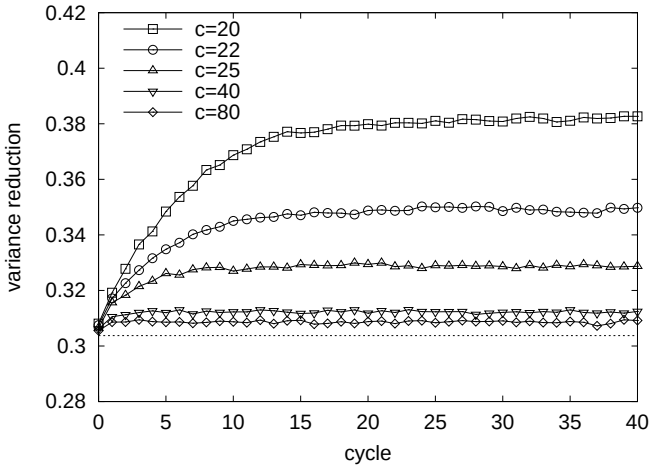



Fig. 10. The variance reduction rate ($\sigma^2_{i+1}/\sigma^2_i$) as a function of consecutive cycles of the averaging protocol on a dynamic newscast network. The dotted line is at $1/(2\sqrt{e})$, the theoretically predicted rate. $N = 10^6$.

material will lose its half as many times on average as the expected number of selection of $k$ by `getPair`, hence the term $\frac{1}{N}E(2^{-\phi_k})E(a^2_{i,k}) = \frac{1}{N}E(2^{-\phi})E(a^2_{i,k})$. Applying this for all $k$ and summing up the terms we have the proof. ■

From this result it is clear that $\phi$ plays a key role, defining fully the rate at which the variance is reduced in each cycle. We have seen in the previous section that in the case of newscast $\phi$ can be approximated by $\phi = 1 + \phi'$ where $\phi'$ has the Poisson distribution with parameter 1, that is, for $j > 0$

$$P(\phi = j) = P(\phi' = j - 1) = \frac{1}{(j-1)!}e^{-1}. \qquad (8)$$

and $P(\phi = 0) = 0$. This is the probability distribution which describes the number of times a correspondent participates in a newscast information exchange. Furthermore, this distribution is independent of $N$, the network size. The implementation of diffusion-based averaging on top of newscast as described in Section VI-B implies the same distribution. Substituting this into the expression $E(2^{-\phi})$ we get

$$E(2^{-\phi}) = \sum_{j=1}^{\infty} 2^{-j}\frac{1}{(j-1)!}e^{-1} = \frac{1}{2e}\sum_{j=1}^{\infty}\frac{2^{-(j-1)}}{(j-1)!} = \frac{1}{2e}\sqrt{e} = \frac{1}{2\sqrt{e}}. \qquad (9)$$

Since the dynamics of the newscast protocol define a non-trivial complex network, we have to verify the assumptions of Theorem 1 empirically. On one hand, we have seen that in the newscast communication network there is significant clustering which might hurt the assumption of uncorrelatedness. On the other hand, the average path length in the network is low and the network itself changes constantly. These properties are useful to break possible correlations.

Figure 10 illustrates the behavior of the diffusion-based averaging protocol on top of newscast. The implementation of diffusion-based averaging is as described in Section VI-B. Cycle 0 corresponds to the first cycle of the averaging protocol, not newscast. The newscast network was already converged when starting averaging.

The figure compares the observed variance reduction with the theoretically predicted rate $1/(2\sqrt{e})$. The network size is $N = 10^6$, the output of a single run is shown. We can see that in the first cycle the observed rate matches the theoretically predicted rate. This is due to the fact that the values stored at each node were initialized in an uncorrelated way. The consecutive execution of the protocol introduces correlation.

It is clear from the observed data that this correlation seems to reduce dramatically at a threshold of approximately $c = 20$. But even with $c = 20$ the variance reduction rate is acceptable, although drifts away from the theoretically predicted rate. With $c = 40$ the theoretical prediction has an acceptable accuracy which indicates that—besides taking care of membership management—from the point of view of this protocol the newscast layer can provide a sufficiently random sampling of available peers.

## VIII. Epidemic-based Aggregation

Here we discuss the case when `aggregate`$(x, y) = \max(x, y)$. We would like to approximate the speed at which the maximum becomes known by the nodes in the newscast network. Considering the simple implementation on top of newscast, as described in Section VI-B, it is easy to see that the dynamics of this process is completely identical to that of broadcasting by means of the push-pull anti-entropy epidemic protocol [1]. The reason is that it is guaranteed that the overall maximum will be advertised by all nodes once they see it. Of course, if many nodes hold the maximal value originally, the process is even faster. In the following we assume that the maximum is unique.

We present a simple model of push-pull epidemic broadcast based on the models of push and pull epidemic broadcasts presented in [1]. Let us assume that method `randomPeer` in Figure 1 returns an unbiased sample of the whole network. Let $p_i$ denote the probability that a fixed node does not know the maximum at cycle 0. If the maximum is unique originally (in cycle 0), then $p_0 = 1 - 1/N$. Let us express $p_{i+1}$ as a function of $p_i$. Clearly, a node will not know the maximum in cycle $i + 1$ (1) if it did not know it in cycle $i$, (2) the peer node it chose as a contact did not know it either (pull) and (3) no peer nodes that knew the maximum contacted the node (push). Formally we can write this as

$$p_{i+1} = p_i p_i \left(1 - \frac{1}{N}\right)^{N(1-p_i)} \qquad (10)$$

From this equation it follows that

$$p_{i+1} < p_i^2 < p_0^{2^{i+1}} = \left(1 - \frac{1}{N}\right)^{2^{i+1}} \qquad (11)$$

This result suggests that $p_i$ decreases super-exponentially fast.

Note that these results make the fundamental assumption that a random sample of the members of the network is available. In the original anti-entropy protocol this assumption holds because a complete list of members at each node is available. Clearly, in newscast it is not evident that this

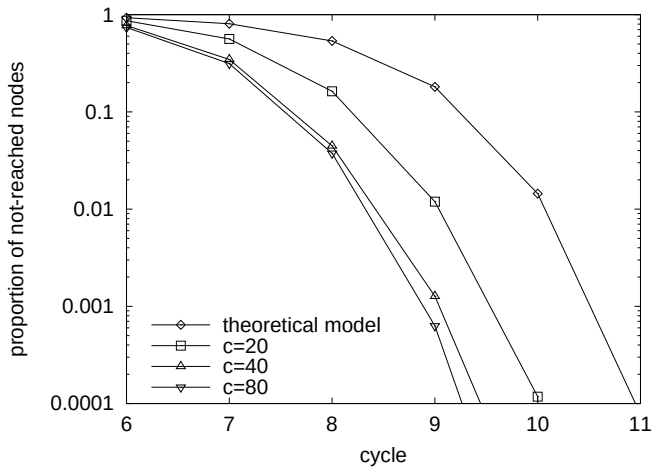



Fig. 11. The proportion of nodes which have not learned about the maximum, as a function of consecutive cycles of the maximum finding protocol. $N = 10^5$, points are averages of 50 runs. Standard deviation is not shown, it is several orders of magnitude lower than the average.

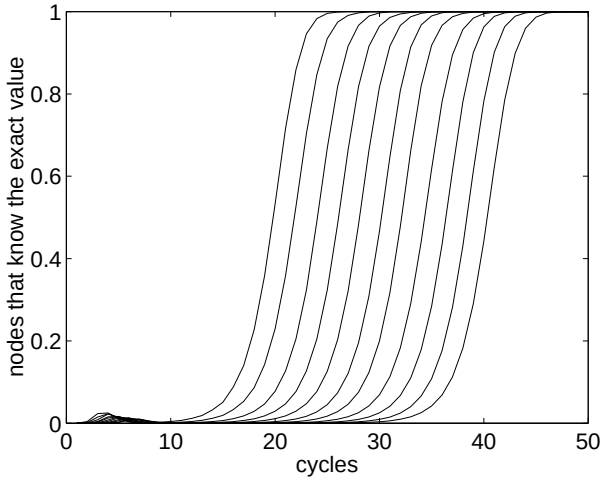



Fig. 12. The proportion of nodes that know the exact network size as a function of the number of cycles $k$ and network size $N$. Consecutive lines (from left to right) correspond to networks with $2^{10}, 2^{11}, \ldots, 2^{20}$ nodes.

assumption is fulfilled with a reasonable accuracy. For this reason empirical validation is necessary.

Figure 11 shows the convergence of $p_i$ when newscast is used as a source of random peers. In all cases we can see that the convergence speed is faster than exponential. Somewhat surprisingly, we see that convergence is much faster than predicted by the model in (10). The reason is that the model assumed that in cycle $i+1$ each node communicates its value from cycle $i$. However, in reality, it is possible that when a node contacts its peer in cycle $i+1$, it has already completed information exchange steps with other peers in the same cycle, as a passive partner. This way, it is possible that it learns about the maximum in cycle $i+1$ and passes this knowledge on in the same cycle, an effect the model does not account for.

## IX. Applications in Network Monitoring and Control

So far we have introduced the concept of newscast computing and analyzed two simple algorithms for calculating the average and finding the maximum. Now we will demonstrate that these algorithms can be used for monitoring network size, measuring the total amount of resources, or distributing alarm signals.

*1) Determining the Size of a Network:* In Section III we briefly mentioned the possibility of finding the number of nodes in a network by calculating the average of a *peak distribution* $\mathbf{x} = (1, 0, \ldots, 0)$, that is, the distribution where one node holds the value 1 and all the others hold 0, and using $1/\overline{x}$ as an estimate of the network size $N$. Now we will take a closer look at this method.

The averaging algorithm that we have analyzed in depth in Section VII has two nice properties: (1) the average of all node values does not change with the number of cycles, and (2) the variance is dropping at rate $\alpha^k$, where $k$ denotes the number of cycles, and $\alpha$ is slightly smaller than 0.4 (see Figure 10). In other words, when the averaging algorithm is applied to the peak distribution, node values converge exponentially fast to $1/N$. To get an idea about the actual performance of this approach we have run a number of experiments using the idealized averaging algorithm from Figure 9, where the `getPair` method returns two randomly selected nodes. We have focused on two aspects: accuracy and speed.

First, we looked at how the number of cycles affects the number of nodes that know the *exact* size of the network. The averaging algorithm was applied to networks of size varying between $2^{10}$ and $2^{20}$ nodes. After every cycle all nodes whose values (after rounding) were equal to the size of the network were counted. Each experiment was repeated 100 times and results were averaged. They are shown in Figure 12. We can see that for networks with size ranging from $2^{10}$ to $2^{20}$ nodes there are about 25-45 cycles needed for full convergence to the exact value $N$. (More precisely, as we represent real numbers by standard 64-bit floats, the "exact value" means here the result of rounding to the closest integer.)

Although it is quite impressive that one can find the exact number of nodes with help of a simple averaging algorithm, in practice much smaller accuracy is sufficient (e.g., 1%). To get a better insight into the relation between this limited accuracy and the number of required cycles we run simulations until *all* the nodes knew the approximate network size. Again, for each setting 100 runs were performed. The results are summarized in Figure 13. We can see that the average number of required iterations dropped from 25-45 to 20-32.

Notice that due to its fast convergence rate, the averaging algorithm can be regularly restarted on a fresh peak distribution (e.g., every 100 cycles). In this way all the nodes are constantly aware of the network size.

*2) Monitoring Total Amount of Resources:* A simple equation: $\sum x = N\overline{x}$ now has an important implication: once we know how to find $N$ and the average $\overline{x}$, we immediately know how to find the sum of $x$'s. And in the light of results presented in the previous sections we already know that both ingredients (i.e. $N$ and $\overline{x}$) can be found in a small number of cycles and

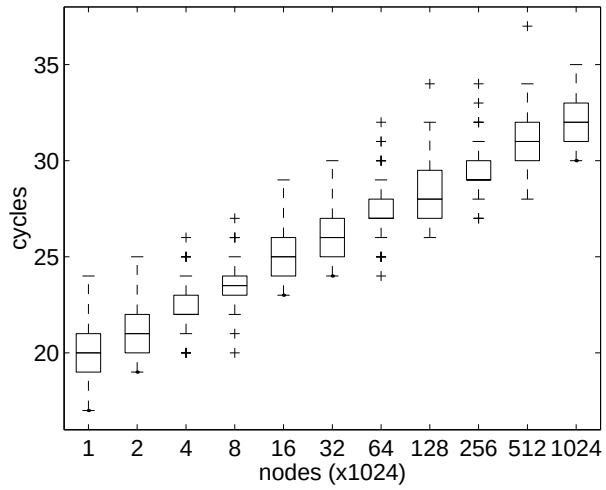



Fig. 13. The number of cycles needed for all agents to learn the approximated network size. Boxes have lines at the lower quartile, median, and upper quartile values. The whiskers (lines extending from each end of the box) show the extent of the rest of the data. Outliers are data with values beyond the ends of the whiskers.

with an arbitrary accuracy. Therefore, we have an efficient algorithm that can be used for finding the sum of all values that are stored by nodes.

In the context of network monitoring it means that one can constantly measure the total load of all the nodes, the total capacity of their disk drives, the total number of files or amount of data stored, etc.

*3) Monitoring Migration of Agents:* Yet another application of aggregation algorithms for network maintenance is measuring the number of agents that joined the network within the last *epoch*. By an epoch we mean here a fixed number of cycles (e.g., 1000). Counting agents that joined the network is simple: every agent has to keep one bit of information: 1, if it joined the network within the current epoch; 0, otherwise. At the end of every epoch the sum of these bits is found, and 1's are set to 0's.

Combining this information with the network size one can calculate the number of agents that left the network during the epoch. This way the fluctuation of the network can be monitored.

An additional mechanism could be built-in into every agent: whenever the number of failures that are observed by a single agent exceeds a certain threshold, this agent may rise an alarm.

*4) Alarm Processing:* As a next example, consider the problem of efficiently broadcasting an alarm signal. As we mentioned, broadcasting is just a special case of computing and disseminating the maximum value across the network. In particular, we let $\mathbf{a}_0 = (a, 0, \ldots 0)$ where $a \neq 0$ denotes the alarm value generated by node 1. In this example, we assume the strongest integration between newscast and aggregation. In other words, we base aggregation (i.e. broadcasting) on the values contained in the cached news items. Whenever `getNews()` is called, the current alarm value (which is either 0 or $a$) is passed as news item to the underlying correspondent, who subsequently stores it in its cache. Likewise, when a correspondent calls `newsUpdate()`, it passes all $c$ news items stored in its cache to its associated agent. If $v_i$ denotes the value of the $i^{th}$ news item, then the agent will adjust its current value to $\max\{v_1, \ldots, v_c\}$.

We already showed that broadcasting can be done super-exponentially fast. However, this dissemination speed comes at a price: there are many nodes that receive the alarm value more than once. We can improve this situation by taking the age of a cached news item into account. Whenever a news item is subject to a cache exchange, its age is incremented by 1.

Instead of randomly selecting a peer from the cache, a node adopts the following selection policy. If a node has already seen the alarm signal, it will give preference to a peer whose cached news item (1) has value 0, and (2) is young. The reasoning is that most likely this peer has not yet seen the alarm signal, whereas other peers with a zero-valued, but old news item are more likely to have already been contacted.

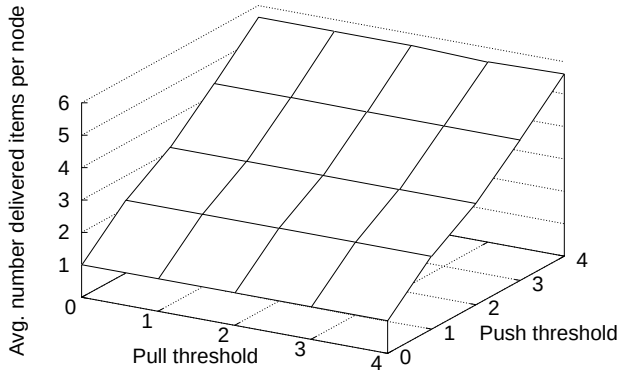
For the same reason, if a node is yet unaware of an alarm, it will give preference to a peer whose cached news item is old. Note that the cache of nodes who are unaware of an alarm contains only zero-valued news items.

The effect of this *selective push-pull policy* can be observed in Figure 14(a). The z-axis shows the average number of delivered alarm signals per node. The x-axis shows the *pull threshold*, which is relevant only for a node that is unaware of the alarm. The pull threshold is the minimal age that a news item should have before its associated peer will be contacted. Note that the pull threshold does not affect the average number of delivered alarm signals: no node will ever see the value more than once. Likewise, the y-axis shows the *push threshold*, which is relevant only for nodes that already have seen the alarm. The push threshold is the maximum age a news item can have in order to for its associated peer to be contacted. From this figure, it can be seen that the push threshold is indeed the only factor for determining the average number of delivered alarm signals per node.

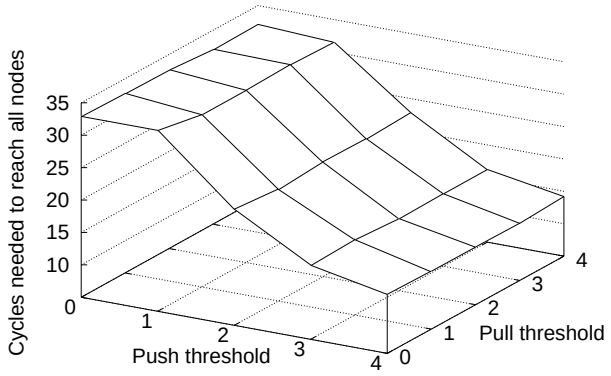
Lowering the number of deliveries comes at a price: the number of cycles needed to disseminate the alarm to all nodes increases. Again, it is the push threshold that is the dominant factor, as shown in Figure 14(b). In this case, the z-axis shows the number of cycles needed to deliver the alarm to all nodes. The x-axis shows the push threshold; the y-axis the pull threshold.

## X. Conclusions

We introduced newscast computing, a general framework for distributed information processing. The core of newscast computing is formed by the newscast protocol, which offers membership management and information dissemination services to distributed applications. Within this framework we developed several algorithms for data aggregation (averaging, counting, summing) and information dissemination (propagation of extreme values or alarms). Using an idealized model we were able to provide statistical analysis of the main properties of these algorithms, namely convergence rate and accuracy. It turned out that the averaging algorithm converges exponentially fast to the (exact) average, whereas single values




(a) The average number of delivered alarms per node.




(b) The dissemination speed of an alarm signal.

Fig. 14. Performance of alarm signal propagation.

(e.g., maximum) can be propagated in epidemic style super-exponentially fast. Numerous experiments with the actual implementation of the newscast model fully confirmed these theoretical findings.

A combination of the newscast protocol with aggregation algorithms resulted in a very robust, scalable, adaptive, and yet very simple and efficient solution of typical network monitoring problems: finding the network size, measuring migration of nodes (counting nodes that join and leave the network), estimating the average load, or fast propagation of alarms. Modest memory and CPU requirements make our solution particularly suitable for monitoring huge networks of small computing devices like mobile phones, PDA's or sensors, where a fully decentralized approach seems to be inevitable.

## REFERENCES


[1] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database management," in *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*. Vancouver: ACM, Aug. 1987, pp. 1–12. [Online]. Available: http://www.acm.org/dl

[2] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, "From epidemics to distributed computing," *IEEE Computer*, to appear.

[3] R. van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Middleware '98*, N. Davies, K. Raymond, and J. Seitz, Eds. Springer, 1998, pp. 55–70.

[4] R. Van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems*, vol. 21, no. 2, May 2003. [Online]. Available: http://www.acm.org/dl

[5] N. T. J. Bailey, *The mathematical theory of infectious diseases and its applications*, 2nd ed. London: Griffi n, 1975.

[6] B. Bollobás, *Random graphs*, 2nd ed. Cambridge; New York: Cambridge University Press, 2001.

[7] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Peer-to-peer membership management for gossip-based protocols," *IEEE Transactions on Computers*, vol. 52, no. 2, Feb. 2003.

[8] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov, "Lightweight probablistic broadcast," *ACM Transactions on Computer Systems*, to appear.

[9] C. Law and K.-Y. Siu, "Distributed construction of random expander graphs," in *Proceedings of The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'2003)*, San Francisco, California, USA, Apr. 2003. [Online]. Available: http://perth.mit.edu/ ching/

[10] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter peer-to-peer networks," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 21, no. 6, pp. 995–1002, Aug. 2003. [Online]. Available: http://www.cs.purdue.edu/homes/gopal/papers.html

[11] R. van Renesse, "The importance of aggregation," in *Future Directions in Distributed Computing*, ser. Lecture Notes in Computer Science, A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, Eds., no. 2584. Springer, 2003, pp. 87–92.

[12] I. Gupta, R. van Renesse, and K. P. Birman, "Scalable fault-tolerant aggregation in large process groups," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, Göteborg, Sweden, 2001. [Online]. Available: http://www.cs.cornell.edu/Info/People/rvr/papers/?B2=Papers

[13] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, "Estimating aggregates on a peer-to-peer network," submitted for publication. [Online]. Available: http://dbpubs.stanford.edu/pub/2003-24

[14] H.-G. Hegering, S. Abeck, and B. Neumair, *Integrated Management of Networked Systems*. San Mateo, CA.: Morgan Kaufman, 1999.

[15] A. Liotta, G. Pavlou, and G. Knight, "Exploiting Agent Mobility for Large Scale Network Monitoring," *IEEE Network*, vol. 16, no. 3, May 2002.

[16] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notifi cation Service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[17] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE Journal on Selected Areas in Communication*, vol. 20, no. 8, pp. 100–110, Oct. 2002.

[18] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware 2001*, ser. Lecture Notes on Computer Science, R. Guerraoui, Ed., vol. 2218. Berlin: Springer-Verlag, Nov. 2001, pp. 329–350.

[19] B. Gedik and L. Liu, "PeerCQ: A Decentralized and Self-Con guring Peer-to-Peer Information Monitoring System," in *23rd International Conference on Distributed Computing Systems*, IEEE. Los Alamitos, CA.: IEEE Computer Society Press, May 2003, pp. 490–499.

[20] M. Bodlaender, J. Guidi, and L. Heerink, "Enhancing UPnP Discovery with Liveness," in *Consumer Communications and Networking Conference*. Los Alamitos, CA.: IEEE Computer Society Press, Jan. 2004.

[21] D. J. Watts, *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton, NJ: Princeton University Press, 1999.