Large-Scale Newscast Computing on the Internet

Márk Jelasity
Maarten van Steen
Vrije Universiteit, Amsterdam
{*jelasity,steen*}*@cs.vu.nl*

Internal report IR-503
October 2002

This paper introduces the *newscast model of computation* for large-scale computing on the Internet. The engine realizing this model is a lazy fully distributed information propagation protocol among the participants which is responsible for membership management and communication. It maintains a constantly changing communication graph over the participants. This graph has useful *emergent* properties like small diameter and sufficiently random structure without deploying special purpose protocols to achieve these properties. For adding a new participant only the address of an arbitrary member is needed and for removal no action is necessary. We provide theoretical and empirical evidence that—besides being simple and lightweight—our newscast computing engine is extremely scalable and robust. We also suggest some interesting application areas including information dissemination, monitoring of large systems, resource sharing and efficient multicasting.

Keywords: Epidemic protocols, dissemination-oriented communication, large-scale distributed systems, probabilistic reliable multicasting, complex adaptive systems

vrije Universiteit

Department of Computer Science

I. Introduction

The Internet is gradually making a transition from its original role as computer network to that of a distributed system. In contrast to a computer network, which essentially supports only data transfer in the form of messages, a distributed system aims to provide a single coherent view on a potentially large collection of independent computers [1]. In essence, a distributed system aims to make its underlying resources available to applications, but such that the distribution of data, processes, and control is transparent [2].

Distribution transparency has been quite successfully realized for local-area systems, that is, distributed systems that span a local-area network. However, combining scalability and transparency is difficult, and simply trying to port local-area solutions to networks such as the Internet will generally fail for two reasons.

First, many existing solutions have been molded into a client-server architecture, which scales badly when the number of clients grow. To alleviate scalability problems, special measures such as client-side caching and server replication need to be taken. Applying these scaling techniques may require a considerable development effort, notably when strong consistency issues are at stake.

The second reason why many local-area solutions cannot be simply ported to large-scale networks, is because these solutions are based on powerful multicasting facilities of the underlying hardware. Such facilities are not available in the Internet, generally rendering the original solution useless. Again, special measures need to be taken to adopt the original solution without too many changes. For example, the design of PGM, a scalable reliable multicast protocol [3], has been strongly motivated by the need to support local-area publish/subscribe solutions that deploy multicasting.

A main advantage that multicast-based solutions have over client-server solutions is their often inherent decentralized nature. This makes these solutions particularly attractive for large-scale networks, except that building scalable reliable multicasting schemes is notoriously difficult [4], [5]. Fortunately, matters improve when dealing with *probabilistic* schemes in which no hard guarantees are given concerning the delivery of a multicast message. If a multicast message is required to be only eventually delivered with a high probability to all current group members, it appears that scalable solutions are quite feasible [6], [7], [8].

The key idea underlying probabilistic multicasting is the use of a simple scheme for reliably disseminating information. Epidemic protocols have shown to be highly effective for this purpose if sufficient information is available over group membership. In this paper we describe a novel gossip-based scheme for information dissemination that distinguishes itself from other similar solutions by its simplicity and scalability. Our approach deploys lazy propagation of information simultaneously disseminating membership and application-specific data. The function of our information propagation protocol is maintaining a stable environment with minimal effort that can still provide certain guarantees and interfaces for applications. We call this environment the *newscast model of computation*. The applications of the model include not only evolutionary computation (EC), a subfield of computational intelligence which originally motivated our research [9], but also probabilistic effective and reliable multicasting, large-scale distributed file sharing, and resource discovery and allocation. An important distinguishing feature of our scheme is that it requires only an extremely simple membership protocol. To join a group, a process can contact *any*, arbitrarily chosen member and merely copy that member's list of current neighbors. No messages need to be sent to leave a group: a process merely stops its communication.

In our model, computations are performed by *agents* whose main activity is to generate and process *news*. Exchanging news is the only form of communication. A news item generated by an agent is overridden by

a fresher item from the same agent even if no other agent has ever processed it. The communication model is generative in the sense that agents are temporally and referentially uncoupled [10], [11]. In other words, a news recipient need not be known or even exist at the time a news item is published, nor does a news item need to explicitly identify its sender and receivers. However, as we shall see, referential coupling can easily be implemented by newscast applications if necessary.

The key underlying idea is that the model works completely on a statistical basis. Both news and agents are supposed to be present in large quantities. What is relevant is not the individual news items or agents but the dynamics of the system as a whole. These dynamics can be such that it leads to a stable equilibrium (control), it can follow the changes of its environment (information dissemination), it can react to some events with very quickly reaching a special state (monitoring) or it can evolve towards some direction (problem solving, etc). Like in the case of ant colonies or other swarms of insects, applications can define the dynamics of the system by defining the behavior of the components in a way that leads to the desired "macro behavior."

Such applications inherit the feature of graceful degradation without any extra investment. In other words, if parts of the system are damaged and fail to work properly, the overall dynamics can continue to be the same or similar so that the system will not even come to a partial or complete halt. Even though it is possible to write badly designed applications in every model, as we will see, newscast applications can often be arbitrarily partitioned with the resulting parts simply continuing their computation, but also be re-merged later on.

Our implementation of the newscast model does not rely on any central services or servers. Our main contribution is that we present a highly reliable and fully distributed dissemination scheme for implementing the two main functions: information dissemination and membership management. This scheme can scale to extremely large networks such as the Internet. The features of the model given in Section III are *emergent* from the implementation. For example, the full membership list is not known by anyone ever in the system, only small portions of it are stored in a distributed fashion. Mathematical analysis and simulations will be given to support our claims on scalability and robustness and to prove that the specifications of the newscast model indeed hold. We also demonstrate that only simple local mechanisms are needed to establish highly robust communication graphs that exhibit desirable small-world properties such as high connectivity and small diameter [12].

Section II gives an overview of related work. Section III gives the formal specification of the newscast model of computation. Section IV elaborates on the possible types of applications. It gives a classification illustrated with examples, including information dissemination, file sharing, and distributed resource allocation. Section V describes the implementation of our model based on an epidemic protocol. Theoretical and simulation results will be given to prove that our engine indeed fulfills the requirements specified in Section III.

II. Related Work

There are different ways to view our work in order to compare it to other systems. One view is to focus on its model of computation, which describes the operations and their semantics as offered to applications. Another is to concentrate on the algorithms and mechanisms by which that model of computation is realized.

We have intentionally provided an explicit description of the newscast model of computation. Such a description is important for application developers and is comparable to the separation between *services* and *protocols* in reference models for computer networks [13]. Taking this functional perspective, the newscast

model is best compared to what Cheriton coined dissemination-oriented communication [14]. In these systems, a sender and receiver never explicitly establish a connection. Instead, a process attaches itself to a channel without further need to reveal its identity. A channel generally allows M-to-N communication where the number of receivers is potentially very large.

This model is currently deployed in many publish-subscribe systems in which each channel is associated with a specific subject (see, e.g., [15]). A more advanced approach is content-based addressing by which, in principle, arbitrary *(attribute,value)*-pairs can be specified to express a receiver's interest in specific messages [16], [17], [18].

The newscast model forms a basis for dissemination-oriented communication by deploying lazy message propagation in what is essentially a broadcast system. An important distinction with existing approaches, is that broadcasting can be achieved only by special application-level measures. In particular, a sender will either need to repeatedly pass its message to the news agency, or repeaters need to be deployed to assist in message propagation. Filtering, either based on subjects or actual message content, is always done at the receiver's side.

In this sense, the newscast model is very different from the core components used in Internet-wide peer-to-peer systems such as Chord [19], Pastry [20], Tapestry [21], and CAN [22]. These systems provide key-based routing and searching, possibly using replication to improve fault tolerance and client-perceived performance. A major difficulty in these systems is content-based searching [23], [24], an issue that is solved in the newscast model by its disseminative nature, but potentially at the cost of higher resource usage.

From an implementation perspective, the newscast model is one of the many gossip-based systems that have been developed since the 1980s. These systems deploy an epidemic protocol for disseminating information across a collection of nodes. Well-known examples of this dissemination style can be found in replicated databases [25], [26], [27] and probabilistic multicasting [6], [7], [8], but also in applications such as failure detection [28], [29].

An important drawback of many gossip-based systems is that in order for the epidemic protocol to be effective, each iteration requires that a node communicates with a randomly chosen other node. In other words, at each iteration, the set of neighbors that each node has corresponds to a uniformly selected subset of all nodes. As a consequence, it is necessary to know the entire set of nodes, in turn leading to scalability problems. Moreover, a special membership protocol is often needed to let nodes join and leave, while keeping randomness properties of the underlying communication graph.

In contrast, in the newscast model each node has only a fixed-size partial view on the total member set. Moreover, we show that in order for a node to join it can contact any one of the current members. Within just a very few iterations of the protocol, the underlying communication graphs will exhibit the same properties as before. Likewise, leaving is simple: it requires no additional communication whatsoever. In other words, the newscast model requires only the minimal membership protocol that one can think of, namely contacting an existing member when joining. We see this as a major and important improvement over many existing gossip-based systems.

From the perspective of information dissemination, the research on probabilistic reliable multicasting as described in [30] is closest to our work, although multicasting is actually an application of newscast computing. Similar to our approach, nodes in this model maintain only a partial view of the entire system and there is not a single node that has global knowledge on how nodes are connected to each other. Each node gossips

`timestamp`	creation time of news item
`content`	news content, possibly empty

TABLE I

FIELDS OF A NEWS ITEM.

messages with nodes from its partial view, and by doing so dynamically adapts its current view. The authors show that the size of partial views is $O(\log n)$ where n is the total number of nodes.

However, an important difference with our newscast model is that probabilistic reliable multicasting is based on constructing and maintaining a random graph (see, e.g., [31]). To this end, a sophisticated membership protocol is needed [32]. An advantage is that this protocol allows nodes to be grouped taking network proximity into account, a property that we have not yet explored in our model. On the other hand, the protocol is relatively complex. Moreover, it is unclear to what extent randomness in the underlying communication graphs is actually maintained as this has not been proven or verified. From our own experience, we believe that some caution is in place. For example, our initial assumption that newscast computing leads to random communication graphs turned out to be false. Instead, we found the graphs to have a small-world topology, a result that is much in line with analyses and experimental results reported on complex networks [33].

III. PROBABILISTIC NEWSCAST COMPUTING

The two main building blocks of a newscast computing application are a *collective of agents* and a *news agency*. The basic idea is that the news agency asks all agents regularly for *news* and also provides them with news about the other agents in the collective. The definition of what counts as news is application dependent, in fact this is the most important aspect of an application. The agents live their lives (perform computations, listen to sensors and the news, etc.) and based on the computations they have completed and the information that have collected they must provide the news agency with news when asked. Examples on how to apply this abstract framework to define useful applications are given in Section IV.

Note that even though the news agency is presented as a sort of server in the model of computation, its functionality can be implemented in a fully distributed fashion as we discuss in Section V.

A. Definitions of Operators

The operators are applied by the news agency exclusively, the agents are passive, they never initiate communication with the news agency. The news agency applies two operators: `getNews()` and `newsUpdate(news[])`. Both have to be implemented by all agents and will be called by the news agency regularly. The fields of a news item are shown in Table I.

To be able to discuss the semantics of these operators let us introduce some notations. Let c denote the *cache size*. This is the maximal size of the array in the parameter of `newsUpdate`. Let t_r denote the *refresh rate*. This defines a time interval which determines the frequency the operators are called for a given agent. Let n denote the size of the agent collective. Note that even though we have to deal with the case when $c \geq n$, the intended normal setting is $c \ll n$. The operators are defined as follows:

`getNews():`

1. This method must be implemented by every agent.
2. It must return data of unspecified type (possibly `null`) which is used by the news agency to initialize the content field of a new news item (see Table I).
3. The news agency calls this method $1 + \xi$ times in each t_r time units for every agent where ξ is a random variable with a Poisson distribution. Its expected value and variance are $\mu = \sigma^2 = 1$.

`newsUpdate(news[]):`

1. This method must be implemented by every agent.
2. It has no return value.
3. The news agency calls this method $1 + \xi$ times in each t_r time units for every agent where ξ is a random variable with a Poisson distribution. Its expected value and variance are $\mu = \sigma^2 = 1$.
4. The array `news[]` contains $\min(c, n)$ random news items (see Table I) submitted by other agents from the collective.
5. In `news[]` there is at most one item from any agent from the collective. If $n \leq c$ then all agents will be represented.
6. It is *not* guaranteed that the agent set defined by `news[]` is statistically independent from the agent set defined by the previous call to `newsUpdate`. However, it is guaranteed that there is an unspecified subset of size at least $c/2$ which represents an unbiased random sample of the n agents.
7. It is *not* guaranteed that a given news item in `news[]` contains the latest contribution of the corresponding agent.
8. It *is* guaranteed that the timestamp of any item in `news[]` is younger than ct_r.

B. Additional Comments

Since this framework is not very usual, some comments may help to clarify the basic concepts. The most characteristic feature is the concept of news. This term is used to emphasize the fact that every contribution of an agent becomes immediately outdated by its next contribution. This does not mean that older submissions become completely irrelevant. It means that in an application the optimal situation should be to know the latest news from an agent and the fresher the news is the better.

This view on consistency forms a major difference from shared memory consistency models [34], [35] where each and every piece of information put into the shared memory can count on the same consistency defined by the model. In our case old information is treated differently, i.e. it is thrown away whenever a fresher item from the same source is available. On the other hand, it is the only way information is removed, it is not possible to remove news items from the news agency directly.

The model allows for some degree of control over information removal though. The implementation of `getNews` can keep returning empty content (i.e. no news). These no-news items will eventually replace the previous news items (not immediately) in the news agency as follows from item 7 and 8 of the specification of `newsUpdate`. The time bound for the removal is determined by item 8.

Another major feature is the probabilistic nature of newscasting. A given agent has no control over the set of news items it receives and it has no control over the time of reception either. However, note that both the set of news items and the time of its reception obeys the probabilistic constraints given above. Our point is that these constraints represent a sufficient basis for a quite diverse class of applications to be discussed later.

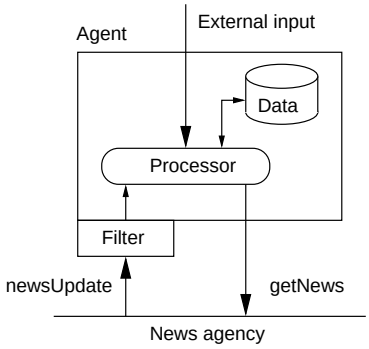

Fig. 1. The general organization of an agent.

For the sake of completeness it should also be mentioned here that our newscast computing engine (described in Section V) allows the size of the collective to change dynamically, i.e. the number of agents can fluctuate during the lifetime of an application. As we will see, removing an agent requires no action at all. Adding an agent is straightforward as well and has basically no costs.

IV. Example Applications

The core of a newscast application is formed by its agents. The general organization of an agent and its relation to the news agency is shown in Figure 1. Delivered news items are generally passed to a local news processor after they have passed a filter that selects only those items that the agent is interested in. Depending on the type of agent, this processor will generate new items that can be passed to the news agency when asked. In many cases, the processor will make use of external input for producing news items. Likewise, its output may depend on previously accepted input and produced output (i.e., the processor is not stateless).

Based on this organization, we distinguish the following types of agents. A *source agent* produces news items independent from the items that are delivered to it by the news agency. Such agents are typically used to report information on local resources, such as resource usage in the case of system monitoring, or information on file content in the case of file sharing.

A *sink agent* merely consumes incoming news items; it never provides news to the news agency. Typically, sink agents are used to attach external applications to the newscast system. We will discuss one such application below.

A *computing agent* takes incoming news items as input for doing a computation, after which it reports the results as another news item. Computing agents are typically used to aggregate results produced by other agents. For example, in the case of systems monitoring, separate agents may process resource usage reports in order to attain information on the global status of a system. This information may be fed back as news into the system so that local measures can be taken to optimize the system.

Finally, a *repeating agent* simply takes the content of an incoming news item and produces a new item with exactly the same content. The main purpose of a repeating agent (or, simply, a *repeater*) is to ensure that certain news items are quickly spread to all agents.

A. Information Dissemination and Multicast

An important application of the newscast model is reliable dissemination of information. A simplistic approach is a collective of source agents that merely report their states. The state of the agent can be defined by

weather sensors for example, consisting of data on pressure, temperature, etc. Looking at the news updates an arbitrary agent receives or connecting a sink agent that serves as a user interface we can get random selections of sensor information. It can be proved based on items 3 and 6 of the specification of `newsUpdate` that we eventually receive information from each agent with a waiting time of $nt_r/2c$ on average. Statistical properties of the waiting time are extensively discussed in Section V-F.2.

We can do better however. To improve the speed by which a news item is disseminated, we can place repeaters in the system that repeat certain types of news they receive. In fact we can implement an epidemic-style multicasting scheme this way. Suppose an agent discovers important news such as a tornado (staying with our previous weather example). Initially at time 0 the tornado is known by only one agent out of n. Let p_i be the probability that an agent in the collective does *not* know about the tornado at time it_r. We know that $p_i = 1$ if $i < 0$ and $p_0 = (n-1)/n$. To approximate p_{i+1} we have to consider that an agent does not know about the tornado at time $(i+1)t_r$ if it did not know about it at it_r (which has probability p_i) and did not learn about it in the meantime. According to this we can give a bound on this probability:

$$p_{i+1} \leq p_i p_{i-c}^{c/2}$$

For the term $p_{i-c}^{c/2}$ we used the assumption of having an unbiased random sample of size $c/2$ (item 6) and assumed that all news items have the maximal age ct_r (an thus the probability p_{i-c} of reporting on the tornado) based on item 8. We assumed that `newsUpdate` is called only once within each t_r time units which is a lower bound based on item 3. After c cycles (i.e. from $i - c \geq 0$ thus $p_{i-c} < 1$) this value starts to converge to zero very quickly and the actual constant proves to be much less than c in our implementation as shown in Section V-F.3. Also note that in fact `newsUpdate` is called twice on average in each t_r seconds.

The expression is very similar to the *pull epidemic protocol* which assumes one contact with a randomly chosen agent in each cycle to check whether it has tornado information [26]. In that case we have

$$p_{i+1} = p_i^2$$

Another implementation of multicasting can be realized outside the newscast model based on the membership information gained from the news updates provided by the news agency. The `getNews` operator has to return the multicast address of the agent. When an agent sees a tornado or hears from it through its own multicast address it sends this information to all addresses received in the latest news update. The theoretical analysis of this approach can be done based on item 6 and the tools developed in [30]. Accordingly, the probability p of reaching each agent from one source is given by

$$\lim_{n\to\infty} p = e^{-e^{-k}} \quad \text{if } \; c/2 = k + O(\log n) + o(1)$$

This means that in this case c has to be increased logarithmically with n.

Finally (going back to the epidemic-style model implemented within the newscast model) consider that it is possible to implement an arbitrary intermediate solution between the lazy dissemination and the epidemic-style multicast scheme simply using probabilistic repeaters that repeat certain kinds of news items with different probabilities. In the lazy case this probability was 0 for each news item while it was 1 for hot news and 0 for the rest of the news items in the epidemic multicast scheme.

B. File-Sharing Applications

The popularity of Internet-based peer-to-peer systems comes from their capabilities for sharing files between users in a completely decentralized fashion. A newscast system also allows users to easily share files. Each agent is assumed to maintain a local collection of files that it is willing to share with other agents. A collection has an associated directory in which each entry identifies and describes a single file. There are two different approaches for newscast file sharing that can be combined into a hybrid one.

In the *push approach*, the `getNews` operator of each agent passes its local directory to the news agency. Because this directory is repeatedly passed to the agency, it is guaranteed that it will eventually be delivered to all agents. However, if we assume that a local file collection changes, no guarantees can be given that when a local directory is delivered that it is still up to date.

When an agent receives an entry identifying a file it wants, it contacts the owner of that file and transfers the file out-of-band to its own local file collection (to be able to contact the owner the content field of the corresponding news item has to contain an address too). An agent can apply filtering techniques to prevent the delivery of entries it is not interested in.

The push approach is not commonly applied to modern file-sharing peer-to-peer systems. Instead of disseminating information on shared files, a user is forced to search for the files he or she wants. In a newscast system, this behavior is mimicked through a *pull approach* by which a directory of *wanted* files is passed to the news agency. When such a directory is delivered to an agent having one of the listed entries, the agent contacts the agent identified in the directory, thus allowing the latter to transfer the required files to its own local collection.

Of course, these two approaches can easily be combined: a directory can simultaneously list the files that an agent is willing to share along with those that are being requested.

An interesting aspect of the newscast model is that as soon as a file becomes popular, that is, many different collections have the same file, it is easy to find such a file. This is due to the fact that a search will soon be delivered to an agent that has the file. On the other hand, the presence of popular files will also be announced very often and may waste network resources. Therefore, when a file becomes popular, an agent should no longer announce its presence, but instead accept only requests for such a file. Determining when to stop announcing the presence of a file is subject to current research.

C. The Distributed Resource Machine

One application that is already implemented is the *distributed resource machine* (DRM) [36], [37] which we sketch briefly here. The idea is that the participating agents all offer an environment (computational resources and libraries) to perform computations. The collective consists of at least three different types of agents: the *controllers*, the *contributors* and the *observers*. The content of news items has two fields: a list of *commands* and the *resources*.

Observers are sink agents that are attached to some software that collects, organizes and displays information that is received by `newsUpdate`. These agents provide a user interface for people who want to monitor the system.

There are only a few controllers which are normally controlled directly by human system administrators. The implementation of operator `getNews` returns the commands issued by the administrator within a given time interval. These commands represent the only way the DRM can be controlled. A controller is normally

`timestamp`	creation time of news item
`content`	news content, possibly empty
`ID`	uniquely identifies agent
`address`	network address of the correspondent of the agent

TABLE II

FIELDS OF A NEWS ITEM AS STORED BY A CORRESPONDENT.

an observer as well.

The vast majority of the agents is a contributor. They behave as repeaters with respect to commands and as sources with respect to resource information. That is, operator `getNews` returns a content field that contains all commands the agent received and that have a timestamp not older than a given bound, in addition to a list of resources that are currently available to computations. This behavior results in an effective multicasting of commands and disseminating information on resources.

When running computations, one has to have access to a DRM contributor and launch the computation there. This computation is allowed to listen to `newsUpdate` as well so it can find the resources it needs and can start spreading itself accordingly. The DRM is currently implemented in Java which makes it easy to transfer code and implement certain security solutions.

V. THE NEWSCAST COMPUTING ENGINE

This section presents a fully distributed, scalable implementation of the specification of the newscast model of computation given in Section III.

A. The Core Epidemic Algorithm

Let us assume that every agent is located on a host in a wide area network. Each agent is assigned a local *correspondent* which represents the news agency from the point of view of the agent. The correspondent is run on the same host as the agent. Each correspondent has a local cache containing news items. The correspondent maintains two additional fields for each news item in the cache, the `address` and the `ID` as shown in Table II. The field `address` contains the network address of the correspondent of the agent identified by the `ID` field of the news item. The field `ID` is generated by the correspondent of the agent in question and uniquely identifies this agent. Every t_r time units each correspondent runs the algorithm from Figure 2. Figure 3 shows the overall architecture.

The algorithm is similar to the traditional push-pull epidemic protocol for spreading database updates, in our case fresh news items [26]. There is an important difference however: no correspondent knows the complete member list but only a random fraction of it. More importantly, the member list itself becomes the subject of information dissemination as well. Since the partial member list and the known news items are both represented by the news items stored in the cache, no correspondent has a copy of the complete database of news items either, each one knows a very small random part of all the news items available in the system if $n \gg c$ (which is the normal setting).

Even though time also plays an important role in the algorithm, no global time synchronization among all

```
1. Pick a random agent peer from cache such that peer is still in the
collective and is accessible.
2. cache.add(new newsItem(agent.getNews()))
3. Remove items older than ct_r from cache.
4. sendCache(peer)
5. peer_cache=receiveCache(peer)
6. agent.newsUpdate(peer_cache)
7. cache.addAll(peer_cache)
8. Keep the newest item only from each agent.
9. Keep the c freshest items according to the timestamps.
```

Fig. 2. The algorithm run by each correspondent in each t_r time units. `agent` is the local agent the correspondent serves which implements the required interface i.e. methods `getNews` and `newsUpdate`. `cache` is a list the correspondent uses for storing news items. Parameter c gives the maximal cache size. The algorithm run by the passive participant (i.e. the peer correspondent) is the same only step 1 is missing and step 5 is lifted to become step 1.

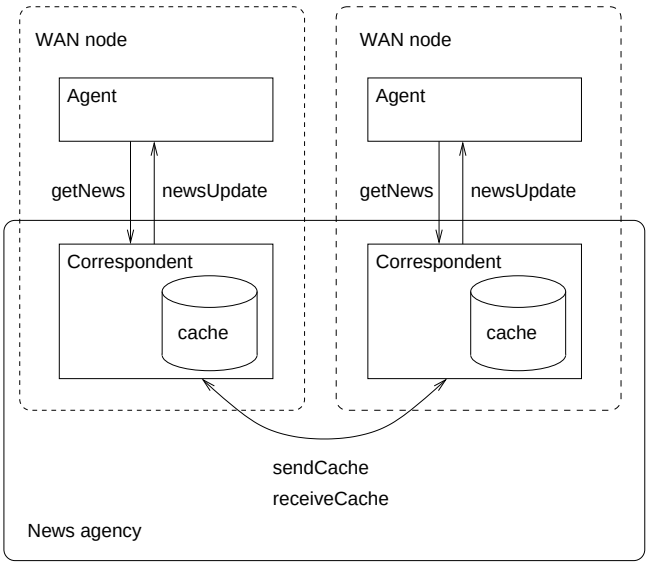

Fig. 3. The newscast computing architecture for two agents. Note that normally a huge number of agents are assumed, this figure is for illustration of the main concepts only. For finer details of the agent see Figure 1.

the correspondents is required. Keeping time consistent *within a single cache* is necessary but this can be easily achieved by always exchanging local time when exchanging the caches so that both correspondents can modify the timestamps of the received news items accordingly.

Filling in the field `ID` can be done by calculating an ID for the agent before the first call to `getNews` and then using this each time when constructing a news item. Note that it is the `ID` that identifies the agent, the address of the correspondent can change during the lifetime of an agent.

B. Membership Management

Adding a new correspondent to the news agency is done by initializing its cache with one or more known addresses of other correspondents. There are different ways to get such addresses as we explain later in

Section V-E. Removing a correspondent requires no action, in fact it is handled as a node failure. As guaranteed by step 3 the correspondent will simply be forgotten in a limited time which is compliant with item 8 of the specification of `newsUpdate` (see also Section V-F.3). Furthermore, as will be shown in Section V-G.2, the network is not sensitive to node removal so stability is not in danger either.

C. Communication Graph Series

The "knows" relation between the correspondents at time t defines a *communication digraph* (directed graph) D_t. The vertex set V_t of D_t contains the correspondents. For correspondents $a, b \in V_t$ we have $b \in out_t(a)$ iff the address of b is in the cache of a at time t, where $out_t(a)$ denotes the set of neighbors defined by the edges directed out of a. Clearly, the above algorithm defines a random series of digraphs, if started on some D_0, as information exchange sessions normally change the set of neighbors each time, and the choice of the peer is random in step 1 of the algorithm in Figure 2. Other series can be defined as well by decreasing or increasing the number of correspondents in different ways while the algorithm is running.

We define the *communication graph* G_t which we get by dropping the orientation in D_t. This graph expresses the possibility of information flow. Recall that information flow is always bidirectional independently of the initiator. In the remaining part of the paper we will mostly concentrate on the dynamic properties of G_t. In particular, we will examine the subseries G_{it_r} $(i = 0, \ldots, N)$. The motivation is that in the interval $[it_r, (i+1)t_r]$ each correspondent initiates exactly one information exchange connection. We say that in this time interval a *complete cycle* is executed.

The simulations presented in the paper make the assumptions that the information exchange sessions are run sequentially by a correspondent (i.e. the correspondent is single threaded) and each correspondent is able to finish one communication it initiates within each time interval of length t_r starting from time 0. If one information exchange session is sufficiently short relative to t_r, these requirements can be achieved. Given the relatively low guaranteed calling frequency this assumption is not unrealistic.

We will not indicate the parameters of the simulation (c, n and possibly other settings) in the graph notation as they will always be clear from the context.

D. Statistical Properties of the Communication Graph

We show that the graphs in the generated random series cannot be described by traditional random graph models. However—since many aspects of the theory of random graphs is well understood—we introduce a random graph model to provide a basis for comparison. We adopt the model introduced by [38]. The random digraph $D_{k-in,c-out}$ is defined as follows: Each vertex $v \in V$ chooses a set $in(v)$ of k random sources for edges directed into v and a set $out(v)$ of l random targets for edges directed out of v. We call such a digraph a k-in, l-out digraph. The edges are chosen without replacement so the graph has $(k+l)n$ edges where $n = |V|$. When $k = 0$ we write D_{c-out}.

In order to provide a basis for comparison, we have to be well aware of the properties of the model we chose because relatively small differences result in significant changes in behavior. For example, for D_{c-out} one has to increase l logarithmically according to $l = k + O(\log n)$ to achieve the probability limit $e^{-e^{-k}}$ for the reachability of each vertex from a specified source as $n \to \infty$ [30]. However, for $D_{k-in,k-out}$ we have strong connectivity with high probability (i.e. probability $1 - o(1)$ as $n \to \infty$) if $k \geq 2$ [38]. In other words, the key feature is that if each vertex has at least 2 incoming edges, then the number of outgoing edges

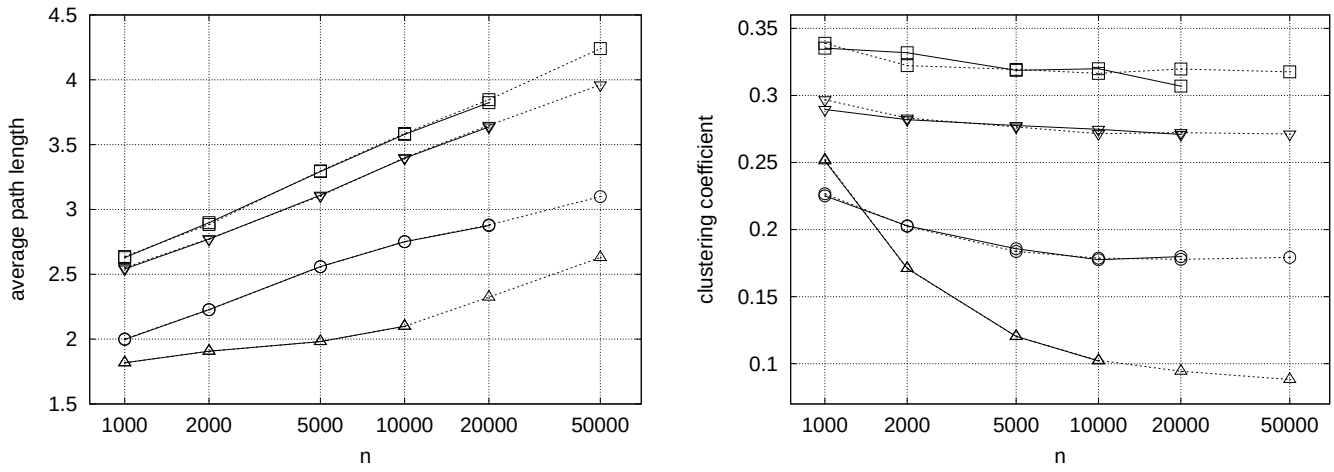

Fig. 4. Average path length from a fi xed vertex (left) and clustering coeffi cient (right) as a function of the collective size n. Points connected by a solid line are averages over a fi xed vertex in $G_{50t_r}, \ldots, G_{5000t_r}$. Points connected by a dotted line are averages over 1000 vertices in G_{50}. $G_0 = G_{c-out}$ was randomly generated. The symbols $\square$, $\circ$ and $\triangle$ correspond to $c = 20$, $c = 40$ and $c = 100$ respectively. The symbol $\triangledown$ means $c = 20$ with LTM size of 10 (see Section V-G.1)

can remain a very small constant. Note also that this means that if we drop the orientation in $D_{2-in,2-out}$ (or alternatively D_{4-out}) the resulting undirected graph G_{4-out} will be connected with high probability.

We examine here two properties which seem to be the most relevant in the context of random networks, in particular for the newscast computing engine: the average path length from a fixed vertex and the clustering coefficient. Other properties like degree distribution and connectivity are discussed later in the context of robustness.

The *average path length* from v is the average of minimal path lengths to each other vertex. If the graph is not connected this value is infinity. The *clustering coefficient* of vertex v is the fraction of pairs of neighbors of v which are also neighbors of each other. Figure 4 shows these properties for our communication graphs as a function of n and c. The average path length shows a clear logarithmic growth. This is important because the speed of information dissemination depends mainly on the minimal path lengths within the network. However, it is clearly shown that we have relatively high clustering. In the case of model G_{c-out} it is easy to show that the clustering coefficient is $1 - (1 - c/n)^2$ which is 0.002 for $n = 20000, c = 20$ for example. This tells us that the assumptions of randomness cannot be maintained, i.e. the sets $out(v)$ for neighboring vertices have a clear correlation.

These properties indicate that our communication graphs have a *small-world* structure which is defined by short minimal path lengths combined with high clustering. These sorts of topologies tend to be *emergent* topologies which result from a large number of components interacting without central control. A wide variety of phenomena and structures can be described by small-world graphs. These include the WWW link graph, the co-author graph in scientific publications, social relationships, food chains, chemical reactions in cells, etc. [33].

Another notable aspect of these results is that both properties are very stable over the whole series, i.e. they are *conserved* by the information exchange sessions. This can be seen by comparing the averages for one vertex over the series and for 1000 vertices in one graph: these values agree very closely. In other words, no matter which graph we look at from the series, we will see the same properties.

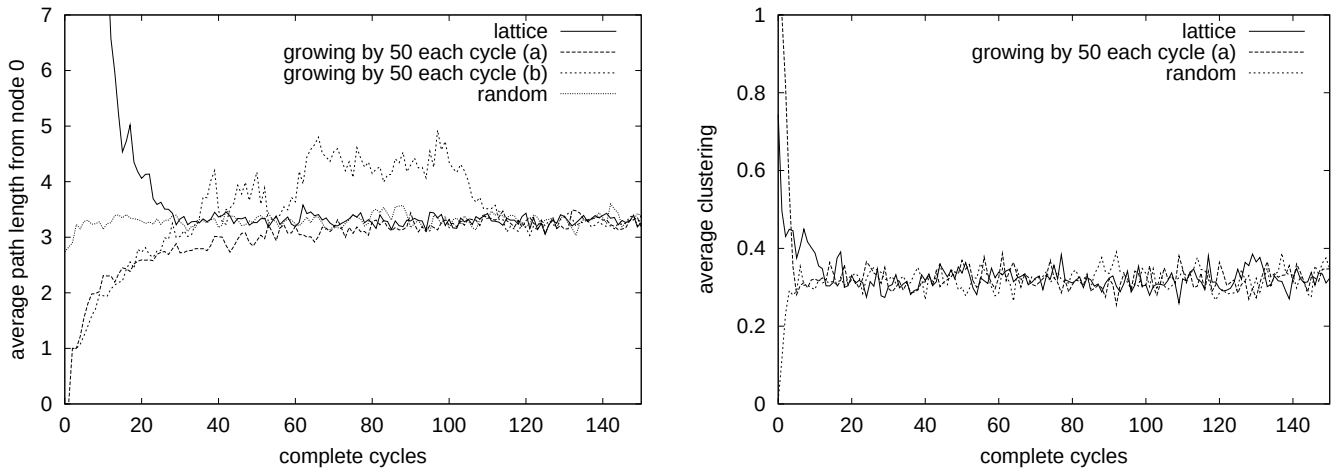

Fig. 5. Convergence with different bootstrapping methods. For all experiments $n = 5000$ and $c = 20$. The methods differ in D_0 only. For "random" it is D_{20-out}. For "lattice" it is a 1-dimensional lattice in which the vertices are ordered linearly as $v_1, \ldots, v_n$ and $out(v_i) = \{v_{j \bmod n} | j = 1, \ldots, 20\}$. For "growing" $V_0 = \{v_0\}$ and $V_{(i+1)t_r} = V_{it_r} \cup \{v_{50i+1}, \ldots, v_{50i+50}\}$ until $|V| = 5000$ with *one* edge from each new vertex to (a) a random old vertex or (b) v_0. Clustering coefficients are averages over 10 fixed vertices and growing method (b) is omitted for clarity (very similar to (a)).

E. Bootstrapping and Convergence

We will argue that the communication graph series is not sensitive to initial conditions, i.e. it is convergent. That means that our approach is not sensitive to the particular implementation of bootstrapping, i.e. the way of building an actual news agency. It also suggests that the system converges to its equilibrium state quickly if disturbed by an external event for instance. The property of convergence also validates our approach of starting all simulations with an initial random communication graph and not paying special attention to the problem of bootstrapping when discussing properties of communication graphs.

Figure 4 shows that the properties of the graph remain stable, but the first 50 cycles were ignored there and the simulation was started from D_{c-out}. In order to support the claim of convergence simulation experiments were conducted using different bootstrapping methods (see Figure 5). The less realistic is the random method while the most realistic ones are the two growing methods.

Note the rapid convergence of the clustering coefficient. Even in the case of the growing methods, the clustering coefficient reaches its final value way before reaching maximal size.

The evolution of the average path length from vertex v_0 is more interesting. The limit value is only slightly larger than the corresponding random value in spite of the large clustering (small diameter was also illustrated in Figure 4). The lattice method is started from a very large average path length of 132 which converges within 40 complete cycles.

Let us take a closer look at growing method (b) where each new vertex is connected to v_0. This is the worst possible case when we have only one server which is responsible for accepting newly added correspondents. Since there are 50 new arrivals in each cycle and we have $c = 20$ only, in each cycle the cache of v_0 is eventually filled with newcomers only, yet—instead of partitioning—this results in only a slight increase of the average path length from v_0. This is due to the fact that the newcomers that could talk to v_0 hold a link to v_0, and the first 20 newcomers hold links to the rest of the network too. When the final size is reached in cycle 100, the limit value of the average path length is restored very quickly.

Even though the protocol can cope with this worst case scenario, note that much more efficient methods can be implemented very easily, which are much closer to method (a). For example a server could easily provide large lists, much larger than c, of members from which the newcomers can select one (or more) to connect to initially. These addresses can be collected by simply adding the addresses that flow through the cache to a database and storing them for a longer time than they are stored in the cache.

F. Conformance to the Specification

Since the calling frequency of `getNews` and `newsUpdate` is the same and in the case of `getNews` this is the only constraint to satisfy, from now on we refer only to `newsUpdate`. The items in the specification which are non-trivial to meet are the calling frequency (item 3) and the statistical properties of the news updates (item 6). Even though the freshness of the received news items (item 8) is trivially satisfied by step 3 of the algorithm in Figure 2, we have to touch on the actual observed freshness during our simulations. This is necessary because step 3 could violate the size specification in item 4 if it is called if $n > c$ creating a sample smaller than both c and n.

First note that the specification assumes that the size of the collective is constant. Extreme events like serious damage (agent removal) or a significant growth have only a temporary effect on the satisfaction of the specification as our simulation results of stability suggested and as further results on robustness will suggest.

F.1 Calling Frequency

The specification defines the calling frequency in item 3. This property is important as it gives a guarantee that having a large number of connections in a short time interval has an extremely low probability, in fact practically zero. The probability of having 10 or more sessions within t_r time units is already 10^{-6} for example which means this is expected to happen once in every 115 days on average if $t_r = 10s$. For a network consisting of nodes with possibly only a low bandwidth network connection this property is essential.

The constant member of the sum $1 + \xi$ corresponds to the one actively initiated information exchange connection, and ξ to the passively received connections. We will show that under relatively weak assumptions on the communication digraph this property can be proved to hold and we also demonstrate empirically that the observed distribution is close to the required one.

Let D_t be a communication digraph randomly generated by our algorithm at time t. Let us pick a vertex a randomly and assume that for each vertex $v \neq a$ the probability $P(v \in out(a))$ is the same value, i.e. $1/(n-1)$. We also assume that the events $v \in out(a)$ $(a \neq v,\ a, b \in V_t)$ are completely independent. We can expect this assumption to hold simply due to the fact that in the algorithm all vertices have completely symmetric roles. Note that this assumption does not imply that these probabilities are the same if we take more information about the relationship between v and a into account. For instance, results on clustering suggest that neighbors of v have a higher chance to point to v, yet the assumption still holds if each vertex has an equal chance to become a neighbor of v (recall that we are considering a digraph). This assumption is therefore much weaker that the assumption that $out(v)$ is randomly drawn from the vertex set independently for each vertex.

Under this assumption, a fixed vertex v is picked by another fixed vertex with probability $1/(n-1)$. There are $n-1$ vertices that might pick v in a complete cycle, and due to the independence assumption the number of vertices that pick v (i.e. ξ) has a binomial distribution with parameters $1/(n-1)$ and $n-1$. This distribution

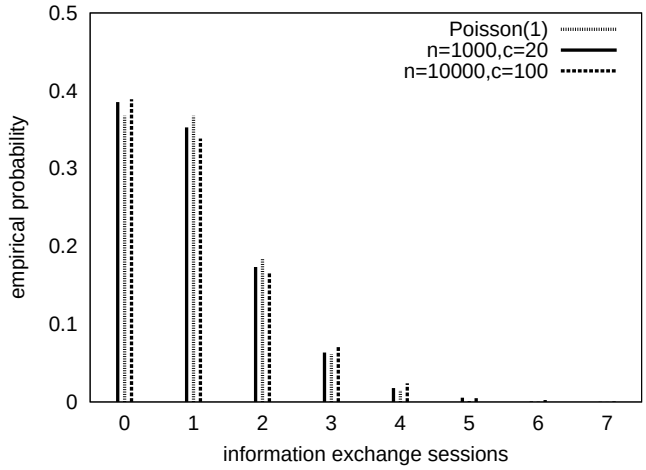

Fig. 6. Information exchange calls by other correspondents in a time interval of t_r. The empirical probabilities come from a sample of 10000, generated by recording the calls in each complete cycle to a fi xed vertex through the series $D_0, \ldots, D_{10000}$ where D_0 was randomly generated. The average (μ), empirical variance (σ^2) and maximum values for $n = 1000$ and $n = 10000$ are respectively 0.9971, 1.0966, 7 and 1.0369, 1.25586, 7.

can be approximated closely with the Poisson distribution with $\lambda = 1$ if n is large. We are interested in networks with at least $n = 1000$ which is sufficient.

Figure 6 shows the empirical distributions observed for two different collective and cache sizes plotted against the Poisson distribution. The three cases are very similar confirming that the specification for calling frequency indeed holds with an acceptable accuracy.

F.2 News Updates

Demonstrating that item 6 is fulfilled is not easy as the subset of size $c/2$ item 6 refers to is not specified. We present indirect evidence as usual for verifying randomness. For this purpose let us define the random variable $\eta_{c,n}$ (or just η_c) as the number of communication sessions until a news item from a member of an agent set of size k shows up in the news update, counted from a fixed time point. We will compare this random variable with η_c^*, the same variable with the assumption that the entire update set represents an unbiased random sample. We expect that

$$P(\eta_c^* > x) \leq P(\eta_c > x) \leq P(\eta_{c/2}^* > x) \quad x \in \mathbb{R}$$

when item 6 is satisfied and we also expect the shape of the distributions to be similar. Figure 7 shows the distribution functions of $\eta*_c$, η_c and $\eta_{c/2}^*$ for different parameter settings. The plot for different values of n is very similar so only $n = 20000$ is shown. It is clear that $P(\eta_c > x)$ is indeed bounded the way we expected and the shape of the distributions is obviously exponential. Furthermore, for $c = 100$ the observed distribution is much closer to the one which we get when the whole cache is an unbiased random sample (η_{100}^*) than to the case which is guaranteed by the specification (η_{50}^*).

After looking at the distribution functions, let us elaborate on the expected values. From the inequality of the distribution functions follows the same inequality of the expected values:

$$E(\eta_c^*) \leq E(\eta_c) \leq E(\eta_{c/2}^*)$$

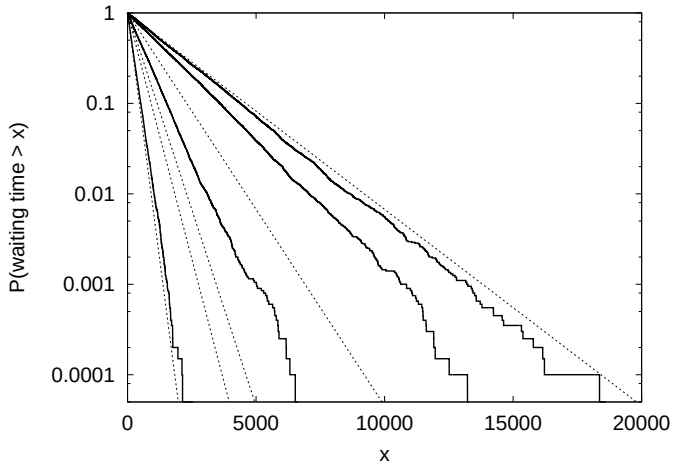

Fig. 7. Distribution functions of η_c (solid lines) and η_c^* (dotted lines) varying c with $n = 20000$ and $k = 1$ fixed. Vertical axis is log scale. The settings for η_c from left to right are $c = 100$, $c = 40$, $c = 20$ with LTM size of 10 (see Section V-G.1) and $c = 20$. The values of c for η_c^* from left to right are 100, 50, 40, 20 and 10.

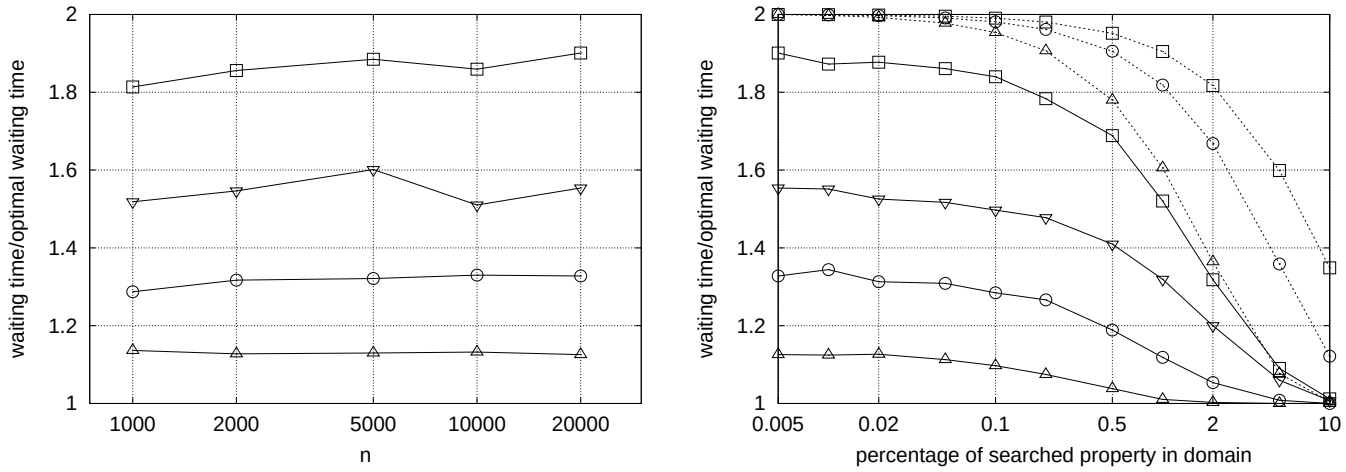

Fig. 8. Experimental values of $E(\eta_c)/E(\eta_c^*)$ (solid lines) and $E(\eta_{c/2}^*)/E(\eta_c^*)$ (dotted lines) varying n and c with $k = 1$ fixed (left) and for varying $100k/n$ and c with $n = 20000$ fixed (right). Note that $E(\eta_{k/2}^*)/E(\eta_c^*) = 2$ if $k = 1$ so dotted lines are omitted from the left graph. The symbols $\square$, $\circ$ and $\triangle$ correspond to $c = 20$, $c = 40$ and $c = 100$ respectively. The symbol $\triangledown$ means $c = 20$ with LTM size of 10 (see Section V-G.1)

Looking at the actual difference can give us more insight on the effect of the parameters c and n on the randomness of the news update. Using the independence assumption, η_c^* has a geometric distribution with $P(\eta^* = i) = (1-p)^{i-1}p$ with $E(\eta_c^*) = 1/p$ where p is the probability that a member of a set of agents of size k shows up in a given news update, and

$$p = 1 - \prod_{i=0}^{c-1} \frac{n-k-i}{n-i}$$

where we have $p = c/n$ if $k = 1$. That means $E(\eta_c^*) = 2E(\eta_{c/2}^*)$ if $k = 1$.

Figure 8 shows experimental results on the expected waiting time. We can see that waiting time is almost independent of n with a fixed k which indicates good scaling properties. Waiting time moves close to the

optimal value if the cache size is increased. This indicates that the factor $1/2$ in the constraint is not a tight bound, especially with a larger c, although a cache size of at least 20 is necessary. Another observation is that with increasing k we can also move towards more randomness. We have to add though that the figure illustrating the effect of k is slightly misleading because with increasing k $E(\eta^*_{c/2})$ also approaches $E(\eta^*_c)$. However the former is still an upper bound. Also note that for e.g. $c = 100$ we found $E(\eta_{100}) < 2.6$ if $k > 100$ (0.5%), and in particular, $E(\eta_{100}) = 1.00003$ if $k = 2000$ (10%). That is, in this region the search is not interesting anymore because the set we are searching for is represented in almost all news updates.

We can draw the conclusion that even though we have seen high clustering earlier which proved that the graphs cannot be described by a uniform random model, in this important aspect the dynamics of the newscast engine approximates the assumption of randomness with reasonable accuracy.

F.3 Freshness of News Items

As already mentioned, we have to prove that step 3 of the information exchange algorithm never violates the size specification in item 4, i.e. that it is called only if $n < c$.

In fact during any of the simulations this step never removed any news items. To give a stronger argument however, we give the statistics of the age of the oldest value in the cache using a sample of size 10000 obtained for $n = 10000$ and $c = 100$. Note that the larger c is the higher the probability is to see older news items. The largest cache we looked at was $c = 100$. In comparison with the guaranteed $100t_r$ the observed statistics are $\mu = 2.48t_r$, $\sigma^2 = 0.11t_r$ and the maximum is $3.85t_r$. This suggests that if $n \gg c$ which is the normal setting, it is very unlikely that an item will be seen which reaches the age that activates step 3.

G. Robustness

G.1 Spontaneous Partitioning and Long Term Memory

When at some point in the series G_i, $i = 0, \ldots$ a graph G_t becomes partitioned—i.e. a subgraph becomes disconnected—we talk about *spontaneous partitioning*. Since the topology constantly changes according to the information exchange algorithm it is important to understand the probability of this phenomenon as a function of parameters n and c, and also to suggest techniques to help prevent it.

We face a methodological problem, however, because for the setting used in the previous simulations we presented so far this phenomenon did not occur. We saw it only once, by accident during preliminary experiments, in a simulation with $n = 20000$ and $c = 20$ in cycle 13859 when a cluster of 32 vertices was separated. Collecting statistical data directly is therefore infeasible. For the above reason statistics for smaller graphs and caches were collected. In such settings it was possible to run long simulations and from the results we can draw some cautious conclusions extrapolating to larger values and, more importantly, we can also test the techniques which are supposed to prevent partitioning.

The results of the simulations are shown in Table III. The table shows the statistics on both the cycle index in which spontaneous partitioning happens first and the size of the smaller cluster which splits away from the network as a result of partitioning. All values correspond to 50 independent runs until partitioning or 50000 complete cycles starting with $D_0 = D_{c-out}$. The first line contains the percentages of the runs in which partitioning happened before cycle 50000. Accordingly, the values marked with a $*$ are only lower bounds, because not all runs lead to partitioning. The empirical average and variance of the size of the separated small cluster (partitioning always resulted in one large and one small component) was calculated from the runs that

	$c = 15$	$c = 16$	$c = 17$	$c = 18$
<50000 (%)	100	100	92	38
average	358	3160	>20570*	>40456*
maximum	1596	13096	>50000*	>50000*
minimum	28	29	50	3812
μ cluster size	33.24	30.08	31.02	31.21
σ^2 cluster size	114.27	87.67	63.71	75.18

TABLE III

SPONTANEOUS PARTITIONING STATISTICS FOR $n = 1000$.

actually lead to partitioning.

Even though on average the waiting time for spontaneous partitioning seems to grow very fast (for $c = 17$ and 18 we have only lower bounds and in fact we did not have the resources to be able to perform this experiment with $c = 19$), unfortunately the minimal observed waiting time does not grow fast enough to allow us to conclude that partitioning is not likely with larger values of c (although it may well be the case).

Lacking hard evidence that partitioning will not happen too often we suggest a solution for preventing partitioning instead: the *long term memory* (LTM). To implement this idea, we add another cache, the LTM, of size c_{ltm}. A new parameter is needed as well, probability p_{ltm}. The modifications of the algorithm in Figure 2 are straightforward: in step 1 `peer` is picked from the LTM instead of the cache with probability p_{ltm}, and when receiving a connection, `peer` is stored in LTM with a probability p_{ltm}. When the size of LTM exceeds c_{ltm} we remove a random element.

Performing the same simulations with $c_{ltm} = 10$ and $p_{ltm} = 0.1$ we were unable to find a single case of partitioning. So, to demonstrate the power of our approach c had to be decreased even more. For $c = 6$ without LTM G_{5t_r} is always already partitioned when started from $D_0 = D_{6-out}$ (as mentioned already, it is known that G_{6-out} is connected with high probability which is confirmed by our experiments as well). In the next couple of cycles the graph falls apart forming more than 90 disconnected clusters. Yet, adding LTM of size 10 with $p_{ltm} = 0.1$ keeps even this graph together. A simulation was run using $n = 1000$, $c = 6$ until 10^6 complete cycles. The graph is usually partitioned during the simulation, the statistics of the number of clusters during the run are the following (as calculated from the sample of size 10^6): $\mu = 1.50895, \sigma^2 = 0.506396, \max = 5$. However, after partitioning connectivity is always restored; $G_{10^6 t_r}$ is connected for example.

In general, the LTM does not interfere with the conformance to the specification, to the contrary, it moves the important properties of the communication graphs towards those of a random graph. Figure 4 shows results with $c_{ltm} = 10, c = 20, p_{ltm} = 0.1$. As can be seen clustering is decreased and the average path length is slightly reduced as well. The same effect can be seen on Figure 8 as well.

At the same time, having demonstrated the ability of the LTM to repair damage in an extremely unstable setting we can conclude that spontaneous partitioning can be restored effectively, especially considering that damage itself happens very rarely with $c \geq 20$.

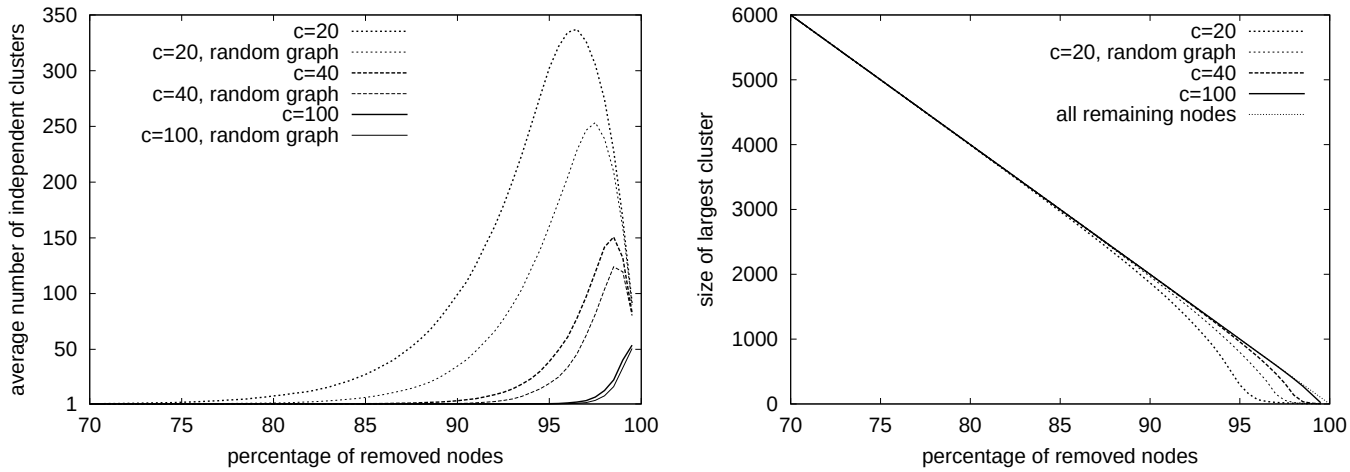

Fig. 9. Partitioning of the communication graph as a function of the percentage of removed nodes (node failures). The depicted values are averages of 50 independent experiments with $n = 20000$ fixed, each time starting with $D_0 = D_{c-out}$ and performing the simulation of node failures on G_0 (random graph) and G_{50t_r}.

G.2 Node Failures

The effect of node failures on the connectivity of the communication graphs was tested (see Figure 9). The graph shows very similar behavior to the random graph G_{c-out}, especially if the cache is large. These results indicate considerable robustness to node failures especially considering the size of the largest cluster. For $c = 100$ we can see that it is possible to remove in fact *any* number of random vertices while the vast majority of the remaining vertices still forms a single connected cluster.

Another important aspect is degree distribution. In particular, we know that if the network is *scale-free*, i.e. if its degree distribution follows a power law, then even though the graph is more robust to *random* vertex removal than G_{c-out}, it is more vulnerable to the removal of vertices with a high degree [33]. Figure 10 shows the degree distribution for a communication graph. We can see that the distribution is clearly non-linear on the log-log scale which would be the case with the power law (although its tail is heavier than that of the random distribution). Consider also that we have a graph series and not a single graph. It was also tested that within this series the degree of a fixed vertex keeps changing over the whole range of the distribution which makes it extremely unlikely in practice that many vertices with a large degree are removed at the same time.

VI. Conclusions

In this paper we have introduced the newscast model of computation. The specification of the model can be implemented in a fully distributed fashion. The key component is a lazy probabilistic information dissemination technique which is responsible for membership management and communication. This technique serves as the engine of the newscast model of computation which supports many different types of applications including efficient reliable multicasting, resource sharing, monitoring and controlling of large systems and possibly—as a future result of our ongoing research—computational intelligence, distributed datamining and modeling of social phenomena.

A major contribution of the paper is the protocol for information dissemination which is extremely scalable and robust and also provides us with a continuously changing communication graph over the set of correspondents which has useful properties like small diameter and a sufficient level of randomness in its neighborhood

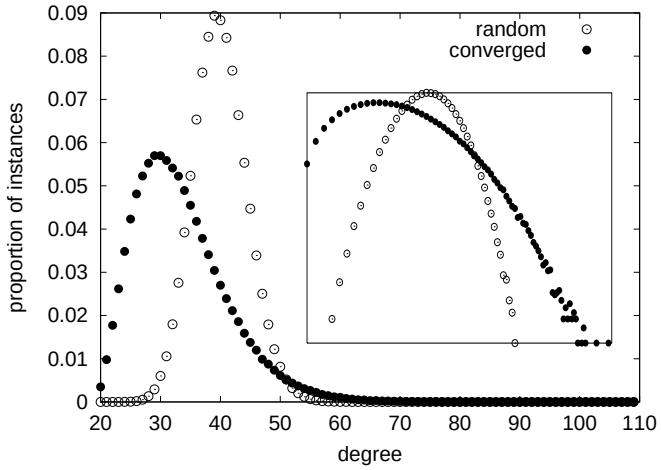

Fig. 10. Degree distribution on linear and log-log scale (inset) using the same range. The depicted values are averages of 50 independent experiments with $n = 20000$ and $c = 20$ fixed, each time starting with $D_0 = D_{c-out}$ and collecting data from G_{50t_r} and G_0.

structure. These properties are *emergent*, i.e. they are achieved not by explicitly trying to increase randomness using a special purpose protocol but as a natural consequence of the interaction of the correspondents which is based on a very simple protocol that uses only local information.

Another consequence of this approach based on emergent behavior is that we do not need special protocols for adding or removing correspondents either. For addition only the minimal information is necessary, i.e. an address of a member and for removal no action is necessary at all beside stopping communication.

It has been demonstrated that the specifications of the model are indeed fulfilled by the proposed information propagation approach. It has been shown that the system converges to the same state from very different starting conditions. The system is also robust to node failures due to the random characteristics of the emergent communication graph.

ACKNOWLEDGMENTS

The authors would like to thank the members of the DREAM project for fruitful discussions, the early pioneers [9] as well as the rest of the DREAM staff, Maribel García Arenas, Emin Aydin, Pierre Collet and especially Mike Preuß with whom we worked on several aspects of the earlier versions of the framework. This work is funded as part of the European Commission Information Society Technologies Program (Future and Emerging Technologies). The authors have sole responsibility for this work, it does not represent the opinion of the European Community, and the European Community is not responsible for any use that may be made of the data appearing herein.

REFERENCES

[1] Andrew S. Tanenbaum and Maarten van Steen, *Distributed Systems, Principles and Paradigms*, Prentice Hall, Upper Saddle River, N.J., 2002.

[2] ISO, "Open distributed processing reference model," International Standard ISO/IEC IS 10746, 1995.

[3] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano, "PGM reliable transport protocol specification," RFC 3208, Dec. 2001.

[4] Brian Neil Levine and Jose J. Garcia-Luna-Aceves, "A comparison of reliable multicast protocols," *Multimedia Systems*, vol. 6, no. 5, pp. 334–348, 1998.

[5] Katia Obraczka, "Multicast transport protocols: A survey and taxonomy," *IEEE Communications Magazine*, vol. 36, no. 1, pp. 94–102, Jan. 1998.

[6] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky, "Bimodal multicast," *ACM Transactions on Computer Systems*, vol. 17, no. 2, pp. 41–88, May 1999.

[7] Meng-Jang Lin and Keith Marzullo, "Directional gossip: Gossip in a wide area network," in *Dependable Computing – EDCC-3*, Jan Hlavička, Erik Maehle, and András Pataricza, Eds. 1999, vol. 1667 of *Lecture Notes on Computer Science*, pp. 364–379, Springer-Verlag.

[8] Patrick Th. Eugster and Rachid Guerraoui, "Probabilistic multicast," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, Los Alamitos, CA., June 2002, IEEE, pp. 313–322, IEEE Computer Society Press.

[9] Ben Paechter, Thomas Bäck, Marc Schoenauer, Michele Sebag, A. E. Eiben, Juan J. Merelo, and Terry C. Fogarty, "A distributed resoucre evolutionary algorithm machine (DREAM)," in *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*. IEEE, 2000, pp. 951–958, IEEE Press.

[10] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli, "Mobile-agent coordination models for Internet applications," *IEEE Computer*, vol. 33, no. 2, pp. 82–89, Feb. 2000.

[11] David Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[12] Mark E. J. Newman, "Models of the small world," *Journal of Statistical Physics*, vol. 101, no. 3-4, pp. 819–841, Nov. 2000.

[13] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, Upper Saddle River, N.J., 4th edition, 2003.

[14] David R. Cheriton, "Dissemination-oriented communication systems," Computer Science Department, Stanford University, 1992.

[15] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, "The many faces of publish/subscribe," Tech. Rep. DSC ID:200104, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, Jan. 2001.

[16] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra, "Matching events in a content-based subscription system," in *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC'99)*, Atlanta, GA, Apr. 1999, ACM, pp. 53–61.

[17] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman, "An efficient multicast protocol for content-based publish-subscribe systems," in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, Austin, TX, June 1999, IEEE, pp. 262–272.

[18] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[19] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, San Diego, CA, 2001, ACM, pp. 149–160, ACM Press.

[20] Antony Rowstron and Peter Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Middleware 2001*, Rachid Guerraoui, Ed. 2001, vol. 2218 of *Lecture Notes in Computer Science*, pp. 329–350, Springer-Verlag.

[21] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.

[22] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker, "A scalable content-addressable network," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, San Diego, CA, 2001, ACM, pp. 161–172, ACM Press.

[23] Steven D. Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suciu, "What can databases do for peer-to-peer?," in *Proceedings of the 4th International Workshop on the Web and Databases (WebDB'2001)*, May 2001.

[24] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon T. Loo, Scott Shenker, and Ion Stoica, "Complex queries in DHT-based peer-to-peer networks," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Mar. 2002.

[25] Divyakant Agrawal, Amr El Abbadi, and Robert C. Steinke, "Epidemic algorithms in replicated databases," in *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, Tucson, AZ, May 1997, ACM, pp. 161–172.

[26] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry, "Epidemic algorithms for replicated database management," in *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, Vancouver, Aug. 1987, ACM, pp. 1–12.

[27] Michael Rabinovich, Narain H. Gehani, and Alex Kononov, "Scalable update propagation in epidemic replicated databases," in *Advances in Database Technology - EDBT'96*, Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, Eds. 1996, vol. 1057 of *Lecture Notes in Computer Science*, pp. 207–222, Springer.

[28] Robbert van Renesse, Yaron Minsky, and Mark Hayden, "A gossip-style failure detection service," in *Middleware '98*, Nigel Davies, Kerry Raymond, and Jochen Seitz, Eds. 1998, pp. 55–70, Springer.

[29] Sridharan Ranganathan, Alan D. George, Robert W. Todd, and Matthew C. Chidester, "Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters," *Cluster Computing*, vol. 4, no. 3, pp. 197–209, July 2001.

[30] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh, "Probablistic reliable dissemination in large-scale systems," *IEEE Transactions on Parallel and Distributed Systems*, 2003, To appear.

[31] Béla Bollobás, *Random Graphs*, Cambridge University Press, Cambridge, UK, 2nd edition, 2001.

[32] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié, "Peer-to-peer membership management for gossip-based protocols," *IEEE Transactions on Computers*, 2003, To appear.

[33] Réka Albert and Albert-László Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, pp. 47–97, Jan. 2002.

[34] David Mosberger, "Memory consistency models," *ACM SIGOPS Operating Systems Review*, vol. 27, no. 1, pp. 18–26, Jan. 1993.

[35] Sarita V. Adve and Kourosh Gharachorloo, "Shared memory consistency models: A tutorial," *IEEE Computer*, vol. 29, no. 12, pp. 66–76, Dec. 1996.

[36] Maribel G. Arenas, Pierre Collet, A. E. Eiben, Márk Jelasity, Juan J. Merelo, Ben Paechter, Mike Preuß, and Marc Schoenauer, "A framework for distributed evolutionary algorithms," in *Parallel Problem Solving from Nature - PPSN VII*, Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villacañas, and Hans-Paul Schwefel, Eds. 2002, vol. 2439 of *Lecture Notes in Computer Science*, pp. 665–675, Springer-Verlag.

[37] DR-EA-M Project, "http://www.dr-ea-m.org/,".

[38] Trevor I. Fenner and Alan M. Frieze, "On the connectivity of random m-orientable graphs and digraphs," *Combinatorica*, vol. 2, pp. 347–359, 1982.