Chord on Demand*

Alberto Montresor
University of Bologna, Italy
montresor@cs.unibo.it

Mark Jelasity†
University of Bologna, Italy
jelasity@cs.unibo.it

Ozalp Babaoglu
University of Bologna, Italy
babaoglu@cs.unibo.it

Abstract

Structured peer-to-peer overlay networks are now an established paradigm for implementing a wide range of distributed services. While the problem of maintaining these networks in the presence of churn and other failures is the subject of intensive research, the problem of building *them from scratch has not been addressed (apart from individual nodes joining an already functioning overlay). In this paper we address the problem of* jump-starting *a popular structured overlay, Chord, from scratch. This problem is of crucial importance in scenarios where one is assigned a limited time interval in a distributed environment such as Planet-Lab, or a Grid, and the overlay infrastructure needs to be set up from the ground up as quickly and efficiently as possible, or when a temporary overlay has to be generated to solve a specific task* on demand. *We introduce* T-CHORD, *that can build a Chord network efficiently starting from a random unstructured overlay. After jump-starting, the structured overlay can be handed over to the Chord protocol for further maintenance. We demonstrate through extensive simulation experiments that the proposed protocol can create a perfect Chord topology in a logarithmic number of steps. Furthermore, using a simple extension of the protocol, we can optimize the network from the point of view of message latency.*

1. Introduction

Structured overlay networks have received considerable attention recently [3, 12, 16]. A wide range of distributed services and applications can efficiently be implemented on top of structured overlays. The fundamental abstraction that is the basis of numerous applications is *key-based routing* [4]. Key-based routing protocols are based on *routing tables* stored at each node and that are used to forward messages for a specific key towards the destination: the node that is responsible for the given key. The neighborhood relations specified by the routing tables define the overlay topology, whose structure depends on the specific implementation.

While the problem of maintaining these networks in the presence of churn and other failures is the subject of intensive research, the problem of *building* them from scratch has not been addressed apart from handling node joins to an existing overlay. Yet, in some important scenarios, we face the problem of *jump-starting* structured overlays from scratch. This problem gains particular importance if one is assigned a limited time interval in a distributed environment such as PlanetLab [15], or a Grid [5], and the overlay infrastructure needs to be set up from the ground up as quickly and efficiently as possible, or when a temporary overlay has to be generated to solve a specific task *on demand*.

Existing join protocols are 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 [3]. 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).

In this paper we propose a solution to the jump-starting problem called T-CHORDthat is simple, scalable, robust, and efficient. T-CHORD is a protocol for bootstrapping the Chord topology *on demand* starting from an unstructured, uniform random overlay. The purpose of T-CHORD is purely jump-starting the overlay; the constructed network is handed over to the Chord protocol for further maintenance.

T-CHORD is based on an existing topology management protocol called T-MAN, a generic mechanism for building and maintaining a wide range of different topologies, including rings, grids and trees. 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. The resulting protocol is scalable and fast, with convergence times that grow as the

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

† Also with MTA RGAI, SZTE, Szeged, Hungary

logarithm of the network size. Furthermore, it is completely decentralized and extremely robust.

Briefly, T-CHORD works as follows. We assume the existence of a uniform random unstructured overlay network that connects a potentially large set of nodes (e.g., [9]). Nodes are assigned unique IDs from a circular ID space. Starting from the initial random overlay, T-MAN is used to build the leaf ring to be used by Chord for consistent routing. At all nodes, as a "side effect" of its execution (by remembering all the encountered nodes), T-MAN can also provide a larger set of nodes from which Chord fingers can be selected. More details are provided in Section 4.

We have evaluated the topologies obtained by T-CHORD through simulation. The results, presented in Section 5, confirm that the obtained topology is equivalent to (in fact, at times slightly better than) the "optimal" Chord topology (as defined in the Chord protocol specification) based on routing performance: loss rate, hop count and latency.

2. System Model

We consider a network consisting of a large collection of *nodes* that are assigned unique identifiers and that communicate through message exchanges. The network is highly dynamic; new nodes may join at any time, and existing nodes may leave, either voluntarily or by *crashing*. Since voluntary leaves can be trivially managed by simple "logout" protocols, in the following we limit our discussion to node crashes, that are much more challenging. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion.

We assume that nodes are connected through an existing routed network, such as the Internet, where every node can potentially communicate with every other node. To actually communicate, a node has to know the identifiers of a set of other nodes (its *neighbors*). This neighborhood relation over the nodes defines the topology of the *overlay network*. Given the large scale and the dynamism of our envisioned system, neighborhoods are typically limited to small subsets of the entire network. The neighbors of a node (and, thus, the overlay topology) can change dynamically.

3. Chord

We provide a simplified description of the Chord overlay, 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 keys having the same ID negligible.

Node IDs are ordered in an ID space modulo 2^t. We say that ID a *follows* ID b in the ring if $(a - b + 2^m) \mod 2^m < 2^{m-1}$; otherwise, a *preceeds* b. We also define a notion of distance over the ring as follows: $d(a, b) = \min(|a - b|, N - |a - b|)$.

Given an ID a, its *successor* (denoted $succ_1(a)$) is defined as the nearest node whose ID is equal to a or follows a in the ring. Furthermore, the j^{th} successor of i (denoted $succ_j(i)$) is defined recursively as the successor of $succ_{j-1}(i)$. Key k is under the responsibility of node $succ_1(k)$.

Each node maintains two sets of neighbors, called *leaves* and *fingers*. leaves define an l-regular lattice graph, where each node n is connected to its l nearest successors $succ_1(n) \ldots succ_l(n)$. For each node n, its j^{th} finger is defined as $succ(n + 2^j)$, with $j \in [0, t - 1]$.

Routing in Chord works by forwarding messages in the ring following the successor direction: when receiving a message targeted at key k, a node n forwards it to its furthest leaf or finger n' that preceeds (or is equal to) $succ_1(k)$. When the message reaches the destination node $succ_1(k)$, it is *delivered* locally.

Thanks 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 [3]. 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.

4. Jump-starting Chord

T-CHORD is heavily based on the topology management facilities offered by T-MAN [8]. For this reason, we first describe the generic T-MAN algorithm, and then we show how it can be used to build the Chord topology.

4.1. The T-MAN Algorithm

The task of T-MAN is to build some desirable topology by connecting all nodes in the network to their appropriate neighbors. The topology can depend on any properties of the nodes including geographical location, semantic description of stored content, storage capacity, etc. The desired topology is defined through a *ranking function* that nodes can use to sort any subset of nodes (potential neighbors) according to preference to be selected as their neighbor.

Let us first define some basic concepts. Nodes maintain addresses of other nodes through *partial views* (*views* for short), which are sets of *node descriptors*. A descriptor is a pair *(address, profile)*, where the address is the information needed for sending a message to a node (e.g., an IP address and a port number), while the profile contains those properties of the nodes that are relevant for defining a topology, such as ID, geographical location, etc.

We can now define the *topology construction problem*. The input to the problem is a set $\mathcal{N}$ of N nodes, the number of nodes l that constitutes the node degree of the desired overlay topology, and a *ranking function* R that can order a list of nodes according to preference from a given node.

$view \leftarrow rnd.view \cup \{(myAddress, myProfile)\}$

(a) Initialization

do at a random time once in each consecutive interval of T time units
  $p \leftarrow \mathsf{selectPeer}(view)$
  $message \leftarrow \mathsf{extractMessage}(view, p)$
  send $message$ **to** p
  receive $message_p$ **from** p
  $view \leftarrow \mathsf{merge}(message_p, view)$

(b) Active thread

do forever
  receive $message_q$ **from** q
  $message \leftarrow \mathsf{extractMessage}(view, q)$
  send $message$ **to** p
  $view \leftarrow \mathsf{merge}(message_q, view)$

(c) Passive thread

Figure 1. The T-MAN algorithm.

The ranking function R takes as parameters a base node n 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 view of all the nodes such that the view of node n, denoted $view_n$, contains the first l elements of a "good" ranking of the entire node set, that is, the first l elements of some ordering returned by $R(n, \mathcal{N} - \{x\})$ is identical to the first l elements of some ordering returned by $R(n, view_n)$. We will call the topology defined by these l nodes the *target topology*. Note that parameter l defines the node degree of the overlay network and is uniform for all nodes.

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. For example, in T-CHORD, the distance function will be defined as the minimal distance between the node IDs on the ring. Other ranking functions, not based on the concept of distance, are discussed in [8],

The pseudo-code of the T-MAN protocol is shown in Figure 1. Each node executes two threads: an active thread initiating communication with other nodes, and a passive thread waiting for incoming messages. The version presented here is a slight variation of the original algorithm, where the size of the views maintained by nodes is fixed. Here, the view size may grow after each exchange, as the set of received descriptors is merged to the local view.

Let us now describe the three functions called by the protocol. Function $\mathsf{selectPeer}(S)$ executed by node n ranks the nodes in S based on their distance from n itself, and selects a random node from the first m nodes (the nearest ones). Function $\mathsf{extractMessage}(S, q)$ first ranks the nodes in set S from the point of view of q (i.e., based on their distance from q), and subsequently returns the first m elements. Parameter m defines both message size, and the size of the set from which a random peer is selected. In other words, it returns the elements that the selected peer would prefer most from the current view of the node. Finally, $\mathsf{merge}(S_1, S_2)$ returns the union of the descriptor sets S_1 and S_2.

It has been shown that this algorithm converges exponentially to the target topology; i.e., the number of cycles to be executed is $O(\log N)$ in a network of N nodes.

4.2. T-CHORD: From T-MAN to Chord

To build a Chord topology, the ring of leaves must be constructed to guarantee consistency, and fingers must be discovered to improve performance. The version of T-MAN presented in this paper makes it possible to solve both problems in a simple way.

To be able to apply T-MAN each node is assigned a unique identifier in the Chord ID space ($[0, 2^t - 1]$). Initially, the T-MAN view is initialized with a random set of nodes that are obtained from a lightweight membership protocol such as the ones described in [9]. The distance function used by T-MAN is defined over the ring, as described in Section 3: $d(a, b) = \min(|a - b|, N - |a - b|)$. In other words, the preferred neighbors are the nearest successors and predecessors in the ID space. T-MAN is executed for a fixed number of cycles at each node. At each cycle, a node n discovers new nodes (by merging exchange messages to its view) that are progressively nearer to n itself. Given the logarithmic convergence time [8], the number of cycles to be executed can be determined in two ways: (i) either by estimating the network size [10, 13]; or by adopting an upper bound (e.g., a value of 30 cycles is sufficient for networks up to one billion nodes, as extrapolated from Figure 5).

Once T-MAN has completed its execution, node n obtains the leaves by extracting them from its local $view$, selecting the l nearest nodes that follow n. To select fingers, the following algorithm is used: for each exponent $j \in [1, t-1]$, select from the view the node nearest to n whose ID belongs to the interval $[n + 2^j \mod 2^t, n + 2^{j-1} - 1 \mod 2^t]$. In other words, even if we have not been able to discover $succ(n + 2^j)$, which is supposed to be the "ideal" finger based on the Chord definition, we select a node whose distance from n has the same order of magnitude. In this way, the logarithmic routing property of Chord is preserved, as we demonstrate in Section 5.

4.3. T-CHORD-PROX: Network Proximity

Given an exponent $j \in [1, t-1]$, several nodes in the $view_n$ may belong to the finger range $[n+2^j \mod 2^t, n+2^{j-1}-1 \mod 2^t]$. In T-CHORD, the finger nearest to n with respect to the ID space has been selected among them, to better approximate the original Chord definition.

An alternative selection mechanism could be based on communication proximity, i.e. on the latency measured be-

tween nodes [6]. This would enable the construction of low-latency routing paths between nodes, improving the overall routing performance of the network.

We propose here T-CHORD-PROX, a variant of T-CHORD based on proximity. The finger selection algorithm works as follows: let $S_j(n)$ be the the set of nodes contained in the view of n whose id is contained in the range $[n + 2^j \mod 2^t, n + 2^{j-1} - 1 \mod 2^t]$. Node n picks p nodes at random from S_j (or the entire S_j set, if its size is less or equal than the parameter p), and measures the latency by sending *distance probes* to them. A distance probe can be implemented as a simple ping-pong exchange, or may be a more complex protocol involving more exchanges [1].

This very simple protocol requires a number of hops similar to the original Chord, but outperforms it in terms of latency.

5. Experimental Results

We performed extensive simulation experiments in order to compare the jump-started overlay to the perfect Chord topology, and to characterize the scalability and robustness of our protocols. All of the experimental results were obtained using PEERSIM, a simulator developed by us and optimized for our gossip-based protocols [11, 14].

5.1. Experimental Settings

In all experiments, all nodes are initialized with a random view obtained from the NEWSCAST protocol [9]. Subsequently, T-MAN is run with the ranking function described in Section 4, to create an ordered ring, and to collect long range links as well. When T-MAN reaches a pre-specified number of cycles, each node runs T-CHORD locally to extract its routing tables from the T-MAN view, creating the Chord topology.

We focus on the *routing performance* of the obtained overlay. Three routing metrics have been taken into consideration. *Hop count* is the number of nodes that are traversed by a message to reach its destination. In case of failures, message timeouts (*failed hops*) are counted separately. *Delivery delay* measures the time needed to reach the destination. Our latency model is based on the King dataset [7], that provides end-to-end latency measurements for a set of 1740 routers. Each node is attached through a 1ms link to a randomly selected router [16]. In case of failures, a time equal to twice the latency is added to the total delay in order to simulate timeouts. *Loss rate* is the fraction of nodes that do not reach the destination node.

Since our goal is to jump-start Chord, the baseline routing performance is defined by the perfect Chord topology over the same set of nodes. We construct this topology offline, using the specification of the Chord protocol, and we compare the performance of this ideal topology with the ones generated by T-CHORD. We emphasize again that our goal is not to develop a novel routing mechanism or a new structured overlay: our goal is to create a Chord topology efficiently from scratch.

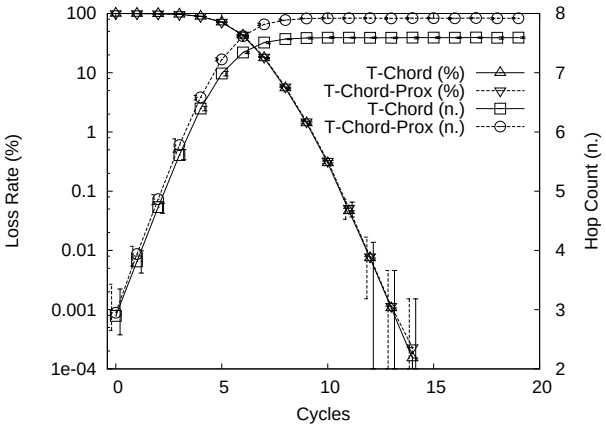

Figure 2. Loss rate and hop count as a function of the number of T-MAN cycles executed

Besides routing performance, we also need to measure communication overhead for building the topology. In case of T-CHORD without proximity, communication costs are given just by T-MAN exchanges. Given the periodic nature of T-MAN, these costs can be easily computed: each T-MAN node sends one message and receives one message on the average per cycle, with m descriptors included in each message. T-MANis run for $O(\log N)$ cycles. In T-CHORD-PROX, the cost of latency probes must also be considered.

Unless stated otherwise, all figures are based on the following parameters: network size $N = 2^{16}$ nodes, message size $m = 10$, leaf set size $l = 10$, maximum number of probes per routing table entry $p = 5$. In all figures, 20 individual experiments were performed. Average values for each of the metrics are shown; error bars are used to quantify variance between experiments. To aid the visualization, some of the bars are shifted horizontally slightly.

5.2. Convergence

The routing performance of the topologies obtained by T-CHORD depends on the number of T-MAN cycles executed before the routing tables are built. In particular, the leaf ring must be completed in order to guarantee the correct delivery of all messages. This is illustrated in Figure 2, where the loss rate and the observed hop count for T-CHORD and T-CHORD-PROX are shown as a function of the number of T-MAN cycles that have been run. Initially, all messages are lost: local views contain only random nodes, so the routing algorithm is unable to deliver messages. The loss rate rapidly decreases, however, reaching 0 after only 14 cycles. At that point, the leaf ring is completely formed in

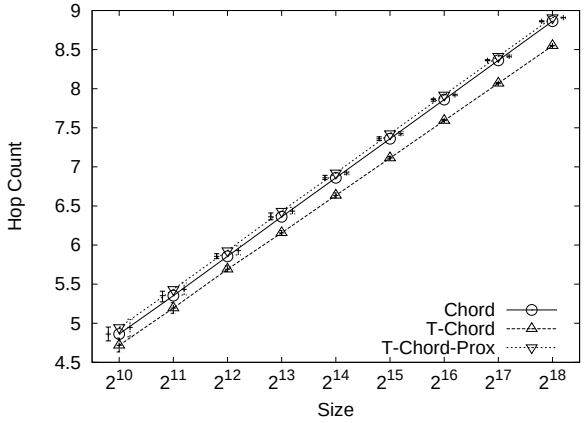

Figure 3. Hop count as a function of network size

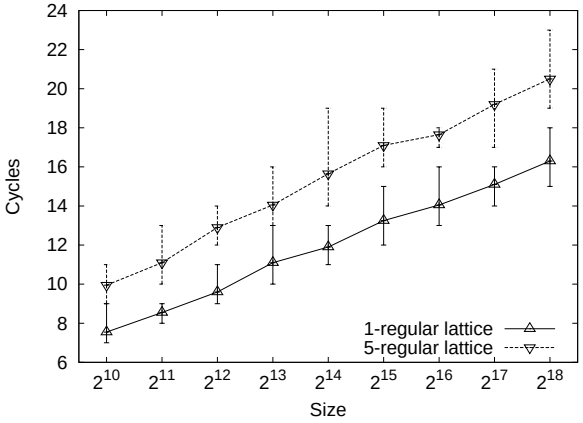

Figure 5. Convergence time as a function of network size

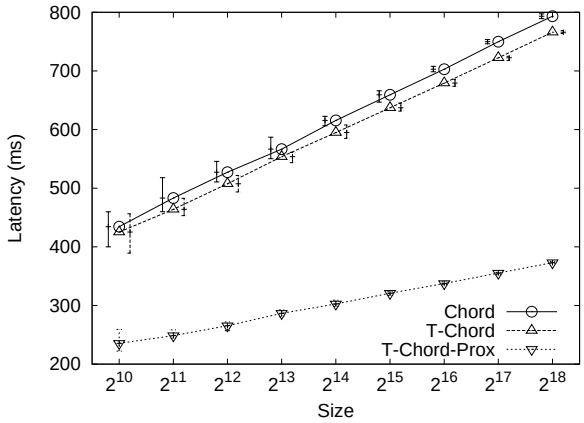

Figure 4. Message delay as a function of network size

all our experiments. Note that the curves for T-CHORD and T-CHORD-PROX overlap almost completely.

Regarding hop counts, the results confirm that the quality of the routing tables stabilizes after few cycles, for both versions of T-CHORD. Latency (not shown for space reasons) follows a similar behavior. The increasing tendency of the hop count curves is explained by the fact that in the beginning, in spite of the low quality overlay, a few messages reach their destination "by chance" in a few hops, while most of the messages are lost.

5.3. Scalability

The experiments discussed so far were run in a network with a fixed size (2^{16} nodes). To assess the scalability of T-CHORD, Figure 3 plots the average hop count against network size varying in the range $[2^{10}, 2^{18}]$. Results for the ideal Chord topology are also shown. All algorithms scale logarithmically with size. Quite interestingly, T-CHORD performs slightly better than Chord. This is explained by the fact that the distance of the longest fingers tend to be larger in our case (due to not strictly satisfying the Chord specification), which speeds up reaching the destination node if it resides in the most distant half of the ring.

Figure 4 plots the average message delay in the same settings. As expected, T-CHORD-PROX outperforms both T-CHORD and Chord, due to its latency-optimized set of fingers. To obtain such performance, T-CHORD-PROX pays a price in terms of latency probes. In this experimental setting, with parameter p set to 5, we have observed a total number of probes per node scaling logarithmically from 45 (for $N = 2^{10}$) to 77 (for $N = 2^{18}$). This is expected, as the number of finger entries that are not empty is $O(\log N)$ [3]. These values are comparable with those reported for other proximity-based protocols like Pastry [1], and can be tuned by varying the p parameter.

Finally, Figure 5 plots the number of cycles needed to obtain the 1-regular lattice (the ring), sufficient to guarantee the consistent routing of messages (absence of message losses) [3], and the l-regular lattice used to provide additional fault-tolerance. In both cases, the convergence is obtained in a logarithmic number of cycles.

5.4. Parameters

To evaluate the impact of the T-MAN message size (m) on the routing performance of our algorithm, we performed the simulations illustrated in Figures 6 and 7. The Figures show that good results are obtained even when using small message size, although it must be noted that in the case of $m = l = 4$, approximately 0.6% of the messages are not delivered to their destination.

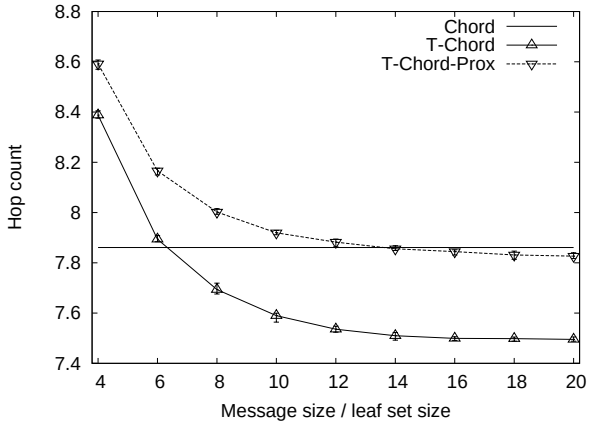

Figure 6. Hop count as a function of message size m and leaf set l

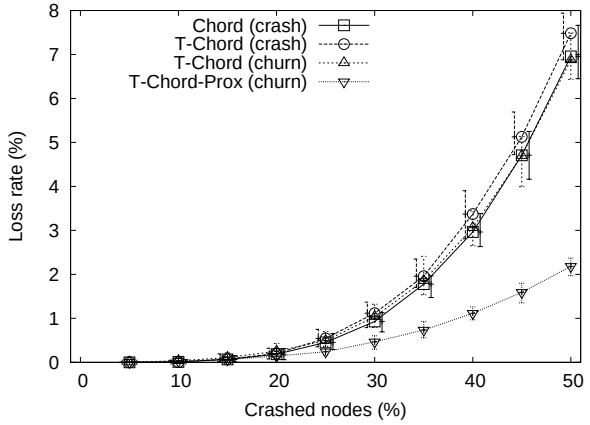

Figure 8. Loss rate under different failure scenarios

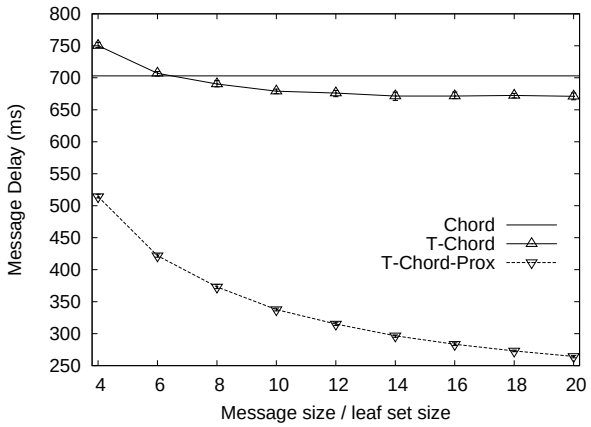

Figure 7. Message delay as a function of message size m and leaf set l

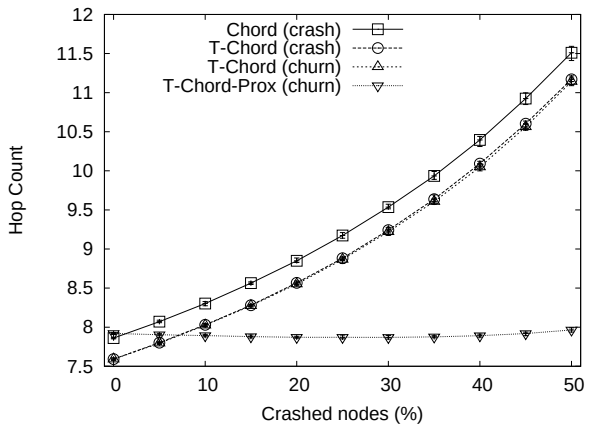

Figure 9. Hop count under different failure scenarios

5.5. Robustness

To test robustness, we have considered two different failure models: *crash* and *churn*. In the former, failures are catastrophic: a given percentage of nodes are suddenly removed from the completed Chord network. In the latter, the same percentage of nodes are removed during the execution of T-CHORD, evenly distributed over time.

The two models play different roles in our analysis. The crash model is the only one applicable to the ideal Chord network that we use for comparison, since we build it offline, without using the actual Chord maintainance protocol. We use this model to obtain a lower bound for routing performance. In the churn model, on the other hand, failures influence the execution of T-MAN; we use this model to show that our algorithm can indeed survive failures during its execution.

It is important to note that a direct comparison between the results of T-CHORD-PROX and the other results is not fair. T-CHORD-PROX probes nodes for latency before inserting them in the finger set, which means that only a few fingers (the ones that fail in the period after the probing) are down when the routing performance is evaluated.

Node joins are not considered: if a node wants to join during the bootstrap phase, it postpones its actual joining until the network is completely formed. At that point, the normal join procedure of Chord can be used.

We have simulated an increasing percentage of nodes removed in a network of size 2^{16}, with T-MAN running for 20 cycles. The results are presented in Figures 8–11. Once again, our routing metrics show that the topology obtained by T-CHORD without proximity is comparable to the ideal

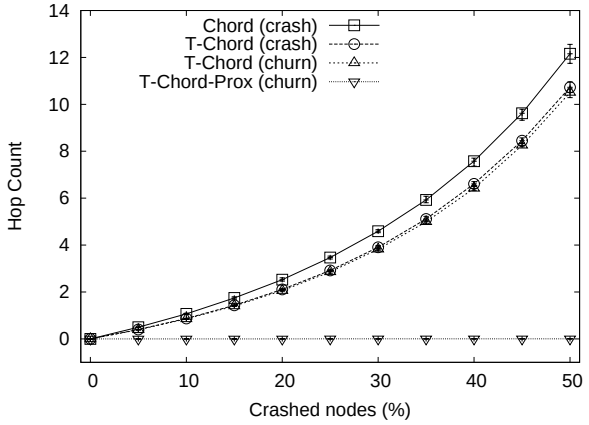

Figure 10. Failed hops under different failure scenarios

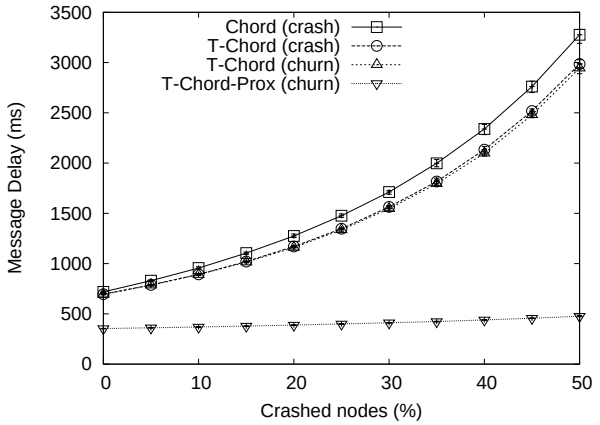

Figure 11. Message delay under different failure scenarios

Chord topology, in both the crash and the churn models.

It is interesting to compare the simulated churn rate with the churn rate observed in deployed P2P networks [17]. In the worst case, the churn rate corresponds to 50% divided by 20 cycles, i.e. 2.5% per cycle. A cycle length of 2 seconds (a perfectly reasonable choice that enables the construction of a 2^{16} topology in less than a minute) corresponds to 0.0125 failures per node per second, two orders of magnitude larger than the rates observed in deployed networks (around 10^{-4} failures per node per second ([17])).

5.6. Practical Considerations

First, the actual execution time of the protocol depends on the length of a cycle, which is a parameter of the protocol. Based on our previous experience with large-scale implementations of gossip-based protocols [**?**], a cycle length of 1-2 seconds is very reasonable. Considering the logarithmic scaling of the execution time, we can conclude that any practical network can be constructed in less then a minute (30 cycles).

Second, in our simulation experiments, we have assumed cycles of the same length at all nodes, and a synchronized protocol start. The first assumption is very weak and is easily satisfied in practical networks, since the drift of clocks is negligable in such a short term. The synchronized start requirement is related to the decision of starting a new structure on demand. In the worst case, this can be accomplished by, for example, a gossip-based broadcast protocol, which requires an additional $O(\log N)$ cycles to be performed.

Finally, while running T-MAN, nodes keep merging the descriptors received in messages to their local view. However, local views do not grow unbounded: as each node progressively finds its position in the ring, the number of *new* nodes contained in messages eventually reaches zero. In our simulation experiments, the average amount of descriptors discovered during the execution ranges from as little as 70 ($N = 2^{10}$) to 140 ($N = 2^{18}$).

6. Related Work

Bootstraping structured overlays is somewhat underemphasized in comparison with other research topics. Existing proposals have assumed networks that are already formed, or networks that grow progressively, using the native join protocol. The discovery of the node to join may be facilitated either by a central (well-known) node, or through a *universal ring*, a shared overlay providing discovery and deployment services [2].

Join protocols enable a new node to find its position inside the structured topology [3, 12, 16]. For example, the single-join protocol of Chord requires a node to perform to locate its position inside the ring, and then to locate each of its $O(\log N)$ distinct fingers [3]. Since both operations require $O(\log N)$ hops (messages), the cost of a single-join is $O(\log^2 N)$.

This aggressive protocol is superseded by a light-weight one that can support concurrent joins. In this case, nodes just find their position in the ring (with a $O(\log N)$ routing operation), while fingers are updated subsequently by a *stabilization protocol*. The protocol is efficient "*... unless a tremendous number of nodes joins the system.*" [3], in which case the updating rate of fingers is not sufficient and routing requires a linear number of hops. In comparison, our approach builds the topology in $O(\log N)$ cycles, with two messages sent and two messages received per node per cycle, with each message being a collection of m 128-bit IDs.

Finally, Voulgaris and van Steen [18] propose an epidemic protocol with a similar goal: jump-starting Pastry. However, their proposal is rather expensive: it requires running $O(\log K)$ instances of a modified NEWSCAST proto-

col [9] in parallel (where K is the size of the ID space), and it does not take latency into account. Besides, it is highly specific to Pastry, whereas our approach, being based on T-MAN, that is able to evolve a wide range of topologies, is potentially more generic. Indeed, we already have preliminary results for building Pastry as well, through an XOR-based ranking function for T-MAN, with costs similar to T-CHORD.

7. Conclusions and Future Work

We have addressed the problem of jump-starting a popular structured overlay, Chord, from scratch. The proposed protocols, T-CHORD and T-CHORD-PROX are scalable, lightweight and robust, and can be applied to scenarios (such as Grids [5] and large-scale testbeds like Planet-Lab [15]), where the overlay infrastructure needs to be built from the ground up as quickly and efficiently as possible.

Although here we targeted Chord, we believe that our approach is more general, and it can be applied to other overlay protocols as well. In fact, we have prelinirary results with Pastry [16]: in this case, two concurrent T-MAN instances are needed, one to build the leaf ring, another to build the prefix-based routing tables. The latter instance uses a specially-designed ranking function, based on the XOR metric.

Our future work will include targeting additional protocols, like Kademlia [12], and the implementation of prototypes for both T-CHORD and T-CHORD-PROX to be tested in Planet-Lab [15] (this activity has already started). Furthermore, we are considering the possibility of using a gossip-based approach not only for bootstrapping, but also for maintaining such topologies in spite of churn.

References

[1] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Proximity Neighbor Selection in Tree-Based Structurd P2P Overlays. Technical Report MSR-TR-2003-52, Microsoft Research, June 2003.

[2] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. One Ring to Rule Them All: Service Discovery and Binding in Structured P2P Overlay Networks. In *Proc. of the 10th SIGOPS European Workshop*, Saint-Emilion, France, Sept. 2002.

[3] F. Dabek et al. Building P2P Systems with Chord, a Distributed Lookup Service. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.

[4] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured P2P Overlays. In *Proc. of the 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, Feb. 2003.

[5] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1999.

[6] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. of SIGCOMM 2003*, pages 381–394, New York, USA, 2003. ACM Press.

[7] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proc. of the SIGCOMM Internet Measurement Workshop (IMW 2002)*, pages 5–18, 2002.

[8] M. Jelasity and O. Babaoglu. T-Man: Gossip-based Overlay Topology Management. In *Proc. of the 3rd Int. Workshop on Engineering Self-Organising Applications (ESOA'05)*, 2005. To appear.

[9] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, 2004.

[10] M. Jelasity and A. Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *Proc. of the 24th Int. Conference on Distributed Computing Systems (ICDCS'04)*, pages 102–109, Tokyo, Japan, Mar. 2004. IEEE Computer Society.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. A Modular Paradigm for Building Self-Organizing P2P Applications. In *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, number 2977 in Lecture Notes in Artificial Intelligence, pages 265–282. Springer-Verlag, Apr. 2004.

[12] P. Maymounkov and D. Mazieres. Kademlia: A P2P Information System Based on the XOR Metric. In *Proc. of the First International Workshop on P2P Systems (IPTPS'01)*, pages 53–65. Springer-Verlag, 2002.

[13] A. Montresor, M. Jelasity, and O. Babaoglu. Robust Aggregation Protocols for Large-Scale Overlay Networks. In *Proc. of the 2004 Int. Conference on Dependable Systems and Networks (DSN'04)*, pages 19–28, Florence, Italy, June 2004. IEEE Computer Society.

[14] PeerSim. http://peersim.sourceforge.net/.

[15] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. of the First ACM Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, Oct. 2002.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale P2P Systems. In *Proc. of the 18th Int. Conf. on Distributed Systems Platforms*, Heidelberg, Germany, Nov. 2001.

[17] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of p2p file sharing systems. In *Proc. of the Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[18] S. Voulgaris and M. van Steen. An Epidemic Protocol for Managing Routing Tables in Very Large P2P Networks. In *Proc. 14th IFIP/IEEE Int. Workshop on Dist. Sys.: Operations and Management, (DSOM 2003)*, number 2867 in LNCS. Springer-Verlag, 2003.