Author's personal copy

Computer Networks 53 (2009) 2321–2339

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

T-Man: Gossip-based fast overlay topology construction ☆

Márk Jelasity [a,*], Alberto Montresor [b], Ozalp Babaoglu [c]

[a] Research Group on AI, University of Szeged and HAS, P.O. Box 652, H-6701 Szeged, Hungary
[b] Dipartmento di Ingegneria e Scienza dell'Informazione, University of Trento, via Sommarive 14, I-38050 Povo (TN), Italy
[c] Dipartimento di Scienze dell'Informazione, University of Bologna, mura Anteo Zamboni 7, I-40126 Bologna, Italy

ARTICLE INFO

Article history:
Available online 1 April 2009

Keywords:
Gossip-based protocols
Overlay networks
Bootstrapping
Self-organizing middleware

ABSTRACT

Large-scale overlay networks have become crucial ingredients of fully-decentralized applications and peer-to-peer systems. Depending on the task at hand, overlay networks are organized into different topologies, such as rings, trees, semantic and geographic proximity networks. We argue that the central role overlay networks play in decentralized application development requires a more systematic study and effort towards understanding the possibilities and limits of overlay network construction in its generality. Our contribution in this paper is a gossip protocol called T-Man that can build a wide range of overlay networks from scratch, relying only on minimal assumptions. The protocol is fast, robust, and very simple. It is also highly configurable as the desired topology itself is a parameter in the form of a ranking method that orders nodes according to preference for a base node to select them as neighbors. The paper presents extensive empirical analysis of the protocol along with theoretical analysis of certain aspects of its behavior. We also describe a practical application of T-Man for building Chord distributed hash table overlays efficiently from scratch.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Overlay networks have emerged as perhaps the single-most important abstraction when implementing a wide range of functions in large, fully-decentralized systems. The overlay network needs to be designed appropriately to support the application at hand efficiently. For example, application-level multicast might need carefully controlled random networks or trees, depending on the multicast approach [1,2]. Similarly, decentralized search applications benefit from special overlay network structures such as random or scale-free graphs[3,4], super-peer networks [5], networks that are organized based on proximity and/or capacity of the nodes [6,7], or distributed hash tables (DHT-s), for example, [8,9].

In current work, protocol designers typically assume that a given network exists for a long period of time, and only a relatively small proportion of nodes join or leave concurrently. Furthermore, applications either rely on their own idiosyncratic procedures for implementing join and repair of the overlay network or they simply let the network evolve in an emergent manner based on external factors such as user behavior.

We believe that there is room and need for interesting research contributions on at least two fronts. The first concerns the question whether a single framework can be used to develop flexible and configurable protocols without sacrificing simplicity and performance to tackle the plethora of overlay networks that have been proposed. The second front concerns scenarios in overlay construction that are often overlooked, such as massive joins and leaves, as well as quick and efficient bootstrapping of a desired overlay from scratch or some initial state. Current approaches either fail

☆ This work was completed while the authors were with the University of Bologna, Italy

* Corresponding author. Tel.: +36 62 544127.
E-mail addresses: jelasity@inf.u-szeged.hu (M. Jelasity), montreso@dit.unitn.it (A. Montresor), babaoglu@cs.unibo.it (O. Babaoglu).
URL: http://www.inf.u-szeged.hu/~jelasity/ (M. Jelasity).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.03.013

Author's personal copy

or are prohibitively expensive in such scenarios. Combining results on these two fronts would enable several interesting possibilities. These include: (i) overlay network creation *on demand*, (ii) deployment of temporary and adaptive decentralized applications with custom overlay topologies that are designed on-the-fly, (iii) federation or splitting of different existing architectures [10].

In this paper we address both questions and present an algorithm called T-Man for creating a large class of overlay networks from scratch. The algorithm is highly configurable: the network to be created is defined compactly by a *ranking method*. The ranking method formalizes the following idea: when shown a set of nodes, we assume each node in the network is able to decide which ones it likes from the set more and which ones it likes less (we will later use this ability of nodes to help them have neighbors they like as much as possible). In other words, each node can order any set of nodes. Formally speaking, the ranking method is able to order any set of nodes given a so called *base node*. By defining an appropriate ranking method, we will be able to build a wide variety of topologies, including sorted rings, trees, toruses, clustering and proximity networks, and even full-blown DHT networks, such as the Chord ring with fingers. T-Man relies only on an underlying peer sampling service [11] that creates an initial overlay network with random links as the starting point.

The algorithm is gossip-based: all nodes periodically communicate with a randomly-selected neighbor and exchange (bounded) neighborhood information in order to improve the quality of their own neighbor set. This approach, while requiring no more messages than the heartbeats already present in proactive repair protocols, is simple, and achieves fast and robust convergence as we demonstrate.

In this paper we limit our study to the overlay construction problem. Using T-Man for overlay maintenance is also possible [12] with performance and cost that are not dramatically different from existing periodic repair protocols currently used in most overlay networks. The originality and attractiveness of T-Man as a maintenance protocol lies in its generality and configurability. The main contribution of this paper is to show that a single, generic gossip-based algorithm can *create* many different overlay networks *from scratch* quickly and efficiently.

1.1. Related work

Related work in bootstrapping include the algorithm of Voulgaris and van Steen [13] who propose a method to jump-start Pastry [9]. This protocol is specifically tailored to Pastry and its message complexity is significantly higher than that of T-Man. More recently, the bootstrapping problem has been addressed in other specific overlays [14–16]. These algorithms, although reasonably efficient, are specific to their target overlay networks.

An approach closer to T-Man is Vicinity, described in [17]. Although Vicinity was inspired by the earliest version of T-Man, it does contain notable original components related to overlay maintenance, such as churn management, and other techniques to boost performance.

Finally, we mention related work that use gossip-based probabilistic and lightweight algorithms. We note that these algorithms are targeted neither at efficient bootstrapping, nor at generic topology management. Massoulié and Kermarrec [18] propose a protocol to evolve a topology that reflects proximity. More recent protocols applying similar principles include [19] and [20]. Repair protocols used extensively in many DHT overlays also belong to this category (e.g., [8,21,22]).

1.2. Contribution

Our contribution with respect to related work is threefold. First, we introduce a lightweight probabilistic protocol that can construct a wide range of overlay networks based on a compact and intuitive representation: the ranking method. The protocol has a small number of parameters, and relies on minimal assumptions, such as nodes being able to obtain a random sample from the network (the peer sampling service). The protocol is an improved and simplified version of earlier variants presented at various workshops [12,23,10]. Second, we develop novel insights for the tradeoffs of parameter settings based on an analogy between T-Man and epidemic broadcasts. We describe the dynamics of the protocol considering it as an epidemic broadcast, restricted by certain factors defined by the parameters and properties of the ranking method (that is, the properties of the desired overlay network). We also analyze storage complexity. Third, we present novel algorithmic techniques for initiating and terminating the protocol execution. We describe how to construct the Chord overlay as a practical application of T-Man. We present extensive simulation results that support the efficiency and reliability of T-Man.

1.3. Road map

Sections 2 and 3 present the system model and the overlay construction problem. Section 4 describes the T-Man protocol. In Section 5 we present theoretical and experimental results to characterize key properties of the protocol and to give guidelines on parameter settings. Section 6 presents practical extensions to the protocol related to bootstrapping and termination, and extensive experimental results are also given to examine the behavior of the protocol in different failure scenarios. Section 7 presents a practical application: the creation of the Chord overlay network [8]. Section 8 concludes the paper.

2. System model

We consider a set of nodes connected through a routed network. Each node has an address that is necessary and sufficient for sending it a message. Furthermore, all nodes have a *profile* containing any additional information about the node that is relevant for the definition of an overlay network. Node ID, geographical location, available resources, etc. are all examples of profile information. The address and the profile together form the *node descriptor*. At times, we will use "node descriptor" and "node" interchangeably if this does not cause confusion.

Author's personal copy

The network is highly dynamic; new nodes may join at any time and existing nodes may leave, either voluntarily or by *crashing*. Our approach does not require any mechanism specific to leaves: spontaneous crashes and voluntary leaves are treated uniformly. Thus, in the following, we limit our discussion to node crashes. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion.

We assume that nodes are connected through an existing routed network, where every node can potentially communicate with every other node. To actually communicate, a node has to know the address of the other node. This is achieved by maintaining a *partial view* (*view* for short) at each node that contains a set of node descriptors. Views can be interpreted as sets of edges between nodes, naturally defining a directed graph over the nodes that determines the topology of an *overlay network*.

Communication incurs unpredictable delays and may be subject to failures. Single messages could lost, links between pairs of nodes may break. Nodes have access to local clocks that can measure the passage of real time with reasonable accuracy, that is, with small short-term drift. Local clocks are not required to be synchronized.

Finally, we assume that all nodes have access to the peer sampling service [11] that returns random samples from the set of nodes in question. From a theoretical point of view we will assume that these samples are indeed random. From a practical point of view, results in [11] as well as our own experimental results in this paper indicate that the peer sampling service indeed has suitable realistic implementations that provide high quality samples at a low cost.

3. The overlay construction problem

Intuitively, we are interested in constructing some desirable overlay network, possibly from scratch, by filling the views at all nodes with descriptors of the appropriate neighbors. For example, we might want to organize the nodes into a ring where the nodes appear in increasing order based on their ID. Or we might want to construct a proximity network, where the neighbors of a node are those that are closest to it according to some metric.

We allow for arbitrary initial content of the views of the nodes in this problem definition (including empty views), noting that, as mentioned in our system model, nodes have access to random samples from the network, so they have access to at least random nodes from the network. In other words, starting from any arbitrary network, we want to fill the node views with the appropriate neighbors as fast as possible at a reasonable cost.

In order to have a well defined problem, we need to specify how the desired overlay is represented as an input to the protocol. The representation must be compact, intuitive, yet descriptive enough to capture the widest possible range of topologies.

Our proposal for the representing the desired overlay is the *ranking method*. As explained before, the ranking method sorts a set of nodes (potential neighbors) according to the "taste" of a given base node. More formally, the input of the problem is a set of N nodes, the *target view size* K (bounded by N) and a *ranking method* RANK. The ranking method takes as parameters the base node x and a set of nodes $\{y_1, \ldots, y_j\}$, $j \leqslant N$, and outputs an ordered list of these j nodes. All nodes in the network apply the same ranking method, which they are assumed to know a priori. Throughout the paper, we will analyze and test only ranking methods that are based on a partial ordering of the given set, and that return some total ordering consistent with this partial ordering (note however, that this is not an inherent restriction). Accordingly, we allow for an element of uncertainty (if there can be many total orderings consistent with the partial ordering we pick a random one).

A *target graph* that we wish to construct is defined by the ranking method. We present the definition of a target graph in a constructive way, through the following (inefficient) approach, for illustration. In this approach, each node disseminates its descriptor to all other nodes such that eventually, every node has collected locally the descriptor of every node in the network. At this point, each node sorts this set of descriptors according to the ranking method and picks the first K elements to be its neighbors. The resulting structure is called a *target graph*. Note that in this manner we define a graph, and not only a topology, because in addition to knowing the structure of the network, such as a ring, we also know the exact location of each node in the structure.

A practical solution to the *overlay construction problem* has to significantly reduce both the communication cost (which is at least linear in N for each node) and the storage cost (which is also linear in N for each node) of the full dissemination approach outlined above in building the target graph. The T-MAN protocol described in the next section does precisely this.

Although representing the target graph through the ranking method and parameter K clearly restricts the scope of the algorithm, through the examples presented here and in the rest of this paper we will see that a wide range of interesting applications are covered. One (but not the only!) way of actually defining useful ranking methods is through a distance function that defines a metric space over the set of nodes. The ranking method can simply return an ordering of the given set according to non-decreasing distance from the base node.

To clarify the notions of ranking method and target graphs, let us consider a few simple examples, where $K = 2$ and the profile of a node is a real number in the interval $[0, M[$. We can define a ranking method based on the one-dimensional distance function between nodes a and b as $d(a,b) = |a-b|$, or alternatively, $d(a,b) = \min(M - |a-b|, |a-b|)$ to obtain a circular structure. As illustrated in Fig. 1a, if the node profiles are more-or-less uniformly distributed over the interval $[0, M[$, the resulting target graph will be a connected line (or ring). If the node profiles are not evenly distributed over $[0, M[$ but are clustered, the same ranking method will result in a target graph that consist of disconnected clusters (Fig. 1b).

It is important to note that there are target graphs of practical interest that cannot be defined through a global distance function. This is the main reason for using ranking

Author's personal copy

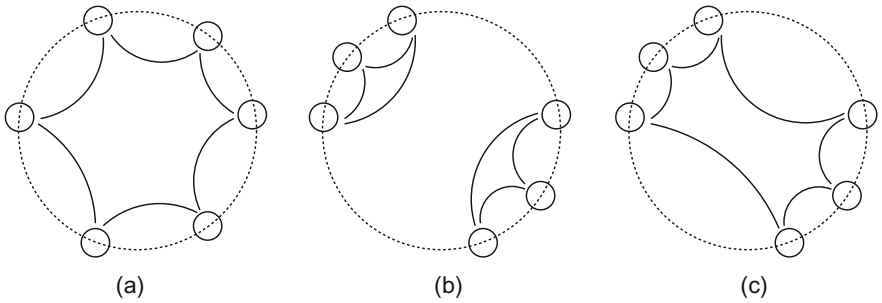


Fig. 1. Target graphs for different ranking methods and $K = 2$. (a) One-dimensional distance-based, circular ranking method applied to a set of uniform node profiles; (b) same ranking method as before but with a different set of node profiles that are clustered; (c) direction-dependent ranking method achieves sorting even for clustered node profiles.

methods, as opposed to relying exclusively on the notion of distance; the ranking method is a more general concept than distance. This fact will become important in Section 7 (practical application example), where it is necessary to be able to build, for example, a ring, even in the case of uneven node descriptor distributions when distance-based ranking methods would define clustered target graphs (as in Fig. 1b). Fig. 1c illustrates how a direction-dependent ranking can be used to avoid clustering in the target graph. Here, the output of the ranking method $\text{RANK}(x, \{y_1, \ldots, y_j\})$ is defined as follows. We first construct a sorted ring out of the set of input profiles $y_1, \ldots, y_j$ and the base node x. We then assign a rank value to each node defined as the minimal hop count to the node from x in this ring. The output of the ranking method is a list of the input profiles ordered according to this rank value. In this manner, the first 2α positions in the ranking contain α nodes preceeding x and α nodes following x in the sorted ring; hence the name "direction-dependent"

```
1: loop
2:     wait(Δ)
3:     p ← selectPeer(ψ, rank(myDescriptor, view))
4:     buffer ← merge(view, {myDescriptor})
5:     buffer ← rank(p, buffer)
6:     send first m entries of buffer to p
7:     receive buffer_p from p
8:     view ← merge(buffer_p, view)
```

(a) active thread

```
1: loop
2:     receive buffer_q from q
3:     buffer ← merge(view, {myDescriptor})
4:     buffer ← rank(q, buffer)
5:     send first m entries of buffer to q
6:     view ← merge(buffer_q, view)
```

(b) passive thread

Fig. 2. The T-Man protocol.

4. The T-Man protocol

As mentioned earlier, the T-Man protocol is based on a gossiping scheme, in which all nodes periodically exchange node descriptors with peer nodes, thereby constantly improving the set of nodes they know — their partial views.

Each node executes the protocol in Fig. 2. Any given view contains the descriptors of a set of nodes. Method MERGE is a set operation in the sense that it keeps at most one descriptor for each node. Parameter m denotes the message size as measured in the number of node descriptors that the message can hold. Method SELECTPEER selects a random sample among the first ψ entries in the ordered list given as its second parameter.

In this section we do not specify how node views are initialized. In the rest of the paper, we always describe the particular node view initialization procedure that we assume. These procedures include random initialization for the purposes of theoretical analysis in Section 5 and practical solutions based on various broadcasting schemes and realistic random peer sampling in Section 6.

We note that the protocol does not place a limit on the view size. This is done in order to decrease the number of parameters, thereby simplifying the presentation. One might expect that lack of a limit on view size might present scalability problems due to views growing too large. As we will show in Section 5, however, the storage complexity of nodes due to views grows only logarithmically as a function of the network size. Furthermore, preliminary experiments for the applications we consider show that imposing a comfortable limit on view sizes (larger than both m and K) does not result in any observable decrease in performance. This suggests that the simplification of ignoring view size limits is justified and is not critical for these applications.

Although the protocol is not round based at the global level, it is often convenient to refer to *cycles* of the protocol execution in the network. We define a cycle to be an interval of Δ time units where Δ is another parameter of the protocol in Fig. 2.

Fig. 3 illustrates the results of T-Man for constructing a small torus (visualizations were obtained using [24]). For this example, it is clear that only a few cycles are sufficient for convergence, and the target graph is already evident even after the first few cycles. In the next sections we will show that this rapid convergence is not unique to the torus example but that T-Man performs well in a wide range of settings and that it is scalable, very similarly to epidemic broadcast protocols.

Author's personal copy

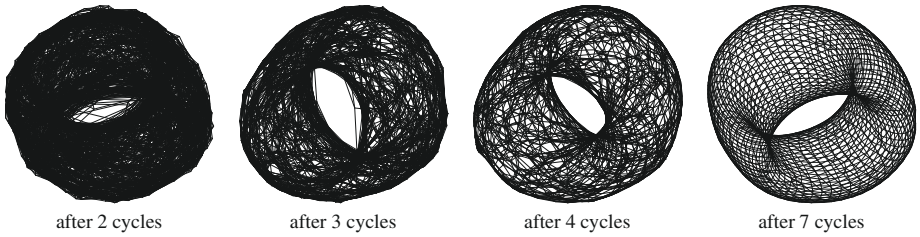


Fig. 3. Illustration of constructing a torus over $50 \times 50 = 2500$ nodes, starting from a uniform random graph with initial views containing 20 random entries and the parameter values $m = 20, \psi = 10, K = 4$.

Table 1
Parameters of the T-MAN protocol.

RANK()	*Ranking method*: determines the preference of nodes as neighbors of a base node
Δ	*Cycle length*: sets the speed of convergence but also the communication cost
ψ	*Peer sampling parameter*: peers are selected from the ψ most preferred known neighbors
m	*Message size*: maximum number of node descriptors that can be sent in a single message

In Table 1 we summarize the parameters of the protocol. Note that K (target view size) is not a parameter of the protocol but is part of the target graph characterization. As such, it controls the size of the target graph, and consequently, affects the running time of the protocol. For example, if we increase K while keeping the ranking method fixed, then the protocol will take longer to converge since it has to find a larger number of links. In fact, K could be omitted if the target graph was defined in some other, more complex manner.

5. Key properties of the protocol

In this section we study the behavior of our protocol as a function of its parameters, in particular, m (message size), ψ (peer sampling parameter) and the ranking method RANK. Based on our findings, we will extend the basic version of the peer selection algorithm with a simple "tabu-list" technique as described below. Furthermore, we analyze the storage complexity of the protocol and conclude that on the average, nodes need $O(\log N)$ storage space where N is the network size.

To be able to conduct controlled experiments with T-MAN on different ranking methods, we first select a graph instead of a ranking method, and subsequently "reverse-engineer" an appropriate ranking method from this graph by defining the ranking to be the ordering consistent with the *minimal path length* from the base node in the selected graph. We will call this selected graph the *ranking graph*, to emphasize its direct relationship with the ranking method.

Note that the target graph is defined by parameter K, so the target graph is identical to the ranking graph only if the ranking graph is K-regular. However, for convenience, in this section we will not rely on K because we either focus on the dynamics of convergence (as opposed to convergence time), which is independent of K, or we study the discovery of neighbors in the ranking graph directly.

In order to focus on the effects of parameters, in this section we assume a greatly simplified system model where the protocol is initiated at the same time at all nodes, where there are no failures, and where messages are delivered instantly. While these assumptions are clearly unrealistic, in Section 6 we show through event-based simulations that the protocol is extremely robust to failures, asynchrony and message delays even in more realistic settings.

5.1. Analogy with the anti-entropy epidemic protocol

In Section 3 we used an (unspecified) dissemination approach to define the overlay construction problem. Here we would like to elaborate on this idea further. Indeed, the anti-entropy epidemic protocol, one implementation of such a dissemination approach, can be seen as a special case of T-MAN, where the message size m is unlimited (i.e., $m \geqslant N$ such that every possible node descriptor can be sent in a single message) and peer selection is uniform random from the entire network. In this case, independent of the ranking method, all node descriptors that are present in the initial views will be disseminated to all nodes. Furthermore, it is known that full convergence is reached in less than logarithmic time in expectation [25].

For this reason, the anti-entropy epidemic protocol is important also as a base case protocol when evaluating the performance of T-MAN, where the goal is to achieve similar convergence speed to anti-entropy, but with the constraint that communication is limited to exchanging a constant amount of information in each round. Due to the communication constraint, performance will no longer be independent of the ranking method.

5.2. Parameter setting for symmetric target graphs

We define a symmetric target graph to be one where all nodes are interchangeable. In other words, all nodes have identical roles from a topological point of view. Such graphs are very common in the literature of overlay networks. The behavior of T-MAN is more easily understood on symmetric graphs, because focusing on a typical (average) node gives a good characterization of the entire system.

We will focus on two ranking graphs, both undirected: the ring and a k-out random graph, where k random out-links are assigned to all nodes and subsequently the directionality of the links is dropped. We choose these two graphs to study two extreme cases for the network diameter. The diameter (longest minimal path) of the ring is $O(N)$

Author's personal copy

while that of the random graph is $O(\log N)$ with high probability.

Let us examine the differences between realistic parameter settings and the anti-entropy epidemic dissemination scenario described above. First, assume that the message size m is a small constant rather than being unlimited. In this case, the random peer selection algorithm is no longer appropriate: if a node i contacts peer j that ranks low with i as the base node, then i cannot expect to learn new useful links from j because now (due to the small m) node j has a strong bias in its view towards nodes that rank high with j as a base node.

On the other hand, if a node i selects peers that rank too high with i as the base node, then convergence might slow down as well. The reason for this is that consecutive peers returned by the peer selection method will more often get repeated; in part because a node i is more likely to select a peer to communicate with that selected i shortly before, and in part because there are simply fewer nodes that are "close" to any given node than nodes that are far from it. This in turn results in increased correlation between the partial views of communicating partners, so the epidemic process is not maximally efficient.

Fig. 4 illustrates this tradeoff using two ranking graphs: the ring and a random graph. The latter is generated by first constructing a 2-out directed regular random graph by selecting two random out-edges for each node, and subsequently taking the undirected version of this graph. The average degree of a node is thus 4, with a small variance. The basic version in Fig. 4a applies the peer selection algorithm which picks a random peer from the highest ranking ψ nodes from the view, as described earlier. The point $\psi = N$ and $m = N$ corresponds to an anti-entropy epidemic dissemination (i.e., peer selection is unbiased and there are no limits on message size) which is optimal.

As predicted, with no limits on the message size ($m = N$), we can observe the effect due to the lack of randomness if the selected peer ranks too high (ψ is small). Furthermore, for large ψ performance again degrades when we place a limit on the message size since the correlation between communicating peers' ranking of the same set of nodes is reduced. This effect is less pronounced for larger m because now we might obtain useful information by chance even if there is little correlation between the rankings.

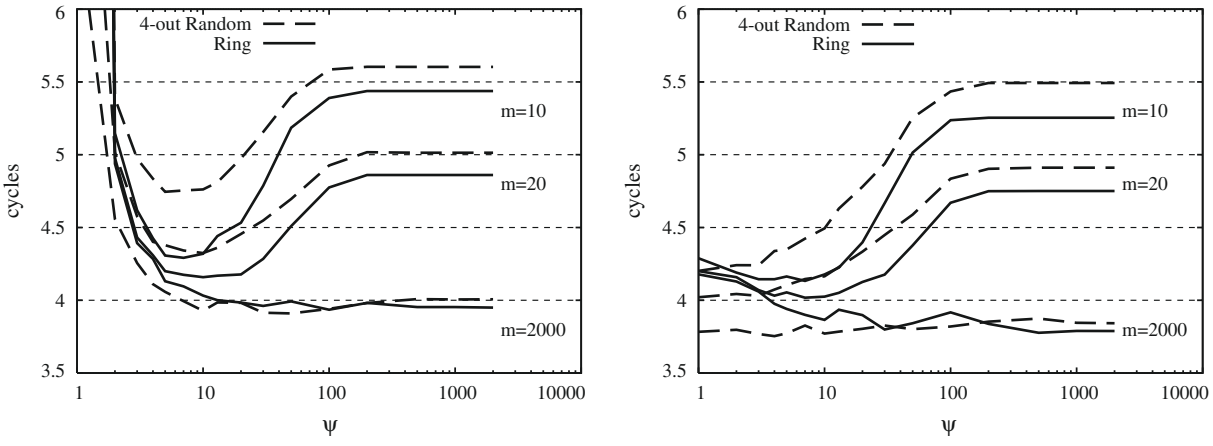
To verify our explanation as to why performance degrades with decreasing ψ, we apply a *tabu list* at all nodes in order to avoid contacting the same peers over and over again. The tabu list contains a fixed number of peers that a given node communicated with most recently. The node then does not initiate connection with any nodes in its tabu list. We experimented with a tabu list size of 4. This mechanism does not add any communication overhead since it simply records the last 4 communications, but it is rather effective in reducing the negative effects of small ψ values as Fig. 4b illustrates.

We can draw several other conclusions from the results in Fig. 4. First, the tabu list slightly improves even the performance of anti-entropy epidemic dissemination with completely random peer selection ($m = \psi = N$). This is due to the fact that initially views contain only few nodes (to be precise, five, in this case). Without a tabu list, this significantly increases the chance of contacting the same peers in the first few cycles, while the views are still small. Such communications are not effective in advancing dissemination due to the correlated views of the communicating peers. Also note that when there is no limit on message size, the random graph outperforms the ring, especially when the tabu list is applied. This is due to the fact that the number of neighbors of a node in the random graph increases exponentially, so even for a small set of closest nodes, diversity is very high.

Finally, we note that the exponentially increasing neighborhood becomes a disadvantage when ψ is larger, because the view of peers that are further away from the base node in the ranking graph will be more uncorrelated to the view of the original peer. This suggests that for such graphs, peer selection should be aggressive ($\psi = 1$) and should be combined with the use of tabu lists.

5.3. Notes on asymmetric target graphs

The topological role of nodes in asymmetric target graphs is not identical. For example, some nodes can be more central or more connected than others, there can be



(a) Basic T-Man protocol

(b) T-Man with Tabu List

Fig. 4. Time to collect 50% of the neighbors at distance one in the ranking graph. Network size is $N = 2000$. Node views are initialized to contain five random links each. Graph (b) was obtained using a tabu list of size 4.

Author's personal copy

bridge nodes connecting isolated clusters, and so on. While symmetric graphs already exhibit complex behavior, we argue that asymmetric graphs cannot be treated reasonably in a common framework. Each case needs a separate analysis that needs to take into account the particular structure of the graph.

To understand the problem better, consider a ranking method that is independent of the base node. This ranking method will induce a star-like structure since all nodes will be attracted to the very same high ranking nodes. In this case, more and more nodes will contact the nodes that rank high in the (in this case, common) ranking. As a result, convergence speeds up enormously, at the cost of a higher load on the central nodes. The reason is simple: the central nodes can collect the high ranking descriptors faster because they are contacted by many nodes. Due to their central position, they also distribute them very rapidly. One can even exploit this effect. For example, if the goal is to build a super-peer topology, with the high bandwidth nodes in the center, then the central nodes might actually be able to deal with the extra load, thus resulting in an efficient, but still fully self-organizing solution.

This effect can be observed in other interesting topologies as well. For example, rooted regular trees, where the non-leaf nodes have k out-links and one in-link, except the root, that has no in-links. If the ranking graph has such a topology, the resulting target graph will be asymmetric with highly nonuniform average traffic at nodes, as shown in Fig. 5. One reason for this result is that a large proportion of the nodes are leaves. Leaf nodes, having only one neighbor, will have a tendency to talk to nodes that are further up in the hierarchy. This adds extra load on internal nodes and puts them in a more central position.

This in turn has a non-trivial effect on the convergence of the protocol, and allows T-Man to have better performance for trees than for symmetric graphs. Fig. 6 illustrates this effect. In Fig. 6a, we can observe the performance of T-Man for a rooted and balanced binary tree as a ranking graph. We can see that there is a peculiar minimum when message size is unlimited but ψ is small. In this region, the binary tree consistently outperforms the ring, even for a small m.

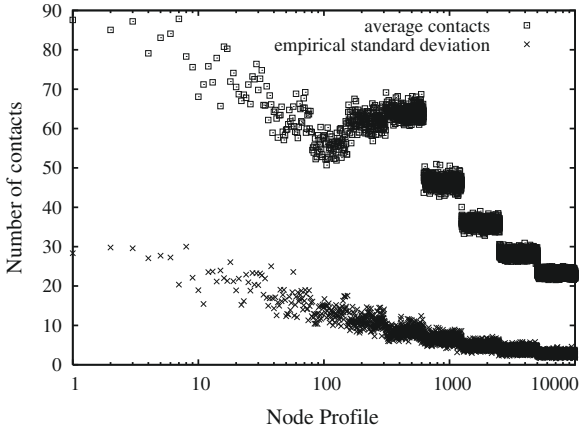


Fig. 5. Number of contacts made by nodes while constructing a binary tree. Statistics are over 30 independent runs. The parameters are $N = 10,000$, $m = 20$, number of cycles is 15, $\psi = 10$ and the tabu list size is 4. In the ranking graph, the root is node 0 and the out-links of node i are $2i + 1$ and $2i + 2$.

This effect is due to the asymmetry of a binary tree. To show this, we ran T-Man with an additional balancing technique, to cancel out the effect of central nodes. In this technique, we limit the number of times any node can communicate (actively or passively) in each cycle to two. In addition, nodes also apply *hunting* [25], that is, when a node contacts a peer, and the peer refuses the connection due to having exceeded its quota, the node immediately contacts another peer until the peer accepts connection, or the node runs out of potential contacts. The results are shown in Fig. 6b. In the region of practical settings of ψ and m, the advantage of the binary tree disappears, while the ring preserves the same performance.

More detailed analysis reveals that in the initial cycles, nodes that are close to the root play a bootstrap function and communicate more than the rest of the nodes. After that, as the overlay network is taking shape, nodes that are further down the hierarchy take over the management of their local region, and so on. This is a rather complex behavior, that is *emergent* (not planned), but nevertheless beneficial. This also suggests that if the target graph is not symmetric, then extra attention is needed when explaining the behavior of T-Man.

5.4. Storage complexity analysis

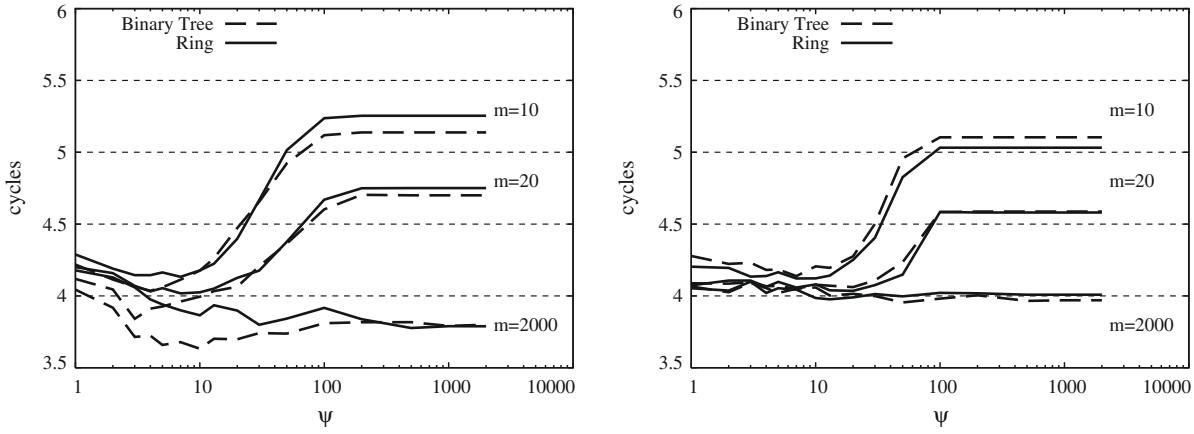
We derive an approximation for the storage space that is needed for maintaining views by the nodes (recall that there is no hard limit enforced by the protocol). This approximation is based on a number of simplifying assumptions that convert the problem into a model of disseminating news items, where only the most interesting news items can spread due to limited message size. Subsequently, we present experimental validation of the approximation using T-Man on different realistic target graphs.

5.4.1. The news spreading model

To derive the approximation, we assume that the ranking method is independent of the base node, that is, all nodes rank a given set of node descriptors the same way. The rational for this assumption is the following. One conclusion of previous sections was that the success of T-Man crucially depends on the fact that whenever a node i selects a peer j using selectpeer, the ranking of the current neighbors of i with node j as a base node is similar to the ranking with node i as a base node, because this way node j can provide relevant node descriptors to node i. Assuming that the ranking does not depend on the base node means that any selected node j is guaranteed to produce an identical ranking to node i, which is the ideal case for T-Man, and this case is approximated well on all graphs where T-Man has good performance.

This assumption, however, introduces a side-effect: it implies that the target graph is a star-like structure, with the m highest ranking nodes forming a clique, and all the other nodes pointing to these m nodes. This level of asymmetry is highly non-typical and therefore is an unrealistic scenario for T-Man. To "fix" this side-effect, we assume that selectpeer returns a random node from the entire network, which makes the role of all nodes identical.

Author's personal copy



(a) T-Man with Tabu List

(b) T-Man with Tabu List and Balancing

Fig. 6. Time to collect 50% of the neighbors at distance one in the ranking graph. The network size is $N = 2000$. Node views are initialized by five random links each. The tabu list size is 4.

In this setting, node descriptors have no relation to actual nodes anymore (that is, the node addresses in the descriptors are never used), so we can think of the model as spreading *news items* that have a natural ranking based on "interestingness".

Let $n(j)$ denote the number of nodes in the network that know about the news item of rank j. The notation $n(j,t)$ allows us to express the time dependence of the same value. We start by showing that $n(j,t) = Nm/j$ if $j > m$ for a large enough t. The main idea is based on the observation that, due to symmetry, $n(j,t)$ grows according to the same curve for all j, but only until the overall number of items in the node's view grows too large and the item with rank j no longer makes it into the exchanged messages (and therefore its replication stops). At that point $n(j,t)$ assumes its final value.

To allow for an approximation of the average storage cost, we model the representation of each news item as a single continuous variable, that is, we assume that all nodes store exactly $0 \leqslant n(j,t)/N \leqslant 1$ instances of the news item of rank j. Under this assumption we can say that the function $n(j,t)$ stops growing when higher ranking items already fill all the available m slots in the messages, since from that point, the news item of rank j will be excluded from all communication:

$$\sum_{k=1}^{j} n(k,t^*) = Nm, \tag{1}$$

where t^* denotes the point in time when this equation holds for the first time. Since $n(j,t)$ never decreases, we have $n(j,t) = n(j,t^*)$ for $t \geqslant t^*$. We know that the functions $n(k,t)$ grow at exactly the same rate for all k, so we can simplify the expressions as $jn(j,t^*) = Nm$, that is,

$$n(j,t) = \frac{Nm}{j}, \quad t \geqslant t^*. \tag{2}$$

This proves the result. Fig. 7 compares the theoretical prediction and the converged distribution obtained experimentally via simulation.

Eq. (2) allows us to approximate the actual storage space that is required for the views of the nodes. We focus only on the items that rank lower than m. The highest ranking m items represent a small constant factor. The sum of all entries with a rank higher than m stored in the system is

$$\sum_{j=m}^{N} \frac{Nm}{j} \approx \int_m^N \frac{Nm}{j}\,dj = Nm(\ln N - \ln m) = Nm \ln \frac{N}{m} = O(N \log N). \tag{3}$$

Therefore each view stores $O(\log N)$ entries on the average. Note that this result is independent of the number of iterations executed, and it is also independent of the actual form of the functions $n(j,t)$; recall that the only assumption we made was that these functions are monotonically increasing.

Finally, we note that $Nm/j = Nmj^{-1}$ is technically a power law distribution, as it follows the form $j^{-\gamma}$. Power laws are very frequently observed in complex evolving networks [26]. The phenomenon is often due to some form of "the rich get richer" effect. One can link our results to the study of other complex networks, for example, social networks. All nodes start with a random constant-size set

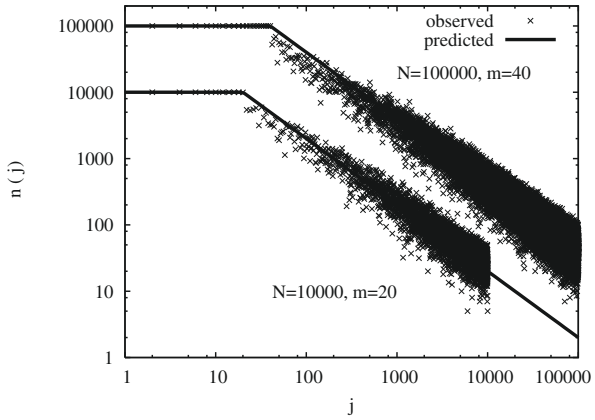


Fig. 7. Experimental results and values predicted by Eq. (2) for $n(j)$ with two sets of parameters $N = 10,000$, $m = 20$ and $N = 100,000$, $m = 40$. For each j, the converged value of $n(j)$ is indicated as a separate point. The observed values correspond exactly to the predicted one for the initial constant section, and are covered by the line segment on the graph.

Author's personal copy

of news items, and they gossip always only the m most interesting ones that they currently know. This dynamics results in a power law distribution of news items, with the most interesting news being known to everyone. Furthermore, each participant learns only about $O(\log N)$ news items from the overall $O(N)$ news items available.

5.4.2. Empirical validation

We verify experimentally that the prediction in (2) holds for T-Man when different ranking methods are employed. This would support as a consequence the claim that Eq. (3) characterizes the storage complexity of the protocol.

We need to generalize $n(j)$ since ranking can now depend on the base node. Let $n(j)$ be the number of nodes that know about the node with rank j according to their *own* ranking of the entire network. Fig. 8 shows the values of $n(j)$ for three ranking graphs at three different times. Although the experiments reported in Fig. 8 were performed without a tabu list, further experiments (not shown) show that tabu lists have no observable effect on the distribution of ranks in the views. They only speed up convergence of the protocol as discussed earlier.

In Fig. 8 we can observe that the ring fulfills the assumptions of Section 5.4.1 best: the $n(j)$ values that have not stopped growing have the same value at each time point, which means they indeed grow at the same rate. The largest deviation can be observed in the case of the random graph. There, the growth of the $n(j)$ values slows down smoothly which implies that the assumption they grow at the same rate does not hold. This results in a slight "overshoot" where the observed values are slightly higher than those predicted.

Note that in the case of the binary tree, the predicted values match closely the observed ones even though the topology is not symmetric. This further underlines the robustness of the prediction. In other words, the seemingly strong assumptions of the theory in fact leave the essential dynamics almost unchanged, which indicates that we could understand important features of the protocol. Of course, the more central nodes need more storage capacity, the prediction holds only on average. However, in our preliminary experiments (not shown), we have seen that setting a reasonable hard limit on the view size that is significantly larger than m (for example, 1000 items) does not result in any significant difference in performance. For this reason we opted for the simplified discussion and we omit hard limits on the view size in this paper.

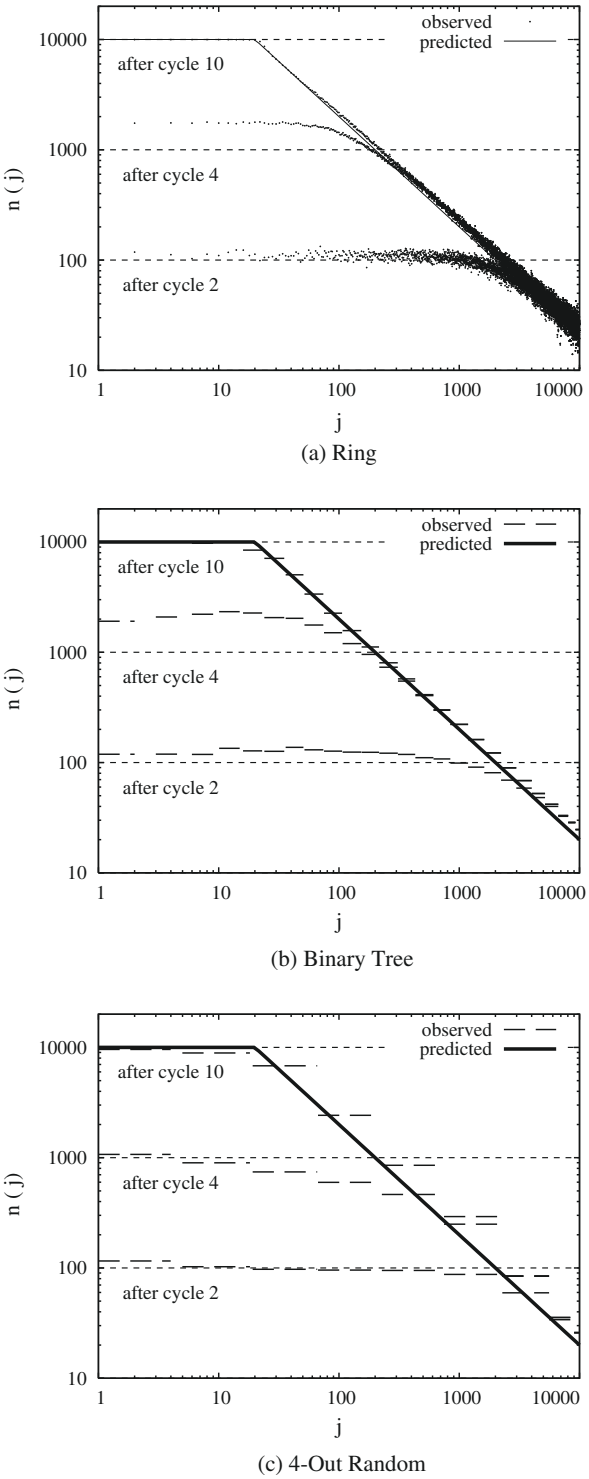


Fig. 8. Experimental and predicted values of $n(j)$ for three different ranking graphs. Experiments were run with $N = 10,000$, $m = 20$ and $\psi = 10$, without a tabu list. Note that the plots contain three snapshots of the simulation for cycles 2, 4 and 10. In figure (a), the dots representing the situation after 10 cycles for values $j \leqslant 100$ are covered by the predicted line.

6. Experimental results

In the previous section we considered the most basic version of the protocol to shed light on its convergence properties and storage complexity. This section is concerned with developing additional techniques that allow for the practical application of the protocol; in particular, we address two important problems: how to start and how to stop the protocol. We also present an extensive empirical analysis under different parameter settings and different failure scenarios, introduced by a brief discussion of the simulation environment and the figures of merit analyzed in this paper.

6.1. A practical implementation

So far we assumed that the protocol is started at all nodes at once, in a synchronous fashion, and we were not dealing with termination at all. We also assumed that

Author's personal copy

at all nodes the initial set of known peers is a random sample from the network. In this section, we replace these unrealistic assumptions with practically feasible solutions.

6.1.1. Peer sampling service

The peer sampling service provides each node with continuously up-to-date random samples of the entire population of nodes. Such samples fulfill two purposes: they enable the random initialization of the T-Man view, as discussed in Section 4, and make it possible to implement a starting service as well, allowing for the deployment of various gossip-based broadcast and multicast protocols.

In this paper we consider an instantiation of the peer sampling service based on the Newscast protocol [11], chosen for its low cost, extreme robustness and minimal assumptions. The basic idea of Newscast is that each node maintains a local set of random node addresses: the (partial) *view*. Periodically, each node sends its view to a random member of the view itself. When receiving such a message, a node keeps a fixed number of freshest addresses (based on timestamps), selected from those locally available in the view and those contained in the message.

Each node sends one message to one other node during a fixed time interval. Implementations exist in which these messages are small UDP messages containing approximately 20–30 IP addresses, along with the ports, timestamps, and descriptors such as node IDs. The time interval is typically long, in the range of 10 s. The cost is therefore small, similar to that of heartbeat messages in many distributed architectures. The protocol provides high quality (i.e., sufficiently random) samples not only during normal operation (with relatively low churn), but also during massive churn and even after catastrophic failures (up to 70% nodes may fail), quickly removing failed nodes from the local views of correct nodes.

6.1.2. Starting and terminating the protocol

We implemented a simple starting mechanism based on well-known broadcast protocols. The content of the broadcast message may be a simple "wake up" specifying *when* to build a predefined network, or it may include additional information specifying *what* network to build (e.g., by providing the implementation of a specific ranking function). To simplify our simulation environment, we adopt the first approach; technical issues related to the second one may be easily solved in a real implementation.

The following terminology is used when discussing the starting mechanism. We say that a node is *active* if it is aware of and explicitly participating in a specific instance of T-Man; if the node is not aware that a protocol is being executed, it is called *inactive*.

Initially, there is only one active node, the *initiator*, activated by an external event (e.g., a user's request). An inactive node may become active by exchanging information with nodes that are already active. When a node becomes active, it immediately starts executing the T-Man protocol. The final goal is to activate all nodes in the system, i.e., to start the protocol at all nodes.

The actual implementation of the broadcast can take many forms that differ mainly in communication overhead and speed.

Flooding. As soon as a node becomes active for the first time, it sends a "wake up" message to a small set of random nodes, obtained from the peer sampling service. Subsequently, it remains silent.

Anti-Entropy, Push-only. Periodically, each active node selects a random peer and sends a "wake up" message [25].

Anti-Entropy, Push–Pull. Periodically, each node (active or not) exchanges its activation state with a random peer. If either of them was active, they both become active [25].

As described above, a node becomes active as soon as it receives a message from another active node. Note, however, that messages belonging to the starting protocol are not the only source of activation; a node may also receive a T-Man message, from a node that has already started to execute the protocol. This message also activates the recipient node.

As is well known, flooding is fast and effective but very expensive due to message duplications. In comparison, the most important advantage of the other two approaches is the dramatically lower communication overhead per unit time. The overhead can further be reduced to almost zero, due to the fact that the starting service messages can be piggybacked, for example, on Newscast messages that implement the peer sampling service.

After the target graph has been built, the protocol does not need to run anymore and therefore must be terminated. Clearly, detecting global convergence is difficult and expensive: what we need is a simple local mechanism that can terminate the protocol at all nodes independently.

We propose the following mechanism. Each node monitors its own local view. If no changes (i.e., node additions) are observed for a specified period of time (δ_{idle}), it suspends its active thread. We call this state *suspended*. If a view change occurs when a node is suspended (due to an incoming message initiated by another node that is still active), the node switches again to the active state, and resets its timer that measures idle time.

6.2. Simulation environment

All the experiments are event-based simulations, performed using PeerSim, an open-source simulator designed for large-scale P2P systems and publicly available at SourceForge [27]. The applied transport layer emulates end-to-end delays between pairs of nodes based on the traces of the King data set [28]. Delays reported in these traces range from 1 ms to 400 ms, and the probability distribution is as shown in Fig. 9.

The following parameters are fixed in the experiments: the size of the tabu list is 4, and the peer selection parameter (ψ) is 1. If different values are not explicitly mentioned, the message size (m) is 20, the cycle length (Δ) is 1 s, and the value of δ_{idle} is set to 4 s. Each experiment is repeated 50 times with different random seeds. Plots show the average of the observed measures, along with error

Author's personal copy

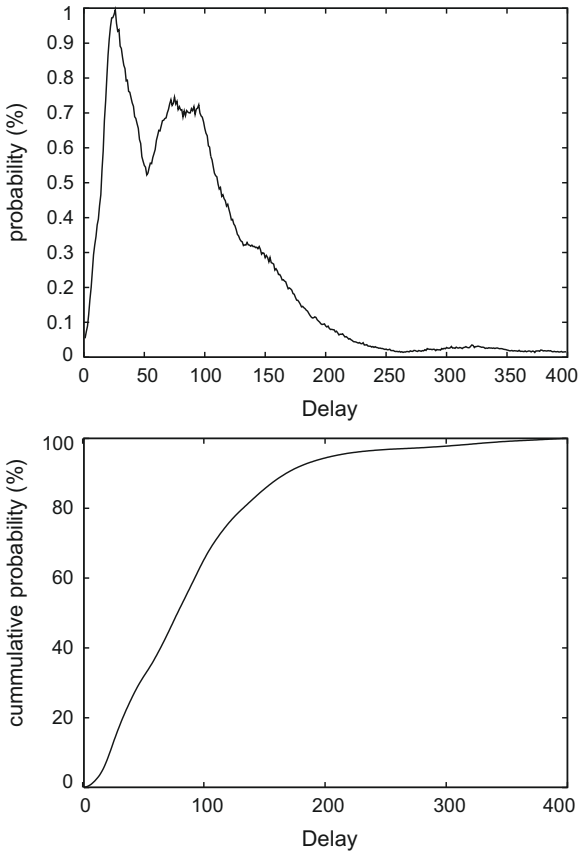


Fig. 9. Probability distribution of end-to-end delays as reported in the King data set [28].

bars; when graphically feasible, individual experiments are displayed as separate dots with a small random translation.

6.3. Ranking methods

To emphasize the robustness of T-Man to the actual target graph being built, we performed all experiments on two different tasks: building a sorted ring, and building a binary tree. These two graphs have very different topologies: the ring has a large (linear) diameter while the tree has a small (logarithmic) one. Besides, as pointed out in Section 5.3, in the tree some nodes are more central than others, while in the ring all nodes are equal from this point of view.

In the previous sections, we applied the concept of a ranking graph to (implicitly) define the ranking method. This approach is not practical, so we need to define explicit and locally computable ranking methods.

6.3.1. Sorted ring

Creating a sorted ring is very useful, for example, for the decentralized computation of the ranking of nodes [29] or jump-starting distributed hash tables, such as Chord [8]. The latter application is further discussed in Section 7.

We assume that the node profile is an element of a collection, over which a total ordering relation is defined. In particular, we work with 60-bit integers as node profiles that are initialized at random for each node. We want the target graph to be a ring, in which the node profiles are ordered (except one pair where the largest and smallest values meet) to close the ring.

To achieve this target graph, the output of the ranking method $\text{RANK}(x, y_1, \ldots, y_k)$ is defined as follows. First we construct a sorted ring (as defined above) out of the set of input profiles $y_1, \ldots, y_k$ and the base node x, and assign a rank value to all nodes: the minimal hop count from x in this ring. The output of the ranking method is an ordered list of the input profiles according to these assigned rank values. Note that this is a *direction-dependent* ranking method, that cannot be induced by a distance metric over the node profiles. For simplicity, we will call T-Man with this ranking method Sorted ring.

6.3.2. Binary tree

The second topology we consider is an undirected rooted binary tree. To achieve a well controlled target graph for the sake of experimental comparison, the node profiles are defined as follows. If there are N nodes, then we assign the integers $1, \ldots, N$ to the nodes in some arbitrary order. The node with value 1 is the root. Using the binary representation of these integers, the node $0a_2 \cdots a_m$ has two children: $a_2 \cdots a_m 0$ and $a_2 \cdots a_m 1$. Numbers starting with 1 belong to leafs.

It is easy to calculate the shortest path length in this tree between two arbitrary nodes, based on the two node profiles. This notion of distance is used to define the ranking function required by T-Man to build the tree: $\text{RANK}(x, y_1, \ldots, y_k)$ sorts the input profiles $y_1, \ldots, y_k$ according to distance from the base node x. For simplicity, we will call T-Man with this ranking method Tree.

6.4. Performance measures

We are interested both in the effectiveness (speed and quality) and efficiency (cost) of the protocol. We evaluate our protocols using the following performance measures: *convergence time*, *target links found*, *termination time* and *communication costs*.

Convergence time. The time needed to obtain the *perfect* target graph. In the case of Sorted ring, each node must know at least its first successor and predecessor in the sorted ring. For Tree, each node different from the root must know its parent, and non-leaf nodes must know their children.

Target links found. The number of links in the target graph that are actually found by T-Man at a certain time, typically at termination time. This allows for a more fine-grained assessment of performance than convergence time.

Termination time. The total time needed to complete (start, execute and stop) the protocol at *all* nodes. This may be considerably longer than convergence time, although, as we will see, typically only few nodes are still active after reaching convergence.

Author's personal copy

Communication cost. The number of messages exchanged. Note that all messages ever exchanged are of the same size.

The unit of time will be cycles or seconds, depending on which is more convenient (note that cycle length defaults to 1 s). We also note that convergence time is not defined if the protocol terminates before converging. In this case, we use the number of identified target links as a measure.

6.5. Evaluating the starting mechanism

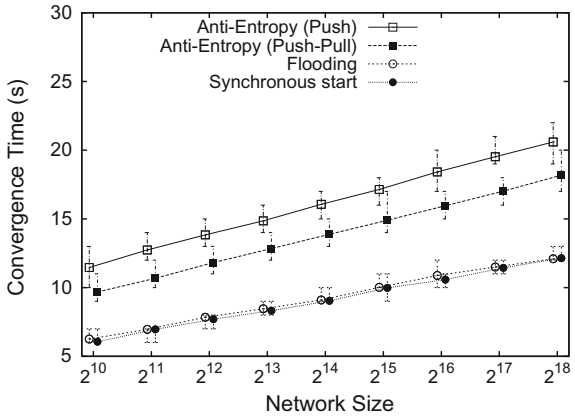
Fig. 10 shows the convergence time for SORTED RING and TREE, using the starting protocols described in Section 6.1.2. The cycle length of the anti-entropy versions was the same as that of T-MAN, and the flooding protocol used 20 random neighbors at all nodes. The case of synchronous start is also shown for comparison. Note that these figures do not represent a direct measure of the performance of well-known starting protocols; rather, convergence time plotted here represents the overall time needed to both start the protocol and reach convergence, with T-MAN and the broadcast protocol running concurrently.

In the case of flooding, "wake up" messages quickly reach all nodes and activate the protocol; almost no delay is observed compared to the synchronous case. Anti-entropy mechanisms result in a few seconds of delay. In the experiments that follow, we adopt the anti-entropy, push–pull approach, as it represents a good tradeoff between communication costs and delay. Note however that (unlike the push approach) the push–pull approach assumes that at least the starting service was started at all nodes already.

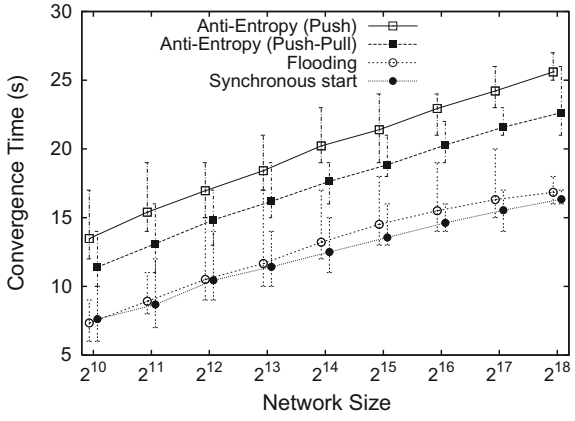
6.6. Evaluating the termination mechanism

We experimented with various settings for δ_{idle} ranging from 2 s to 12 s. Fig. 11 shows both convergence time (bottom three curves) and termination time (top three curves) for different values of δ_{idle}, for SORTED RING and TREE, respectively. In both cases, termination time increases linearly with δ_{idle}. This is because, assuming the protocol has converged, each additional cycle to wait simply adds to the termination time.

For small values convergence was not always reached, especially for TREE. For SORTED RING, all runs converged except the case when $\delta_{idle} = 2$ and $N = 2^{16}$, when 76% of the runs converged. For TREE, all runs converged with $\delta_{idle} > 5$ and no runs converged for $(\delta_{idle} = 2, N = 2^{13}), (\delta_{idle} = 2, N = 2^{16})$, and $(\delta_{idle} = 3, N = 2^{16})$. Even in these cases, the quality of the target graph at termination time was almost perfect,

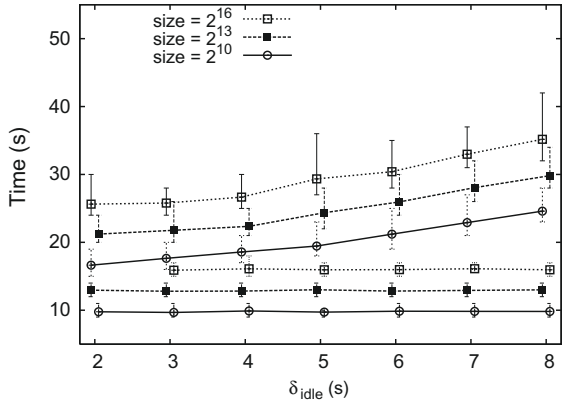


(a) SORTED RING

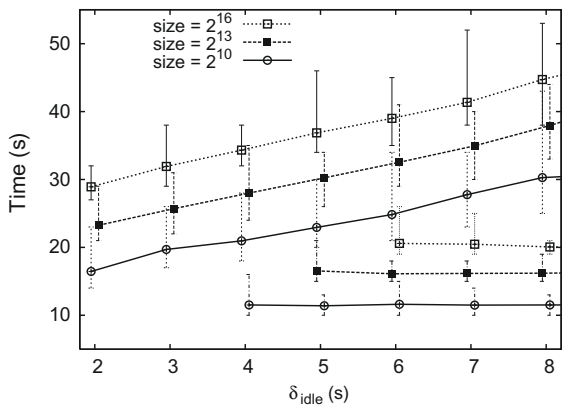


(b) TREE

Fig. 10. Convergence time as a function of size, using different starting protocols.



(a) SORTED RING



(b) TREE

Fig. 11. Convergence time (bottom curves) and termination time (top curves) as a function of δ_{idle}.

Author's personal copy

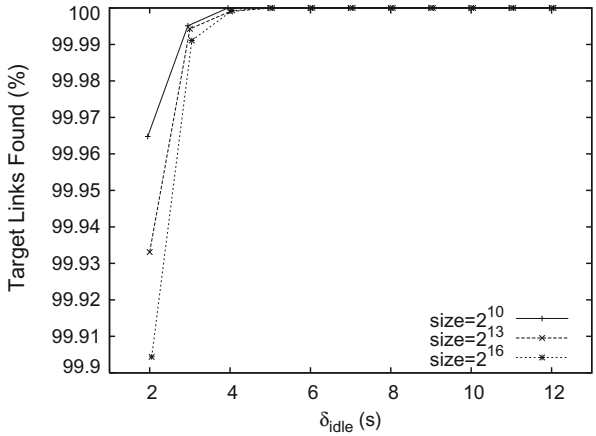
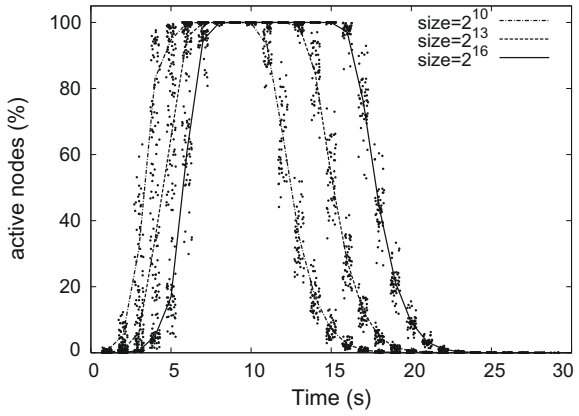
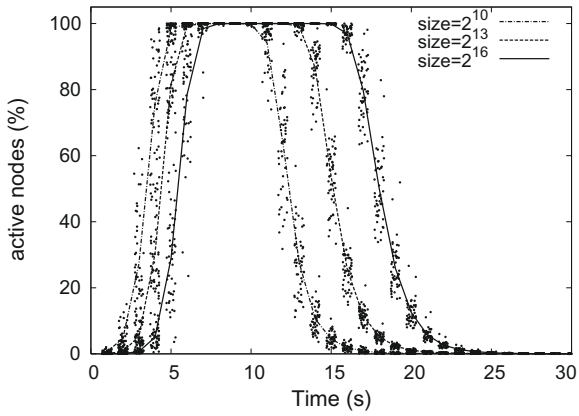


Fig. 12. Quality of the target TREE graph at termination time as a function of δ_{idle}.



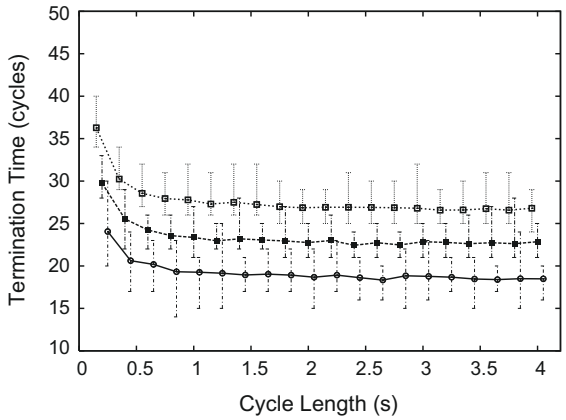
(a) SORTED RING



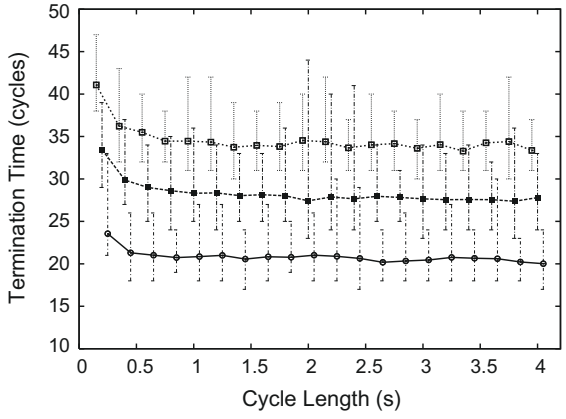
(b) TREE

Fig. 13. Proportion of active nodes during execution.

as shown in Fig. 12. In the worst of our experiments, we observed that no more than 0.1% of the target links were missing at termination. This may be sufficient for most applications, especially considering that the target graphs will never be constructed perfectly in a dynamic scenario, where nodes are added and removed continously. Nevertheless, from now on, we discard the parameter combinations that do not always converge.



(a) SORTED RING



(b) TREE

Fig. 14. Termination time as a function of cycle length.

Apart from longer executions, an additional consequence of choosing large values of δ_{idle} is a higher communication cost. However, since not all nodes are active during the execution, the overall number of messages sent per node on average is less than one quarter of the number of cycles until global termination. To understand this better, Fig. 13 shows how many nodes are active during the construction of SORTED RING and TREE, respectively. The curves show both an exponential increase in the number of active nodes when starting, and an exponential decrease when stopping. The period of time in which all nodes are active is relatively short.

These considerations suggest the use of higher values for δ_{idle}, at the cost of a larger termination time and a larger number of exchanged messages. The chosen value of δ_{idle} (4 s) represents a good tradeoff between the desire of obtaining a perfect target graph and the consequently larger cost in time and communication.

6.7. Parameter tuning

6.7.1. Cycle length

If a faster execution is desired, one can always decrease the cycle length. However, after some point, decreasing cycle length does not pay off because message delay becomes longer than the cycle length and eventually the network will be congested by T-MAN messages. Fig. 14 shows the behavior

Author's personal copy

of T-Man with a cycle length varying between 0.2 s and 4 s. The figure shows the number of cycles required to terminate the protocol. Small cycle lengths require a larger number of cycles, while after a given threshold (around 1 s), the number of cycles required to complete a protocol is almost constant. The reason for this behavior is that with short cycles, multiple cycles may be executed before a message exchange is concluded, thus wasting bandwidth in sending and receiving old information multiple times.

6.7.2. Message size

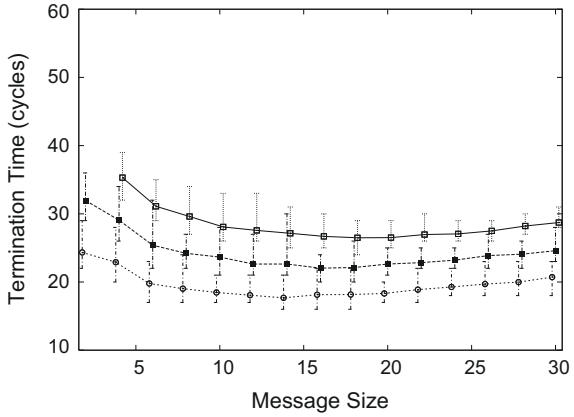
In Section 5, we have examined the effect of the message size parameter (m) in detail. Here we are interested in the effect of message size on termination time. Fig. 15 shows that by increasing the size of messages exchanged by Sorted ring termination time slightly increases after around $m = 20$. The reason is that a node becomes suspended only after the local view remains unchanged for a fixed number of cycles, but increasing the message size has the effect of increasing the number of cycles in which view changes might occur, thus delaying termination. The results for Tree have more variance, which might have to do with the unbalanced nature of the topology, as discussed in Section 5.3.

6.8. Failures

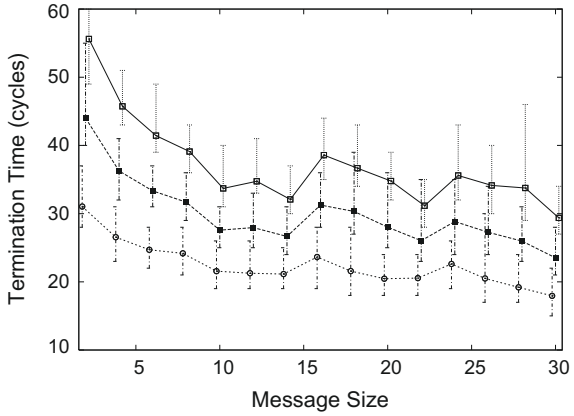
The results discussed so far were obtained in static networks, without considering any form of failure. Here, we consider two sources of failure: message losses and node crashes. Since in this paper we consider only the overlay *construction* problem, and not *maintenance*, we do not explicitly consider scenarios involving node churn. Instead, we model churn through nodes leaving, and do not allowing joining nodes to participate in an ongoing construction. Furthermore, since we do not have a leave protocol, leaving nodes are identical to crashing nodes from our point of view.

6.8.1. Message loss

While a simple solution could be to adopt a reliable, connection-oriented transport protocol like TCP, it is more attractive to rely on a lightweight but perhaps unreliable transport. In this case, we need to demonstrate that T-Man can cope well with message loss. Fig. 16 shows that T-Man is highly resilient to message loss and so a datagram-oriented protocol like UDP is a perfectly suitable choice, as message losses only slow down the protocol slightly. Many message exchanges are either never started or never completed, thus requiring more cycles to terminate the protocol execution. The *quality* does not suffer much either. In both Sorted ring and Tree, around 1% of the target links may be missing, as shown by Fig. 17. Note that the mean message loss ratio for geographic networks like the Internet is around 2% [30], an order of magnitude smaller than the maximum message loss ratio tested in our experiments.

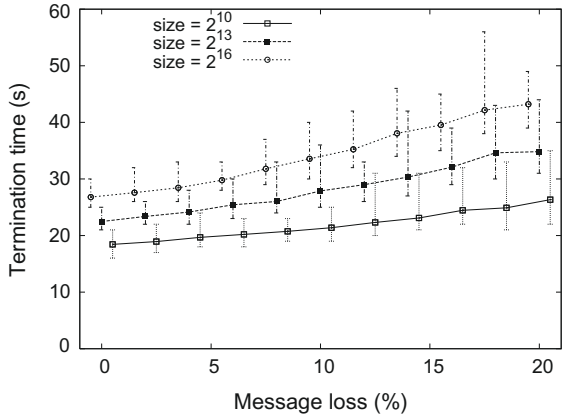


(a) Sorted ring

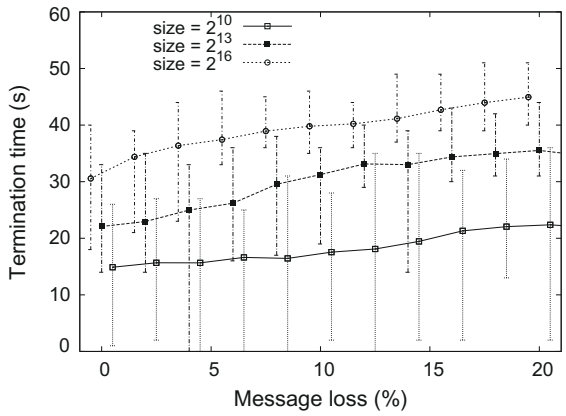


(b) Tree

Fig. 15. Termination time as a function of message size.



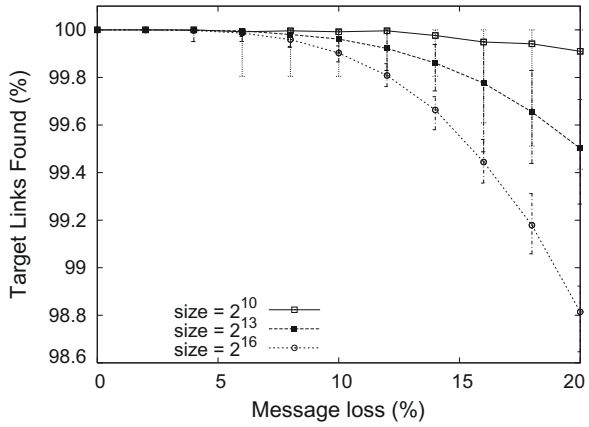
(a) Sorted ring



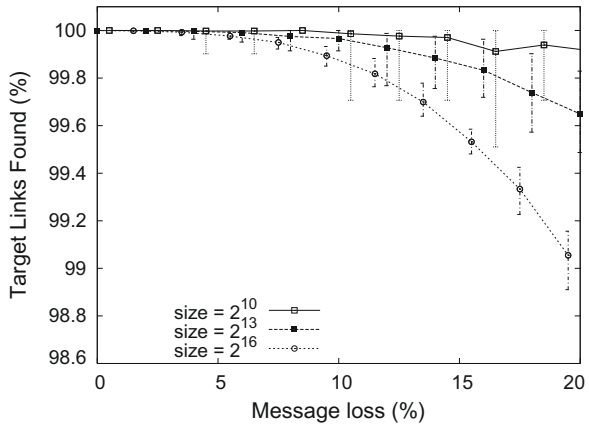
(b) Tree

Fig. 16. Termination time as a function of message loss rate.

Author's personal copy

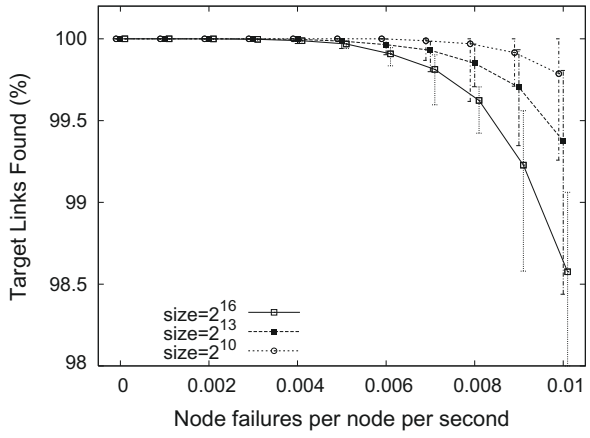


(a) SORTED RING

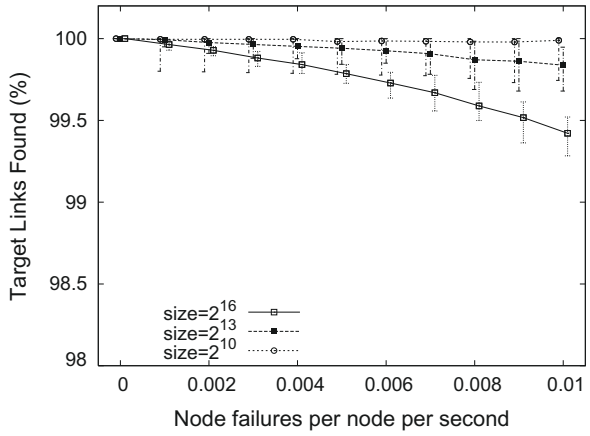


(b) TREE

Fig. 17. Target links found by the termination time as a function of message loss rate.



(a) SORTED RING



(b) TREE

Fig. 18. Target links found by the termination time as a function of failure rate.

6.8.2. Node crashes

Fig. 18 shows the behavior of T-MAN with a variable failure rate, measured as the total number of nodes leaving the network per second per node. We experimented with values ranging from 0 to 10^{-2}, which is two orders of magnitude larger than the value of 10^{-4} suggested as the typical behavior of some P2P networks [31]. The results show that both SORTED RING and TREE are robust in normal scenarios, with TREE being considerably more reliable in the range of extreme failure rates. This is due to the unbalanced nature of the topology as discussed in Section 5.3.

7. Bootstrapping CHORD

After analyzing the behavior of T-MAN on relatively basic examples, in this section we present a more complex application: rapidly bootstrapping CHORD-like networks [8]. We call this protocol T-CHORD.

7.1. A brief introduction to CHORD

CHORD is an example of a key-based overlay routing protocol. In such protocols, subsets of the key space are assigned to nodes, and each node has a routing table that it uses to route messages addressed by a specific key towards the node that is responsible for that key. These routing protocols are used as a component in the implementation of the *distributed hash table* abstraction, where (key, object) pairs are stored over a decentralized collection of nodes and retrieved through the routing protocol.

We provide a simplified description of CHORD, necessary to understand T-CHORD. Nodes are assigned random t-bit IDs; keys are taken from the same space. The ID length t must be large enough to make the probability of two nodes or two keys having the same ID negligible. Nodes are ordered in an sorted ring as described in Section 6.3.1. The way this ring is constructed naturally inspires a *follows* relation over the entire ID (and key) space: we say that a follows b if $(a-b+2^t)|2^t < 2^{t-1}$; otherwise, a *precedes* b. We also define a notion of distance, again, inspired by the sorted ring, as follows: $d(a,b) = \min(|a-b|, 2^t - |a-b|)$. The *successor* of an arbitrary number i (that is, not necessarily existing node ID) is the node with the smallest ID that follows i, as defined above. We denote the successor of i by $succ_1(i)$. The concepts of predecessor, jth successor, and jth predecessor are defined similarly. Key k is under the responsibility of node $succ_1(k)$.

Each node maintains a routing table that has two parts: *leaves* and *fingers*. Leaves define an r-regular lattice, where

Author's personal copy

each node n is connected to its r nearest successors $succ_1(n) \dots succ_r(n)$. Fingers are long range links: for each node n, its j^{th} finger is defined as $succ_1(n+2^j)$, with $j \in [0, t-1]$. Routing in CHORD works by forwarding messages following the successor direction: when receiving a message targeted at key k, a node n forwards it to its leaf or finger that precedes (or is equal to) and is closest to $succ_1(k)$, the intended recipient of the message.

Due to the fingers, the number of nodes that need to be traversed to reach a destination node is $O(\log N)$ (with high probability), where N is the size of the network [8]. Leaves, on the other hand, are used to improve the probability of delivering a message in case of failures, and to avoid that the ring can be broken into disjoint partitions.

7.2. The T-CHORD protocol

In the context of CHORD, our overlay construction problem translates to initializing the routing tables of all nodes simultaneously from scratch. The existing join protocol is not designed to handle the massive concurrency involved in a jump-starting process, when all the nodes are trying to join at the same time [8]. On the other hand, naive approaches where nodes are forced to join the overlay in some specified order results in at least linear time needed to construct the network (not to mention the serious problem of synchronizing the operations).

For constructing the leaf set and the fingers simultaneously, we apply T-MAN with an appropriate ranking

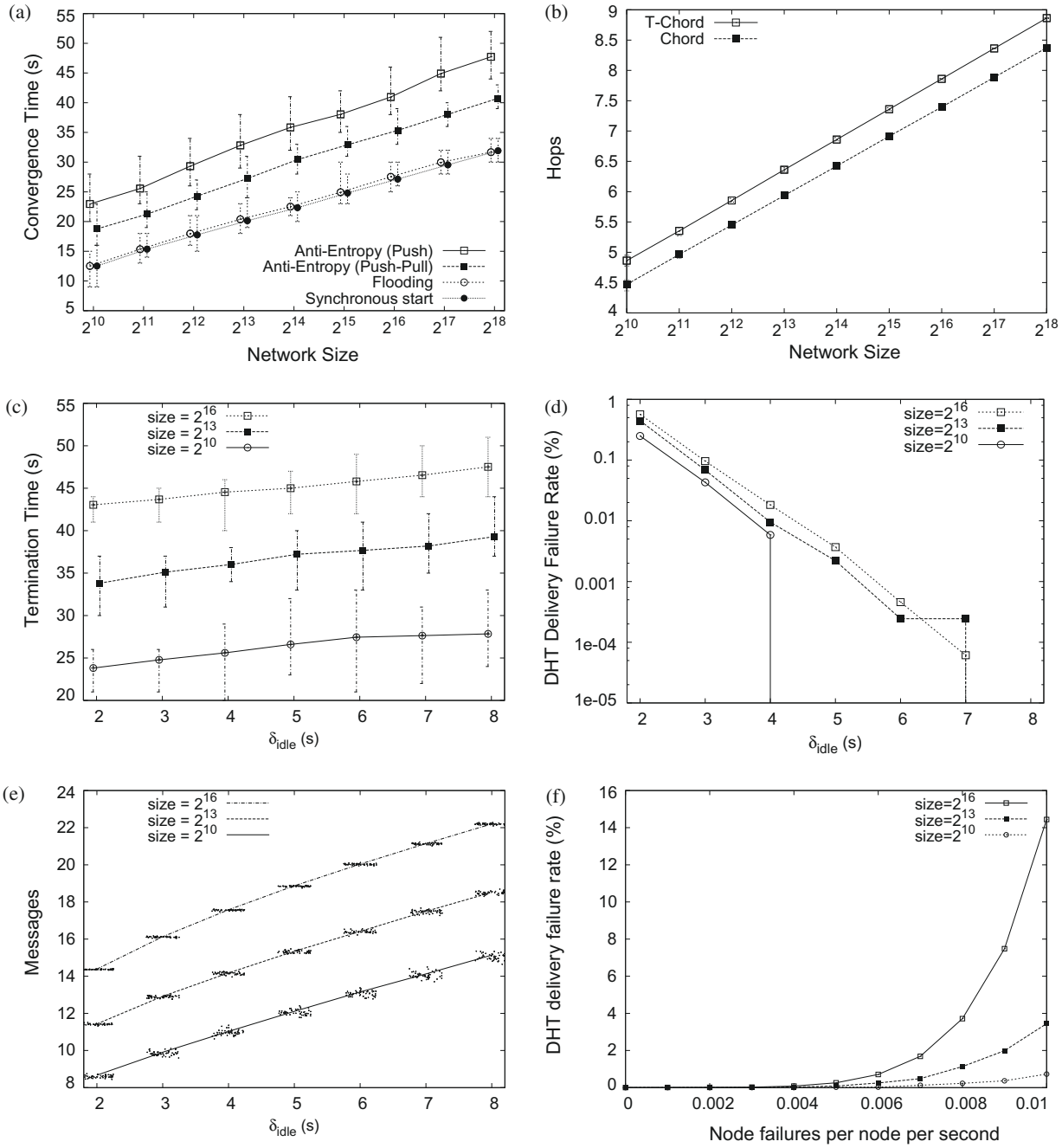


Fig. 19. Experimental results with T-CHORD.

Author's personal copy

method. As usual, we use node IDs as node profiles. The ranking method first produces a ranking using the ranking method of SORTED RING as seen in the previous sections. Let this ranking be $x_1, x_2, \ldots$. Subsequently, the ranking method finds a set of descriptors that represent the best approximation of the ideal finger set of the base node, and orders this set according to increasing distance from the base node in the ID space. Let this ordered list be $y_1, y_2, \ldots$. The final ranking is calculated from the list $x_1, y_1, x_2, y_2, \ldots$. In this list some descriptors appear twice. To get the final ranking we remove the *second* occurrence of each descriptor from this list.

When receiving a message, as before, the node simply merges it with its current view. At any time, the actual finger set is then constructed by each node locally from nodes in its current view. We also note that we obtained very good preliminary results simply relying on the local views that result from pure ring construction. However, if finger candidates are also included in the messages, we achieve a noticeable improvement. This approach can be generalized to other target graphs as well, such as PASTRY [9], as shown in [10]. All the other components, that is, the peer sampling service, the starting service and the termination mechanism are applied as before.

7.3. Experimental results

First, we modify slightly the definition of some performance measures for this application. While the concept of termination time remains unchanged, convergence is defined as the time needed to obtain routing tables, that are sufficiently correct to route messages without errors (i.e., without messages failing to reach their destination). In other words, in a converged network we now allow for the possibility of having sub-optimal routing tables (with missing leaves or fingers) provided routing works without error. Note that if the ring is complete, then all messages are guaranteed to be delivered, so according to this new measure, convergence occurs no later than the convergence of the ring.

The default parameters are the same as for SORTED RING and TREE. Note however that the applied message size of 20 now means that 10 entries are reserved for the leaf set construction and 10 for the fingers. For this reason, convergence is expected to take longer than with 20 entries assigned to a single target graph.

Fig. 19a shows the convergence time for different starting protocols and for a variable network size. Convergence now takes longer than the values shown in Fig. 10 due to the decreased message size (10 instead of 20) relative to each part of the routing table. Fig. 19b compares the quality of the converged routing tables obtained by T-CHORD with the ideal CHORD overlay described in [8], measured as the average number of hops needed to deliver messages. The performance of T-CHORD is slightly worse due to potentially missing routing table entries, but approximates that of the perfect CHORD network closely.

Fig. 19c presents termination times for different values of parameter δ_{idle}. For small values of δ_{idle} and for large networks, we found that the protocol never reaches convergence. Nevertheless, Fig. 19d shows that even for small values of δ_{idle}, the number of messages never delivered to the correct destination is smaller than 1%, which means that the obtained overlay is a good approximation of CHORD. However, for $\delta_{idle} = 8$, all our test runs resulted in 100% successful message delivery, so we adopt this value for the protocol. The slight disadvantage is a larger number of messages exchanged and a slower termination time.

Fig. 19e shows the average number of messages sent by a node in the network until termination. This is significantly lower than the termination time, which could be expected based on our findings discussed before (see Fig. 13).

Finally, Fig. 19f shows the behavior of T-CHORD in a faulty environment. Similar to SORTED RING and TREE, the variable failure rate, measured as the total number of nodes leaving the network per second per node, ranges from 0 to 10^{-2}. An increasing number of routed messages get lost, although, as mentioned previously, the upper end of the failure rate range can be considered extreme. For normal failure rates, the performance is only slightly degraded.

8. Conclusions

In this paper we have presented T-MAN, a lightweight gossip-based protocol for constructing various overlay networks. The target network is given by the ranking method, which is a parameter of the protocol. T-MAN is robust to the target network: it exhibits good performance that is mostly invariant over a wide range of target networks such as rings and trees. The protocol is simple and robust to failure scenarios which makes it attractive for practical applications.

In closing, we note that T-MAN has been successfully applied for constructing the PASTRY overlay network [10]. We do not discuss this particular application here due to space limitations. In this paper we have chosen to focus on overlay construction as opposed to overlay maintenance, which we have explored elsewhere [12]. Our overlay maintenance techniques involve limited local view sizes and periodic removal of old entries from the view. In addition, random samples from the network are constantly injected into the local view.

The most important future development involves characterizing the performance of the protocol theoretically, based on the target network. In this paper we have presented numerous observations derived mostly from heuristic and empirical considerations that outline possible directions for such a theoretical framework.

Acknowledgements

Partial support for this work was provided by the Future and Emerging Technologies unit of the European Commission through Projects BISON (IST-2001-38923) and DELIS (IST-2002-001907). M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

Author's personal copy

References

[1] R. van Renesse, K.P. Birman, W. Vogels, Astrolabe: a robust and scalable technology for distributed system monitoring, management, and data mining, ACM Transactions on Computer Systems 21 (2) (2003) 164–206.

[2] P.T. Eugster, R. Guerraoui, S.B. Handurukande, A.-M. Kermarrec, P. Kouznetsov, Lightweight probabilistic broadcast, ACM Transactions on Computer Systems 21 (4) (2003) 341–374.

[3] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proceedings of the 16th ACM International Conference on Supercomputing (ICS'02), 2002.

[4] L.A. Adamic, R.M. Lukose, A.R. Puniyani, B.A. Huberman, Search in power-law networks, Physical Review E 64 (2001) 046135.

[5] A. Montresor, A robust protocol for building superpeer overlay topologies, in: Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing (P2P'04), IEEE Computer Society, Zurich, Switzerland, 2004, pp. 202–209.

[6] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making gnutella-like p2p systems scalable, in: Proceedings of ACM SIGCOMM 2003, 2003, pp. 407–418.

[7] S. Voulgaris, A.-M. Kermarrec, L. Massoulié, M. van Steen, Exploiting semantic proximity in peer-to-peer content searching, in: Proceedings of 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004), 2004, pp. 238–243.

[8] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. 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), ACM, ACM Press, San Diego, CA, 2001, pp. 149–160.

[9] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, in: R. Guerraoui (Ed.), Middleware 2001, Lecture Notes in Computer Science, vol. 2218, Springer-Verlag, 2001, pp. 329–350.

[10] M. Jelasity, A. Montresor, O. Babaoglu, The bootstrapping service, in: Proceedings of the 26th International Conference on Distributed Computing Systems Workshops (ICDCS WORKSHOPS), IEEE Computer Society, Lisboa, Portugal, 2006, International Workshop on Dynamic Distributed Systems (IWDDS).

[11] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, M. van Steen, Gossip-based peer sampling, ACM Transactions on Computer Systems 25 (3) (2007) 8.

[12] M. Jelasity, O. Babaoglu, T-Man: gossip-based overlay topology management, in: S.A. Brueckner, G. Di Marzo Serugendo, D. Hales, F. Zambonelli (Eds.), Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers, Lecture Notes in Computer Science, vol. 3910, Springer-Verlag, 2006, pp. 1–15.

[13] S. Voulgaris, M. van Steen, An epidemic protocol for managing routing tables in very large peer-to-peer networks, in: Proceedings of the 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, (DSOM 2003), Lecture Notes in Computer Science, vol. 2867, Springer, 2003.

[14] K. Aberer, A. Datta, M. Hauswirth, R. Schmidt, Indexing data-oriented overlay networks, in: Proceedings of 31st International Conference on Very Large Databases (VLDB), ACM, Trondheim, Norway, 2005.

[15] A. Shaker, D.S. Reeves, Self-stabilizing structured ring topology p2p systems, in: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005), IEEE Computer Society, Konstanz, Germany, 2005, pp. 39–46.

[16] D. Angluin, J. Aspnes, J. Chen, Y. Wu, Y. Yin, Fast construction of overlay networks, in: Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2005, pp. 145–154.

[17] S. Voulgaris, M. van Steen, Epidemic-style management of semantic overlays for content-based searching, in: J.C. Cunha, P.D. Medeiros (Eds.), Proceedings of Euro-Par, Lecture Notes in Computer Science, vol. 3648, Springer, 2005, pp. 1143–1152.

[18] L. Massoulié, A.-M. Kermarrec, A.J. Ganesh, Network awareness and failure resilience in self-organising overlays networks, in: Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS 2003), Florence, Italy, 2003, pp. 47–55.

[19] F. Bonnet, A.-M. Kermarrec, M. Raynal, Small-world networks: from theoretical bounds to practical systems, Principles of Distributed Systems, vol. 4878, Springer, 2007, pp. 372–385.

[20] J.A. Patel, I. Gupta, N. Contractor, JetStream: achieving predictable gossip dissemination by leveraging social network principles, in: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications (NCA 2006), Cambridge, MA, USA, 2006, pp. 32–39.

[21] B.Y. Zhao, L. Huang, A.D.J. Jeremy Stribling, J.D. Kubiatowicz, Exploiting routing redundancy via structured peer-to-peer overlays, in: Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP 2003), 2003, pp. 246–257.

[22] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, H. Yu, OpenDHT: a public DHT service and its uses, in: Proceedings of ACM SIGCOMM 2005, ACM Press, 2005, pp. 73–84.

[23] A. Montresor, M. Jelasity, O. Babaoglu, Chord on demand, in: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005), IEEE Computer Society, Konstanz, Germany, 2005, pp. 87–94.

[24] Y. Koren, Embedder, http://www.research.att.com/yehuda/index_programs.html.

[25] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, D. Terry, Epidemic algorithms for replicated database maintenance, in: Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87), ACM Press, Vancouver, British Columbia, Canada, 1987, pp. 1–12.

[26] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (1) (2002) 47–97.

[27] PeerSim, http://peersim.sourceforge.net/.

[28] K.P. Gummadi, S. Saroiu, S.D. Gribble, King: estimating latency between arbitrary internet end hosts, in: Internet Measurement Workshop (SIGCOMM IMW), 2002.

[29] A. Montresor, M. Jelasity, O. Babaoglu, Decentralized ranking in large-scale overlay networks, in: Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW 2008), 2008.

[30] S. Kalidindi, M.J. Zekauskas, Surveyor: an infrastructure for Internet performance measurements, in: Proceedings of INET'99, San Jose, CA, USA, 1999.

[31] M. Castro, M. Costa, A. Rowstron, Performance and dependability of structured peer-to-peer overlays, in: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04), IEEE Computer Society, 2004.

Mark Jelasity received his Ph.D. in 2001 from the University of Leiden, The Netherlands. From 2000 until 2006 he worked in various institutions including the Free University of Amsterdam, the University of Bologna and INRIA. He currently holds a senior research scientist position in the Research Group on Artificial Intelligence of the University of Szeged, Hungary, and the Hungarian Academy of Sciences. His research interest lies in self-organizing algorithms and their applications in large-scale distributed systems.

Alberto Montresor is Associate Professor of Computer Science at the University of Trento, Italy. He received his Ph.D. in 2000 from the University of Bologna, Italy, where he designed Jgroup, a partition-aware group communication system. He served as Research Associate in Bologna until 2005, when he moved to Trento. He is author of more than 50 papers in international conferences and journals; he is been particularly active in the fields of peer-to-peer and self-organizing systems, having served as program co-chair of P2P'06 and SASO'09, and as general chair of P2P'07.

Author's personal copy

Ozalp Babaoglu is Professor of Computer Science at the University of Bologna, Italy. He received a Ph.D. in 1981 from the University of California at Berkeley where he was a principal designer of BSD Unix. He is the recipient of 1982 Sakrison Memorial Award, 1989 UNIX International Recognition Award and 1993 USENIX Association Lifetime Achievement Award for his contributions to the UNIX system community and to Open Industry Standards. Before moving to Bologna in 1988, Babaoglu was an Associate Professor in the Department of Computer Science at Cornell University. He is active in several European research projects in distributed computing and complex adaptive systems. Babaoglu is an ACM Fellow and serves on the editorial boards for ACM Transactions on Computer Systems, ACM Transactions on Autonomous and Adaptive Systems and Springer-Verlag Distributed Computing.