T-Man: Gossip-Based Overlay Topology Management[*]

Márk Jelasity[**] and Ozalp Babaoglu

University of Bologna,
Dipartimento di Scienze dell'Informazione,
Mura Anteo Zamboni 7, 40126 Bologna, Italy
{jelasity, babaoglu}@cs.unibo.it

Abstract. Overlay topology plays an important role in P2P systems. Topology serves as a basis for achieving functions such as routing, searching and information dissemination, and it has a major impact on their efficiency, cost and robustness. Furthermore, the solution to problems such as sorting and clustering of nodes can also be interpreted as a topology. In this paper we propose a generic protocol, T-Man, for constructing and maintaining a large class of topologies. In the proposed framework, a topology is defined with the help of a *ranking function*. The nodes participating in the protocol can use this ranking function to order any set of other nodes according to preference for choosing them as a neighbor. This simple abstraction makes it possible to control the self-organization process of topologies in a straightforward, intuitive and flexible manner. At the same time, the T-Man protocol involves only local communication to increase the quality of the current set of neighbors of each node. We show that this bottom-up approach results in fast convergence and high robustness in dynamic environments. The protocol can be applied as a standalone solution as well as a component for recovery or bootstrapping of other protocols.

1 Introduction

In large, dynamic, fully distributed systems, such as peer-to-peer (P2P) networks, nodes (peers) must be organized in a connected network to be able to communicate with each other and to implement functions and services. The neighbors of the nodes—the "who is connected to whom", or "who knows whom" relation—define the *overlay topology* of the distributed system in question. This topology can dynamically change in time, and in every time point, it defines the possible interactions between the nodes.

Although it would be desirable, it is typically very difficult to ensure that all nodes are aware of every other participating node in the system. The reason is

[*] This work was partially supported by the Future and Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923) and DELIS (IST-2002-001907).

[**] Also with MTA RGAI, SZTE, Szeged, Hungary.

S.A. Brueckner et al. (Eds.): ESOA 2005, LNAI 3910, pp. 1–15, 2006.

that the set of participating nodes changes quickly, and (due to the large number of nodes) it is not feasible to maintain a complete list of the nodes. This means that all nodes are aware of only a limited subset of other nodes, so efficient and robust algorithms are necessary to create, maintain and optimize the topology.

Overlay topology forms the basis for, or has a major impact on many functions. It is well known that functions such as searching, routing, information dissemination, data aggregation, etc, need special topologies for good performance and high efficiency. Furthermore, solutions to other problems including sorting and clustering can be readily expressed as topologies. For example, in the case of sorting, we are looking for a linear structure that represents some total ordering relation. For all these functions, numerous topologies have been suggested and even more protocols to construct and repair them have been proposed.

Motivated by these observations, we consider topology management as a general purpose function that is desirable in distributed systems. In this paper we specifically target very large scale and highly dynamic systems. Key requirements of topology management in such environments include robustness, scalability, flexibility and simplicity. Besides, it is a great advantage if a topology manager is flexible enough to allow for changing the managed topology at run time *on demand*, without having to develop a new protocol for each possible topology from scratch. Since topology is a very general abstraction, that can be used to express solutions to problems and to enhance and support other functions, such functionality would allow us to increase the efficiency of deploying fully distributed application dramatically. We would need only one running topology component and the application area of the system could be changed at run time whenever necessary. With a protocol that supports quickly changing topologies, it even becomes possible to automatically *evolve* topologies through, for example, an evolutionary process.

In this paper we propose a generic protocol, T-Man, with the aim of fulfilling the requirements outlined above. The desired topology is described using a single ranking function that all nodes can apply to order any subset of potential neighbors according to preference for actually being selected as a neighbor. Using only local gossip messages, T-Man gradually evolves the current topology towards the desired target structure with the help of the ranking function. We show experimentally that the protocol is scalable and fast, with convergence times that grow only as the logarithm of the network size. These properties allow T-Man to be practical even when several different topologies have to be created on demand, and also in dynamic systems where the set of nodes or their properties change rapidly. Additionally, the general formulation of the ranking function allows us to deal with a wide range of different topologies.

Although this work is concerned mainly with exploring the basic properties of T-Man by examining simple topologies like ring, mesh and binary tree, it is possible to illustrate its practicality with more realistic applications. We briefly outline three such applications: sorting, clustering and a distributed hash table (DHT).

Related work includes gossip-based protocols, that have gained notable popularity in various contexts [1, 2, 14]. In this paper we suggest a novel application of the gossip communication model to solve the topology management problem. Issues related to topology management itself have also received considerable attention. Examples from the vast literature include DHTs [7, 11, 13], unstructured overlays [9,3], and superpeer topologies [16]. As for topology construction, Massoulié and Kermarrec [6] propose a protocol to evolve a topology that reflects proximity, Voulgaris and van Steen [15] propose a method to jump-start Pastry. Unlike these specific solutions, T-Man is a generic framework and can be used to construct and maintain a large class of different topologies quickly in a simple and scalable manner.

2 The Problem

We assume that we are given a (perhaps random) overlay network, and we are interested in constructing some desirable topology by connecting all nodes in the network to the right neighbors. The topology can be defined in many different ways and it will typically depend on some properties of the nodes like geographical location, semantic description of stored content, storage capacity, etc. We need a formal framework that is simple yet powerful enough to be able to capture most of the interesting structures. Our proposal is the *ranking function* that defines the target topology through allowing all nodes to sort any subset of nodes (potential neighbors) according to preference to be selected as their neighbor.

For a more formal definition, let us first define some basic concepts. 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. Nodes maintain addresses of other nodes through *partial views* (*views* for short), which are sets of c *node descriptors*. In addition to an address, a node descriptor contains a *profile*, which contains those properties of the nodes that are relevant for defining the topology, such as ID, geographical location, etc. The addresses contained in views at nodes define the links of the *overlay network topology*, or simply the *topology*. Note that parameter c defines the node degree of the overlay network and is uniform for all nodes.

We can now define the *topology construction problem*. The input of the problem is a set of N nodes, the view size c and a *ranking function* R that can order a list of nodes according to preference from a given node. The ranking function R takes as parameters a base node x and a set of nodes $\{y_1, \ldots, y_m\}$ and outputs a set of orderings of these m nodes. The task is to construct the views of the nodes such that the view of node x, denoted view_x, contains exactly the first c elements of a "good" ranking of the entire node set, that is, $R(x, \{\text{all nodes except } x\})$ contains a ranking that starts with the elements of view_x. We will call this topology the *target topology*.

In the presence of churn (ie, when nodes constantly join and leave the overlay network) we talk about maintenance of the target topology instead of construction. Instead of a formal definition, we define the problem as staying "as close as possible" to the target topology. The actual figures of merit to characterize maintenance can be largely application dependent in this case.

One (but not the only) way of obtaining ranking functions is through a distance function that defines a metric space over the set of nodes. The ranking function can simply order the given set according to increasing distance from the base node. Let us define some example distance-based topologies of different characteristics. From now on, to simplify our language and notation, we use the nodes and their profiles interchangeably.

Line and ring. The profile of a node is a real number. The distance function for the line is $d(a, b) = |a - b|$. In the case of a ring, profiles are from an interval $[0, N]$ and distance is defined by $d(a, b) = \min(N - |a - b|, |a - b|)$ Ranking is defined through this distance function as described above.

Mesh, tube and torus. The 1-dimensional topology defined above can be easily generalized to arbitrary dimensions to get for example a mesh or a torus. The profiles are two-dimensional real vectors. The distance for the mesh is the Manhattan distance. It is given by calculating the 1-dimensional distance described above along the two coordinates and returning the sum of these distances. Applying the periodic boundary condition (as for the ring) results in a tube for one coordinate and a three dimensional torus for both coordinates.

Binary tree. A low diameter topology can be constructed from a binary tree: the profiles are binary strings of length m, excluding the all zero string. Distance is defined as the shortest path length between the two nodes in the following undirected rooted binary tree. The string $0 \ldots 01$ is the root. Any string $0a_2 \ldots a_m$ has two children $a_2 \ldots a_m 0$ and $a_2 \ldots a_m 1$. Strings starting with 1 are leafs. This topology is of interest because (unlike the previous ones) it has a very short (logarithmic) diameter of $2m$.

There are very important ranking functions that cannot be defined by a global distance function, therefore the ranking function is a more general concept than distance. The ranking functions that define sorting or proximity topologies belong to this category. Examples will be given in Section 6.1.

3 The Proposed Solution

The topology construction problem becomes interesting when c is small and the number of nodes is very large. Randomized, gossip-based approaches in similar settings, but for other problem domains like information dissemination or data aggregation, have proven to be successful [2, 4]. Our solution to topology construction is also based on a gossip communication scheme.

3.1 The Protocol

Each node executes the same protocol shown in Figure 1. The protocol consists of two threads: an active thread initiating communication with other nodes, and a passive thread waiting for incoming messages.

Each nodes maintains a view. The view is a set of node descriptors. A call to MERGE(view$_1$,view$_2$) returns the union of view$_1$ and view$_2$.

```
do at a random time once in each
consecutive interval of T time units
  $p$ ← selectPeer()
  myDescriptor ← (myAddress,myProfile)
  buffer ← merge(view,{myDescriptor})
  buffer ← merge(buffer,rnd.view)
  send buffer to $p$
  receive buffer$_p$ from $p$
  buffer ← merge(buffer$_p$,view)
  view ← selectView(buffer)
```

(a) active thread

```
do forever
  receive buffer$_q$ from $q$
  myDescriptor ← (myAddress,myprofile)
  buffer ← merge(view,{myDescriptor})
  buffer ← merge(buffer,rnd.view)
  send buffer to $q$
  buffer ← merge(buffer$_q$,view)
  view ← selectView(buffer)
```

(b) passive thread

Fig. 1. The T-Man protocol

The two key methods are selectPeer and selectView. Method selectPeer uses the current view to return an address. First, it applies the ranking function to order the elements in the view. Next, it returns the first descriptor (according to this ordering) that belongs to a live node. Method selectView(buffer) also applies the ranking function to order the elements in the buffer. Subsequently, it returns the *first c elements* of the buffer according to ranking order.

The underlying idea is that in this manner nodes improve their views using the views of their current neighbors, so that their new neighbors will be "closer" according to the target topology. Since all nodes do the same concurrently, neighbors in the subsequent topologies will be gradually closer and closer. This also means that the views of the neighbors will keep serving as a useful source of additional, even better links for the next iteration.

Last but not least, we need to explain the origin and role of the buffer rnd.view. This buffer contains a random sample of the nodes from the entire network. It is provided by a *peer sampling service* [3]. The peer sampling service described in [3] is implemented in a very similar fashion: nodes periodically exchange their random views and update their local views thereby creating a new random sample. These random views define an approximately random overlay network. The buffer rnd.view is the current set of neighbors in this random overlay network. The peer sampling service is extremely robust to failure and maintains a connected network with a very high probability.

The role of the random buffer is most important in large diameter topologies. In this case, if a node has a low quality neighbor set and if most of the rest of the nodes have a high quality neighbor set (forming a large diameter topology, e.g., a ring), then this node needs to perform many exchanges until it can reach the optimal set of neighbors, because the speed of "finding its neighborhood" is related to the diameter of the topology. The random buffer adds long range links that help speeding up convergence.

Although the protocol is not synchronous, it is often convenient to refer to *cycles* of the protocol. We define a cycle to be a time interval of $T/2$ time units where T is the parameter of the protocol in Figure 1. Note that during a cycle, each node is updated once on the average.

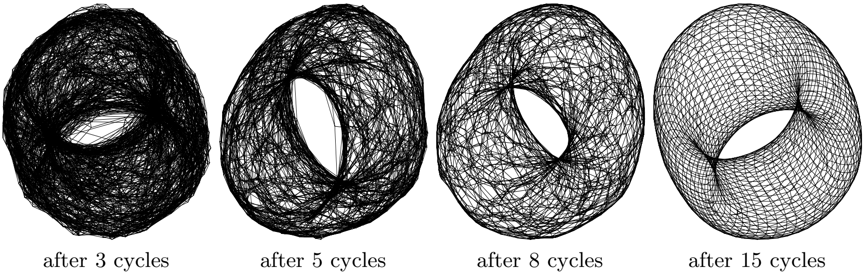


Fig. 2. Illustrative example of constructing a torus over $50 \times 50 = 2500$ nodes, starting from a uniform random topology with $c = 20$. For clarity, only the nearest 4 neighbors (out of 20) of each node are displayed.

Figure 2 illustrates the results of the protocol when used to construct a small torus (visualizations were done using [5]). For this example, it is clear that 15 cycles are sufficient for convergence, and the target topology is already evident even after very few cycles. As we will see, T-Man proves to be extremely scalable and the time complexity of the protocol remains in this order of magnitude even for a million nodes.

3.2 Optimizations

We can increase the performance of the protocol by applying two well known techniques described in [1]. First, we set a connection limit of 1, that is, in each interval of T time units (i.e., two cycles), we allow each node to receive at most one connection. Since each node also initiates one connection in this interval, this means that during two cycles, each node communicates at most twice. Second, we also apply *hunting*, that is, instead of trying only one peer, each node actively keeps looking for nodes (from the current view) that have not exceeded their connection limit in the given interval. Our preliminary experiments indicate that these techniques noticeably improve the convergence speed of T-Man.

As another possibility for optimization, note that a node does not need to send the entire buffer containing the union of the fresh descriptor, the old view and the random buffer. In fact, the contacted node will use at most the c closest items from this buffer so it is sufficient to first sort the buffer applying the ranking function *of the peer*, and sending the first c items. Since all nodes use the same ranking function, they can easily emulate the ordering from the point of view of any other node.

4 Simulation Experiments

All the simulation results presented in this paper were produced using PeerSim, an open-source simulator developed at the University of Bologna [10].

We examine the three distance-based ranking functions that define the ring, torus and binary tree topologies, as defined in Section 2. The motivation of this choice is that the ring is a large diameter topology and it is relevant for the sorting application (Section 6.1), the binary tree is of a logarithmic diameter and the torus is relevant in proximity problems being based on a 2-dimensional grid. The network sizes (N) examined are 2^{14}, 2^{17} and 2^{20}. We initialize the profiles of the nodes in a regular manner, that is, in the case of the ring topology, we assign the numbers $1, 2, \ldots, N$ to the nodes, and likewise for the torus $((1,1),(1,2),\ldots,(\sqrt{N},\sqrt{N}))$ and the binary tree (all binary strings of length $\log_2 N$).

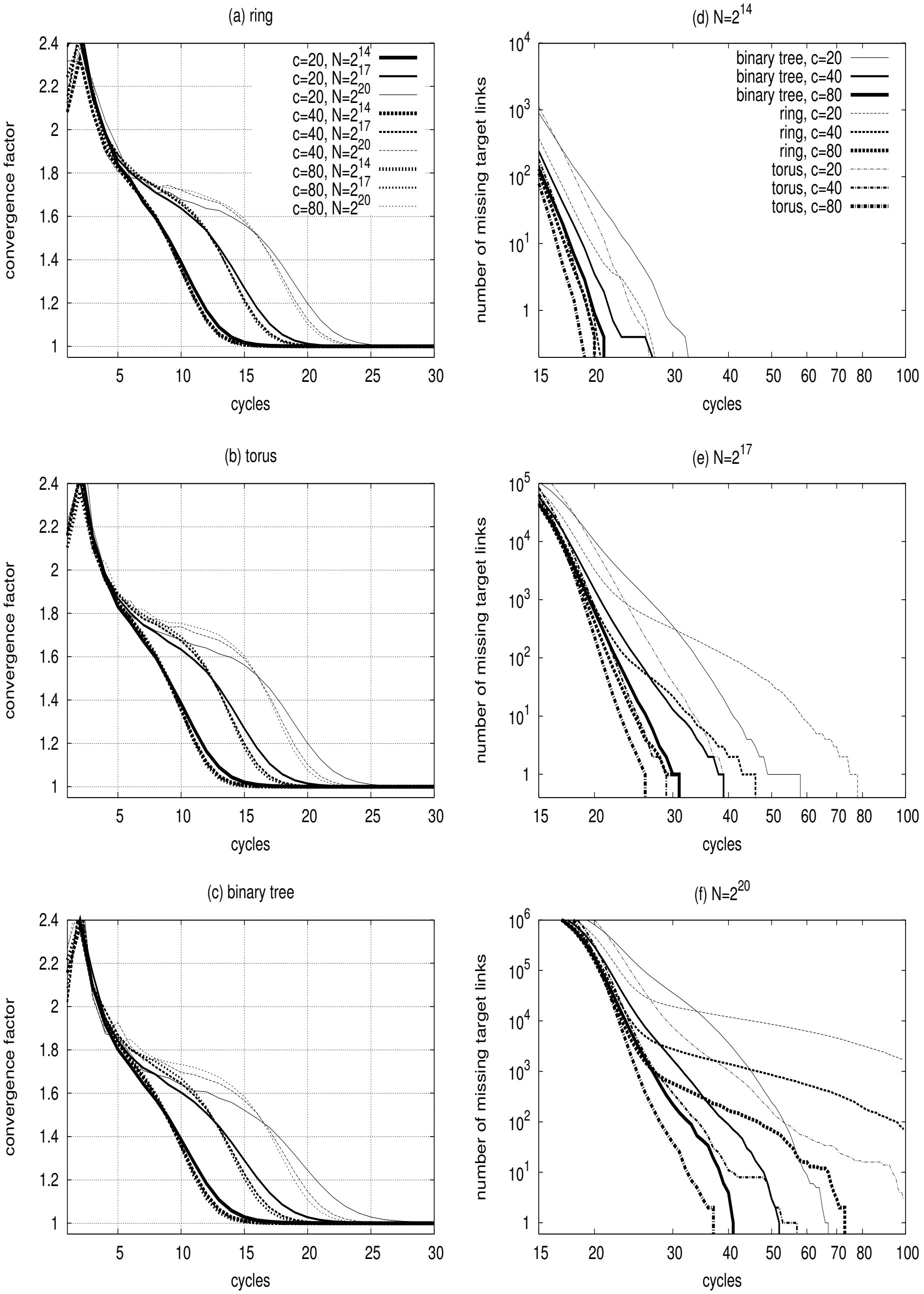
This regularity is not critical for the success of the protocol. On the contrary, one of the important applications is sorting an arbitrary set of numbers, as we argue in Section 6.1. However, this controlled setting allows us to monitor the dynamics of the protocol in a more informed manner as the distance function becomes equivalent to the hop count in the *target topology* defined by the links that connect nodes at distance 1 (the target links). During the experiments we focus on the dynamics of the number of target links that are found. As a measure of performance, the *convergence factor* is defined as the factor by which the number of target links found increases from one cycle to the next. Note that a constant convergence factor means exponential increase.

The NEWSCAST protocol was used as the implementation of the peer sampling service [3], which works very similarly to T-MAN maintaining a dynamic random overlay and using it to provide random peers. The NEWSCAST protocol is extremely scalable and robust, and its communication cost is similar to that of T-MAN. The cache size of NEWSCAST was 30 and its cycle length was identical to that of T-MAN. In this section, we focus on convergence starting from a random network, that is, the views are initialized at random and the nodes start to run the protocol at the same time. In Section 5 we examine the effect of churn, that is, with nodes continuously joining and leaving the network.

The results are shown in Figure 3. The results clearly indicate a logarithmic relationship between network size and convergence speed. This is illustrated especially well by the plots comparing the convergence factor for different network sizes as a function of time. We can see a constant shift of convergence time when the network size is increased by the same multiplicative factor (2^3). Quite interestingly, initial convergence does not depend on the view size c, nor does it depend on the characteristics of the target topology.

When the topology has already converged, the few nodes that are still incorrectly linked can be though of as "climbing" on the converged structure during the consecutive cycles of T-MAN. This means that in this end phase convergence time does depend on the target topology. Accordingly, in the binary tree topology, we observe rapid convergence. In fact, logarithmic convergence, because the evolved structure allows for efficient routing, being low diameter. Similar arguments hold for the torus, only the convergence time there is not logarithmic but grows with the square root of the network size in the worst case. In both cases, we can observe fast convergence even for the smallest view size.

The case of the ring is different, because the target topology has a large diameter that grows linearly with the network size, so the remaining few misplaced


(a) ring

(b) torus

(c) binary tree

(d) N=2^{14}

(e) N=2^{17}

(f) N=2^{20}

Fig. 3. Comparison of convergence speed in the initial phase and in the final phase for network sizes $N = 2^{14}, 2^{17}, 2^{20}$ and $c = 20, 40, 80$ for the ring, torus and binary tree topologies. The results displayed are averages of 10 runs for $N = 2^{14}$ and $N = 2^{17}$, and show a single run for the case $N = 2^{20}$.

nodes reach their destination slower. Still, for $c = 80$ we have *perfect* convergence after cycle 72 even for $N = 2^{20}$, and only a small percentage of target links are missing with $c = 20$ and $c = 40$. For the smaller network sizes we always observe full convergence in less than 80 cycles, independently of the characteristics of the target topology.

5 Self-healing

In this section we consider scenarios with churn, that is, with nodes constantly leaving and joining the network. We introduce a simple extension of the protocol to increase its adaptivity and subsequently we experimentally evaluate the proposed solution.

5.1 Age-Based View Update

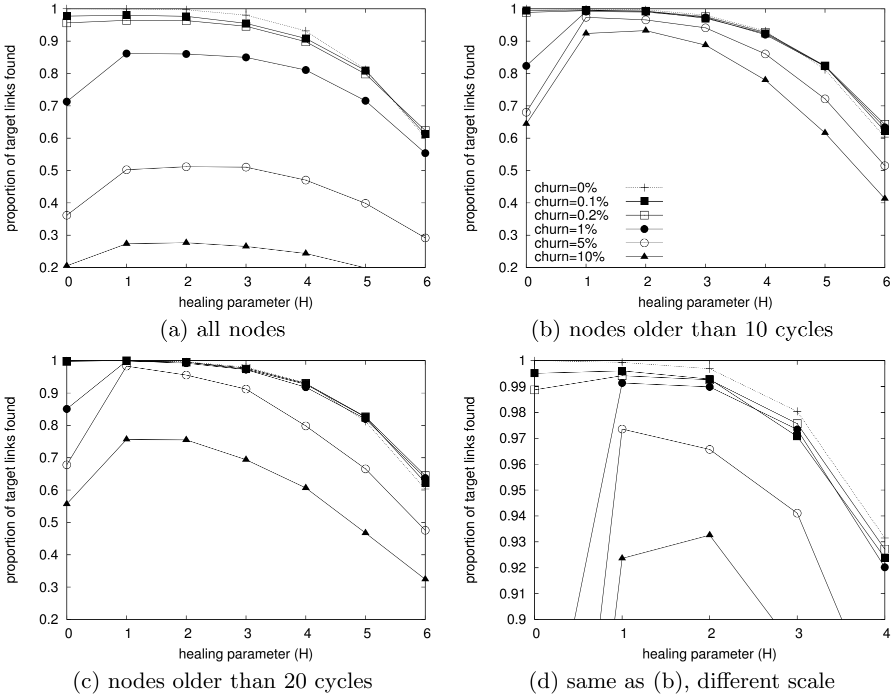
We extend the protocol given in Figure 1 by a simple technique to handle dynamic environments. The key idea is that we remove a few old descriptors from the view in each cycle. As a result, we expect to decrease the number of "dead links", that is, descriptors describing nodes that are no longer in the network. By decreasing the number of dead links, we expect to increase the quality of the views of the live nodes.

To implement this idea, the node descriptors stored in the view must also contain an age field. This field is initialized to be zero (when the node adds its own descriptor to the buffer to send) and increased for all view entries every time the node communicates. Before merging the view to the buffer to be sent, each node removes the H oldest descriptors from the view. Finally, the MERGE operation has to be modified to prefer the freshest item when removing duplicate items.

5.2 Experimental Results

To test the efficiency of this solution, we performed experiments with different scenarios involving churn. In all experiments, network size was 10^4, and $c = 20$. The cache size of NEWSCAST (the applied peer sampling service) was 30 and its cycle length was identical to that of T-MAN. Churn was modeled by removing a given percentage of randomly selected nodes from the network in each cycle and replacing them with new nodes that were initialized with random links. The ranking function defined a 1-dimensional ring. However, due to churn, node profiles were initialized by a random 62 bit integer, not regularly as in Section 4. For this reason, to define a connected ring, we applied the direction dependent version of the ranking function as described in Section 6.1.

The results of the experiments, illustrating various settings for the healing parameter H and churn rates, are shown in Figure 4. First of all, note that the churn rates can be considered very high. If we set the cycle length parameter $T/2 = 10s$, then, based on the Gnutella trace described in [12], the churn rate is less than 0.2% per cycle. In this light, 5% or especially 10% churn is extremely high.



(a) all nodes

(b) nodes older than 10 cycles

(c) nodes older than 20 cycles

(d) same as (b), different scale

Fig. 4. Experimental results in the presence of churn. $N = 10,000$, $c = 20$.

Each point in the plots was generated by running the respective scenario until 300 cycles, to make sure that the proportion of correctly identified target links converges, and taking this converged value. The first observation is that over all the nodes, high churn rates decrease the overall quality of the views in the network quite significantly. However, for such high churn rates, the network always contains many newcomers. If we consider the convergence times presented in Section 4, we can note that for a newcomer at least 10 cycles are necessary to optimize its view, so we cannot reasonably expect better performance according to this measurement. However, if we restrict ourselves to nodes that are old enough, we get a very different picture. For $H = 1$ and $H = 2$, we observe a very good quality network even for churn rates of 10% which is especially notable because the expected lifetime of a node in this network is only 10 cycles. In fact, the number of nodes older than 10 cycles is around 3500, one third of the network.

We can also observe that too aggressive "healing" in fact damages the network even when there is no churn. The setting $H = 6$ is consistently worse than any other setting. However, the positive effect of self-healing can be observed when comparing the case of $H = 1$ with $H = 0$ (no healing). This consistently results in a significant performance improvement. In general, $H = 1$ appears to be the best choice, except in the most extreme churn where $H = 2$ is slightly better.

As a final note, it is interesting to observe that performance is in fact no so bad even without the application of the healing method ($H = 0$). This is due to the fact, that in our scenarios the overall number of dead links is guaranteed not to decrease below 50%. To see this, consider the case when the proportion of dead links is p in the network and we remove n nodes, replacing them by n new nodes, that have links to random *live* nodes. Due to the removal, the number of dead links on average decreases by ncp while it increases by the number of links that pointed to these nodes: on average $nc(1-p)$. if we assume that all nodes in the network have the same in-degree (it is true for our ranking function here). This dynamics always converges to $p = 0.5$. This fact emphasizes the importance of the bootstrapping method, especially in the presence of extreme churn.

6 Application Examples

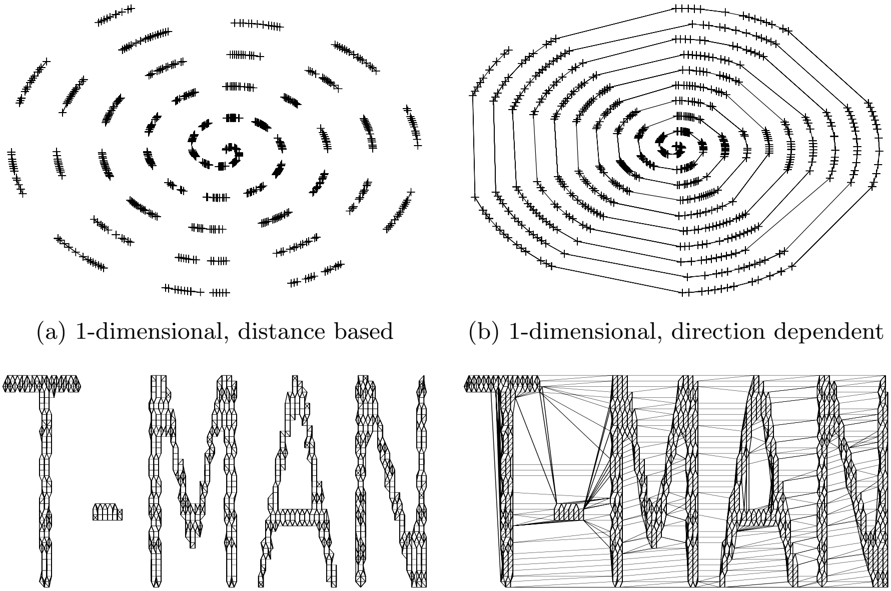
The primary goal of this section is to underline the generality of the approach by outlining the main ideas in using T-Man to solve some potentially important applications.

6.1 Clustering and Sorting

So far we have considered rather artificial settings to understand the behavior of T-Man better. In particular, the profiles of the nodes were initialized in a regular manner. In practice this will hardly happen. Typically, the properties of the nodes follow some specific distribution. This distribution can also be “patchy”, there can be dense clusters separated by unrepresented regions in the profile space. Very informally, when applying T-Man in such a setting using a simple distance-based ranking function, the resulting topology will be clustered and most likely disconnected, because nodes in the same cluster will eventually be linked to each other only. An illustrative example is given in Figure 5 for the 1- and 2-dimensional cases. In many applications, like clustering based on semantic, geographic or any other definition of proximity, this property can be exploited to find the desired clusters.

In the case of the sorting problem, where we would like to connect each node to those nodes that directly preceed and follow them according to some total ordering relation, we need to prevent clustering. This can be achieved by the following direction dependent ranking. First, separate the set of nodes to be ranked into two groups: one that is to the left, and another that is to the right of the base node. Order these two sets according to the underlying desired ordering. Merge the ordered sets so that a node that had index i in any of the sets is assigned index $2i$ or $2i + 1$ in the final ranking, choosing randomly between these two possibilities. Applying the 1-dimensional ranking function makes it possible to practically reduce the sorting problem to the one dimensional topology construction problem that we have studied extensively in Section 4. In Section 5 we used exactly this sorting method as a ranking function.

Direction dependent ranking can be easily extended to other problems, for example, creating a *connected* topology in two dimensions that reflects geo-



(a) 1-dimensional, distance based

(b) 1-dimensional, direction dependent

(c) 2-dimensional, distance based

(d) 2-dimensional, direction dependent

Fig. 5. Illustrative example of converged topologies obtained with distance-based and direction dependent ranking, with $N = 1000$, $c = 20$. The line is displayed as spiral for convenience. Only the closest 2 and 4 links are shown from each node for the 1- and 2-dimensional example, respectively.

graphical proximity. In this case, a node divides the space into four quarters, and classifies each node to be ranked into four categories accordingly. The node then sorts the nodes in each class according to an underlying distance function, and produces the ranking similarly to the two dimensional case: if a node has index i in any of the four quarters, then it will be assigned an index randomly from between $4i$ and $4i + 3$.

The effect of direction dependent ranking is illustrated by two small examples in Figure 5. In the case of both the distance based and direction dependent ranking the nodes are mapped to points forming the plotted structures: equal length intervals in 1-dimension and letter-shaped clusters in 2-dimensions. The profile of the nodes is defined as their 1- or 2-dimensional coordinates, respectively. Observe the clustering effect with the distance based ranking and, with direction dependent ranking, the perfect sorting in 1-dimension and the connected topology in 2-dimensions.

6.2 A DHT

As an illustration, we very briefly present a simplistic way of evolving a distributed hashtable (DHT) topology with T-Man. The ranking function for the target topology is defined by a XOR-based distance. The distance we use is not

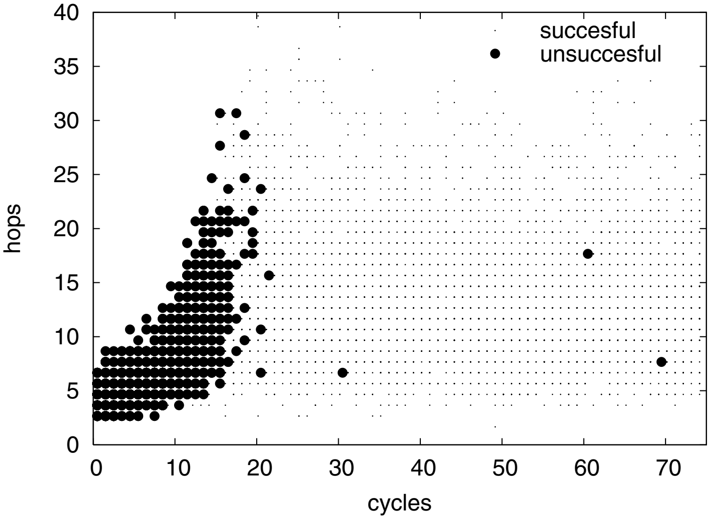


Fig. 6. Hop count and success of routing as a function of time (cycles). Each point represents the result of the routing algorithm from a random node to another random node. In each cycle 100 experiments are shown, with a small random translation. Node ID-s are random 62 bit integers, network size is $2^{20} (> 10^6)$, size of routing table is 60: 30 from the Pastry-inspired topology, 30 from the ring.

that of [7]. Instead, we define the XOR distance over a set of binary numbers as the number of bit positions in which they have a different digit. This ranking function is responsible for evolving long range links. The idea is that in this topology each node should know about nodes that differ from it in a few digits only, resulting in a link set pointing to ID-s with a varying length of common prefix (including long prefixes); a well known way of achieving efficient routing.

As a backup, we also evolve a sorted ring topology using another instance of T-Man as described above, to maximize the probability that routing is successful. The routing table is composed of the neighbors in these two topologies, and the next hop for a target is selected based on numeric difference between the ID of the target and the table entries (now interpreting ID-s as numbers). We require strictly decreasing difference to avoid loops. The links from the ring topology are used only if no suitable links are available in the XOR-based topology. If the distance cannot be decreased but the target is not found, the routing attempt is failed.

Figure 6 illustrates the convergence of the routing performance while the topology is being evolved, starting from random routing tables. We can observe that the number of missed targets quickly becomes insignificant (from cycle 23 only 3 cases out of the 5300 shown in the figure), and the hop count of both the successful and unsuccessful routes remains low. Note that in our example, assuming a perfect topology, the worst case hop count would be 20.

Finally, note that this approach is mainly for illustration. The protocol presented in [8] for building the Chord [13] DHT represents a more realistic example.

7 Conclusions and Future Work

We have presented a protocol for topology management, T-Man, that is simple, general and fast. Simplicity makes it easier to implement, debug and understand. Generality allows it to be applied as an off-the-shelf component for prototyping or even as a production solution that could be implemented even before the final desired topology is known. In fact, the ranking function can be generated dynamically by users, or by some monitoring service, and the corresponding topology can be created on the fly. Finally, speed makes it possible to construct a topology quickly from scratch (recovery from massive failures or bootstrapping other protocols on demand) or where topology maintenance is in fact equivalent to the continuous re-creation of the topology (for example, due to massive churn).

Our current work is towards the application of T-Man for jump-starting existing DHT implementations and providing them robustness in the presence of massive failures and extreme churn [8]. We are also continuing our study of T-Man at an abstract level to better understand its behavior and characterize its scope and performance. In particular, it would be important to characterize the class of topologies that are "easy" or "difficult" for T-Man. Note that any *arbitrary* topologies can be expressed by at least one appropriate ranking function; in fact in general by many ranking functions: any function that ranks the neighbors in the target topology highest is suitable. This means that the open questions are: which of the possible ranking functions is optimal for a given problem, and how does convergence and the speed of convergence depend on the different topologies. Although the protocol does certainly not work with the same efficiency for all problems, we observed very similar performance for rather different and important topologies, so the empirical results are promising.

References

1. Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, British Columbia, Canada, August 1987. ACM Press.
2. Patrick Th. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, May 2004.
3. Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In Hans-Arno Jacobsen, editor, *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, 2004.
4. Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005.
5. Yehuda Koren. Embedder.
   `http://www.research.att.com/~yehuda/index_programs.html`

6. Laurent Massoulié, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Network awareness and failure resilience in self-organising overlays networks. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS 2003)*, pages 47–55, Florence, Italy, 2003.
7. Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, 2001.
8. Alberto Montresor, Márk Jelasity, and Ozalp Babaoglu. Chord on demand. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, pages 87–94, Konstanz, Germany, August 2005. IEEE Computer Society.
9. Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter peer-to-peer networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 21(6):995–1002, August 2003.
10. PeerSim. http://peersim.sourceforge.net/.
11. Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer-Verlag, 2001.
12. Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Systems Journal*, 9(2):170–184, August 2003.
13. 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)*, pages 149–160, San Diego, CA, 2001. ACM, ACM Press.
14. Robbert van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.
15. Spyros Voulgaris and Maarten 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)*, number 2867 in Lecture Notes in Computer Science. Springer, 2003.
16. Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, Los Alamitos, CA, March 2003. IEEE Computer Society Press.