Evaluating the Cost and Robustness of Self-organizing Distributed Hash Tables

Sveta Krasikova, Raziel Gómez, Heverson Ribeiro, Etienne Rivière, Valerio Schiavoni

Evaluating the cost and robustness of self-organizing Distributed Hash Tables

Sveta Krasikova, Raziel C. Gómez, Heverson B. Ribeiro,
Etienne Rivière, and Valerio Schiavoni

Université de Neuchâtel, Switzerland. first.last@unine.ch

Abstract. Self-organizing construction principles are a natural fit for large-scale distributed system in unpredictable deployment environments. These principles allow a system to systematically converge to a global state by means of simple, uncoordinated actions by individual peers. Indexing services based on the distributed hash table (DHT) abstraction have been established as a solid foundation for large-scale distributed applications. For most DHTs, the creation and maintenance of the overlay structure relies on the exploration and update of an already stabilized structure. We evaluate in this paper the practical interest of self-organizing principles, and in particular gossip-based overlay construction protocols, to bootstrap and maintain various DHT implementations. Based on the seminal work on T-Chord, a self-organizing version of Chord using the T-Man overlay construction service, we contribute three additional self-organizing DHTs: T-Pastry, T-Kademlia and T-Kelips. We conduct an experimental evaluation of the cost and performance of each of these designs using a prototype implementation. Our conclusion is that, while providing equivalent performance in a stabilized system, self-organizing DHTs are able to sustain and recover from higher level of churn than their explicitly-created counterparts, and should therefore be considered as a method of choice for deploying robust indexing layers in adverse environments.

1 Introduction

The scale and complexity of distributed systems have dramatically increased in the last decade. Among the abstractions that allow building large-scale distributed services, DHTs (distributed hash tables) play a fundamental role. Applications of DHTs are numerous, ranging from communication backbone [2, 9] to file systems [3] and media streaming [1]. DHTs are at the heart of many industrial large-scale storage systems such as Amazon S3/Dynamo [4].

A DHT is a decentralized index, where each node is responsible for a small disjoint subset of the index. Nodes are organized in some overlay network. The base structure of this overlay allows establishing routing paths towards the node responsible for a given object (based on the *key* of this object, typically a hash of its name or its content). Additional links are added to this base structure in order to provide faster routing, and to exploit redundancy for tolerating crash faults. A plethora of DHTs structures have been proposed. For instance, based on rings [24, 25, 29], based on a prefix-trees [20], based on the Butterfly graphs [18], based on De Bruijn graphs [8]; or using clusters of complete graphs [10].

One of the defining characteristics of large-scale distributed systems is their instability: due to their size, nodes are expected to join and leave regularly, a phenomenon called *churn*. Being able to tolerate high levels of churn is a key requirement for large-scale decentralized systems.

In the classical method to build DHTs, a new node will gradually *navigate* an existing structure to find its neighbors, and inform nodes that should connect to it. Under churn, or when there are concurrent nodes arrivals, this process may lead to an inconsistent state due to conflicts in the concurrent join or repair processes. To avoid this, joins and repairs should ideally be performed as form of transactions, first contacting all nodes involved in the operation, locking them, applying the change, and finally unlocking them to ensure the atomicity of the overlay changes. A more widespread method is to have all nodes join in sequence, but this means that constructing a network requires a time that is linear in the number of nodes. It is also unclear how this process of sequential joins can be coordinated without a centralized authority. Both methods are costly, error-prone and non-scalable. The reason why these issues have received little attention in the literature (with the exception of [11]) is the overuse of simulations for DHT evaluations, where simplified models removed the need for atomic structure updates.

Gossip-based overlay construction has been proposed as an alternative to the explicit construction and maintenance of overlay networks. When designed properly, gossip-based algorithms can indeed provide self-stabilization guarantees [5]. This means that they are able to recover from any chaotic or incorrect state, as a self-stabilizing system is constantly converging to one of its correct states. Gossip-based protocols are based on simple pairwise interactions between nodes. Nodes typically only have a limited view of the whole system; this view is nevertheless sufficient to take local decisions and implement global behaviour without requiring complex multi-node operations. Two similar frameworks were proposed for gossip-based overlay construction: Vicinity [26] and T-Man [12]. They offer generic construction services allowing users to specify how nodes select their neighbors in the target structure. By converging gradually to the perfect set of neighbors on each node, Vicinity and T-Man create global structures autonomously. The use of a Peer Sampling Service [14] is required to provide convergence guarantees. Gossip-based overlay construction protocols allow bootstrapping a new structure from any state, such as a previously existing structure or even random connections [11].

In their seminal work, the authors of T-Man [12] present the gossip-based construction of Chord [25] as an application of their framework. The resulting system, named T-Chord, is evaluated through cycle-based simulations without considering the impact of churn. We are interested in this work in evaluating the cost, performance and robustness of gossip-constructed DHTs deployed in adverse environments. In addition to T-Chord, we contribute three novel gossip-constructed DHTs using the T-Man framework, namely T-Pastry (following [24]), T-Kademlia (following [20]) and T-Kelips (following [10]). We deployed all four DHTs in a cluster of up to 600 independent nodes and evaluated the cost, performance and robustness of the resulting DHTs structures including when subject to various level of churn. Our findings are the following: the overlays constructed by gossip provide

similar performance to the explicitly-constructed ones, or better performance, when there is no churn. They impose a permanent but steady bandwidth consumption at each node (typically within 2 to 10 KB/s), but are able to (1) perform better under churn and (2) recover to a stable structure faster than DHTs using regular, explicit construction. We also make general observations on the convergence of DHT links using gossip, which can vary widely in performance depending on the nature of the created graph and the mean by which each node selects its neighbors.

The rest of the paper is organised as follows. We review related work in Section 2. We present the principles of T-Man in Section 3, and its use for constructing the four DHT structures in Section 4. We detail our experimental results in Section 5, before concluding in Section 6.

2 Related work

We discuss related work on the evaluation of DHTs and gossip-based protocols as deployed systems. We are not aware of any work evaluating the performance of gossip-constructed DHTs deployed in the field.[1]

The study of regular DHTs deployed over the internet has sparkled significant interest in the last decade. The authors of [27] analyze the lookup performances of Kad, a Kademlia-inspired DHT [20], to identify optimal configuration sweet-spots for routing efficiency. Several studies reveal that Kademlia presented slowness issues affecting its routing performance [22]. The Vuze/Azureus network has been subject of a profiling analysis [6] to characterize the dynamics of its underlying DHT, in particular in the presence of churn. Similar simulation studies for Tapestry and OneHop exist [17]. Analytical studies can help in modeling the expected performance of DHTs, in stable environments [27] or under churn [28].

Studies of gossip protocols deployed *in the wild* offer concrete evidence of their effectiveness but are not numerous. The *BuddyCast* gossip protocol was implemented and deployed to support social recommendation, peer and content discovery, in the context of the Tribler P2P social network [21]. LayStream [19] stacks multiple gossip-based protocols to provide an efficient and churn-resilient dissemination overlay for online video-streaming. LayStream uses T-Man [12] to construct a spanning tree that considers network costs, but does not require a DHT.

3 Gossip-Based Overlay Construction

This background section describes the gossip-based overlay construction protocol T-Man [12], that we use to bootstrap DHTs in the rest of this paper. The goal of a T-Man instance is to construct for each peer a *view*, which contains contact information for a number *neighbor* peers. Unless otherwise noted, we use instances for which the view size is *bounded* to a maximum of c peers. Neighbors are selected according to a *distance function*. The basic principle of T-Man is to periodically look for the "best" neighbors, among the ones in the current view and candidates obtained by exchanges with other peers. Peers are ranked

[1] The one-hop DHT at the core of S3/Dynamo [4] uses gossip and is deployed, but it targets a stable data center environment with less faults and a complete graph structure.

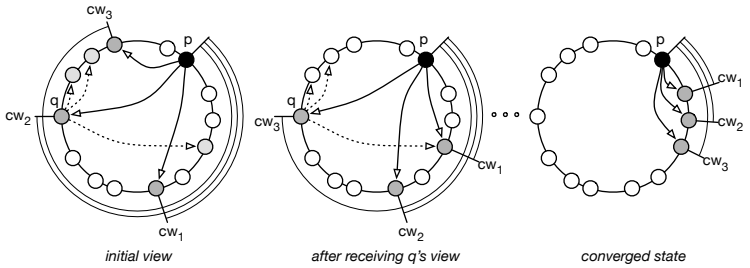


Fig. 1: Convergence of a peer p's view towards clockwise ring neighbors with T-Man.

according to the distance function, and the c closest are kept as neighbors. In case of a tie, other criteria can be used for ranking, or simply a random pick.

In order to guarantee convergence, a T-Man instance must rely on an underlying instance of the Peer Sampling Service (PSS) [14]. The PSS also creates a view of bounded size at each peer, but with the goal that the resulting overlay resembles a random graph as much as possible, guaranteeing a strong resilience to partitions and good recovery properties. This graph is constantly evolving, providing a stream of fresh random peers to each node. The PSS is in charge of inserting new peers in the system and bootstrapping the views for the T-Man instances. The PSS is typically implemented by means of gossip-based interactions itself. The use of the random peers from the PSS allows nodes to avoid ending in a state where the current selection of neighbors is a local minima that is not the global minima, with no possibility to learn about neighbors that belong to the latter [26].

T-Man interactions follow the classical gossip framework. Each peer features two threads. The *active* thread is in charge of initiating exchanges, and the *passive* thread is in charge of answering these requests. The active thread triggers an exchange every Δ time units. It selects the partner either from its T-Man view or from the PSS; more typically, alternating between the two options. The partner selection is normally based on the *age* of the entry, i.e. the time since its inception by the active thread of the corresponding node during an exchange. *Older* links are tested first in order to get rid of links to failed peers within a bounded time. The initiator selects from its view the set of entries it wishes to share with the partner (typically, all of them if entries are not associated with heavy payloads as will be the case in the remainder of this paper). After the exchange, on both sides peers p and q consider the union of their current view and the received entries to form a new view in the SELECTTOSEND function. Entries are sorted according to the specific distance function d associated with this T-Man instance, and on both sides the c first entries are kept as the new view.

We illustrate the process of overlay construction using T-Man with a simple example of a ring in Figure 1. This example will be the basis for the two of the DHT constructions described in the next pages, T-Chord and T-Pastry. Each peer has a identifier id, drawn from a large circular space (e.g., $[0..2^m-1]$). We define distance cw as the *clockwise* distance on the ring, i.e., $cw(p.id,q.id) = (2^m + q.id - p.id) \bmod 2^m$. In this example, the size of the view is $c=3$, therefore the goal of peer p is to converge towards its three *successors*

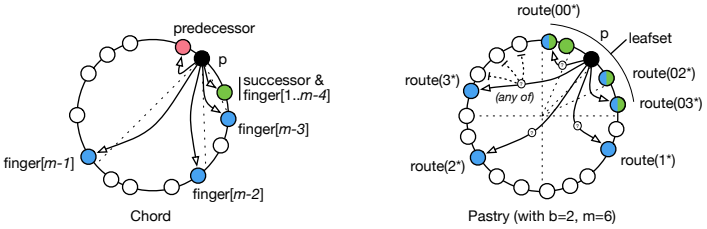


Fig. 2: Ring-based DHTs: Chord [25] and Pastry [24].

on the ring. We see that, by interacting with peer q, p learns about peers that minimize distance cw and eventually converges to the three closest peers.

4 Four Gossip-Based Self-Organizing DHTs

We present in this section the four DHTs used in our evaluation. In the interest of space, we focus our description on the overlay structure aspect, that is built using gossip. Other aspects, such as the routing process, data storage and replication, use the mechanisms presented in the original papers [10,20,24,25]. For each DHT, we provide a list of T-Man instances (Table 1) used to construct the different sets of links each node must possess to form the global structure. We will first present all of the instances as operating independently (as illustrated on the right hand side of Figure 3) and revisit this by presenting runtime optimisations in Subsection 4.5. There exists two types of links in each structure. *Mandatory* links are required to ensure correct routing towards the peer responsible for a given key. *Optional* links are not required for correctness but allow to speed up routing, or provide redundant links for robustness.

4.1 T-Chord

We start by describing the self-organizing version of Chord [25], T-Chord, originally proposed in [12]. The structure is shown in Figure 2 (left). A node p is assigned a bit identifier $p.id$ in $m=128$ bits, determining its position on a ring. The first view links to the set of mandatory *successors*. It uses the cw distance function defined in the previous section. While the original Chord algorithm uses a single successor neighbor, using a set of $c=3$ successors allows routing around failures and implementing replication. A single-neighbor view links to the *predecessor* of the node. The distance function is the counterclockwise distance ccw, where $ccw(p.id,q.id) = (2^m + p.id - q.id) \bmod 2^m$. Node p is responsible for the range of keys following its predecessor and including its own key. The predecessor is used in the original Chord to implement reparation procedures. In T-Chord, the constant convergence towards the optimal peers does not require the presence of the predecessor. Maintaining it allows to compare against the original structure and speed up the convergence of the predecessor's successor list by initiating exchanges that would otherwise only happen through random selection via the PSS. Finally, a set of long-range neighbors, or *fingers*, allows efficient routing on the ring, which otherwise would take $O(N)$ steps, N being the number of nodes. Each finger must point to the node that is the immediate successor to the finger *destination*, covering increasing portions of the ring. Each finger f_i's destination is

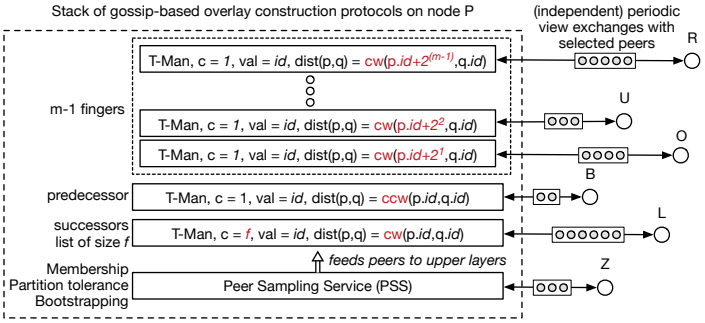


Fig. 3: T-Chord: stack of $m+1$ T-Man instances and the PSS on peer P.

at a double distance on the ring from the destination of the previous finger f_{i-1}: the first finger points to $p.id+2^1$ and is effectively the first successor, while the last finger points halfway across the ring to $p.id+2^{m-1}$. The distance function is therefore cw, with the distance computed from this destination rather than from $p.id$. Fingers allow $O(\log N)$ routing costs, but are not mandatory for routing correctness. We note that there is conceptually one instance of T-Man *per finger*, leading to $1+1+m-1=m+1$ T-Man instances in addition to the mandatory PSS, as shown in Figure 3. We explain in Section 4.5 how the exchanges for these multiple instances can be grouped. Furthermore, in practice only instances for longer fingers need to be enabled, i.e., instances that target a destination that is before the farthest successor can be disabled as they would be duplicates of the successors.

4.2 T-Pastry

T-Pastry creates the structure of Pastry [24] by gossip using a set of T-Man instances. The structure of Pastry as seen from a node p is shown in Figure 2 (right). Similarly to Chord, node p has an identifier $p.id$ in $m=128$ bits. This identifier is represented as digits in base 2^b. Figure 2 uses $b=2$ but our implementations

View	Distance function	Size	Criticality
T-Chord ($m+1$ instances)			
successors	cw (from $p.id$)	3	critical
predecessor	ccw (from $p.id$)	1	optional
$m-1$ inst.: f_i, $i \in [0..m-1]$	cw (from $p.id+2^i$)	1 each	optional
T-Pastry ($2+(m.2^b)/b$ instances)			
leafset (inc)	cw (from $p.id$)	8	critical
leafset (dec)	ccw (from $p.id$)	8	critical
$(m.2^b)/b$ instances: RT_π, $\pi \in q.id.\mathtt{pre}(\{0..m/b\}) \times \{0..2^b-2\})$	pd (from each prefix π_i of $p.id$, plus one non-common digit)	1 each	optional
T-Kademlia (m instances)			
bucket B_i with $i \in [0..m-1]$	xor (from each prefix π_i of $p.id$, plus one non-common bit)	3 each: crit. (1^{st}), opt. (others)	
T-Kelips ($G=\sqrt{N}$ instances)			
p's group $G_{p.id\%G}$	sg (from $p.id\%G$)	∞	critical
$G-1$ instances: other groups G_i with $i \in [0..G-1] \setminus (p.id\%G)$	sg_i (from i)	3 each: crit. (1^{st}), opt. (others)	

Table 1: T-MAN instances (inst.) in T-Chord, T-Pastry, T-Kademlia and T-Kelips.

of T-Pastry and Pastry in Section 5 uses $b=4$. Each node maintains two sets of immediate neighbors on the ring, in both directions. These mandatory links form the *leafset*. Two instances of T-Man are required, using previously-defined distance functions *cw* and *ccw* and a view size of $c=8$ for each. In addition, each node maintains a *routing table* (RT). The goal of the RT is to link to nodes that allow matching a prefix with a target key k that is longer than the common prefix between $p.id$ and k. The RT allows $O(\log N)$ routing hops. For instance, in Figure 2, p with $p.id$ starting with `01*` links to nodes that allows matching `02*`, `2*`, `3*`, etc. The RT is organized as a set of $\frac{m}{b}$ rows, one per increasing prefix length, and 2^b columns, one per digit value. A key difference with T-Chord is that multiple nodes can be selected for a given RT entry, e.g., any node in the upper left quartile of the ring is adequate for the entry `3*`. This property allows selecting any of the eligible nodes for this RT entry, in particular based on network proximity. We do not make use of this possibility in our implementation, as our cluster testbed does not have heterogeneous links. Each of the RT entry can be (conceptually) filled by one T-Man instance for which the distance is set as *pd* (binary prefix distance) to a target prefix π_i of $p.id$, of $i \in [0..\frac{m}{b}]$ digits: $pd(\pi_i,q.id)=0$ if $\pi_i = q.id.\texttt{pre}(i)$), where $q.id.\texttt{pre}(i)$ is the prefix of $q.id$ of i digits, and ∞ otherwise. The total number of T-Man instances for the RT is $(m.2^b)/b$. While this number seems to indicate a large amount of independent T-Man instances running concurrently, this is merely a conceptual description. We explain in Subsection 4.5 how instances are actually grouped and merged by the runtime.

4.3 T-Kademlia

Kademlia [20] is a DHT design allowing prefix routing based on the XOR metric. Nodes have identifiers in $m=160$ bits and the structure as seen by a node p is depicted in Figure 4 (left). It can be modeled as a binary tree, where for each prefix π of $p.id$, a *bucket* represents the set of nodes with a prefix that differs in the $|\pi|+1^{\text{th}}$ bit. For instance, peer p with $p.id=$`1011` and $m=4$ considers 4 buckets, containing nodes with prefixes `0`, `11`, `100`, `1010`, respectively. The binary tree is not materialized–instead, the goal of each node is to find at least one representative neighbor in each bucket, if such a node exists. T-Kademlia uses m T-Man instances, one per bucket. Each instance B_i with $i \in [0..m-1]$ is parametrized with a prefix π_i of $p.id$, with $|\pi_i|=i-1$, to which one bit is added, which is the complement of the i^{th} bit of $p.id$. The distance function used for each instance B_i is *xor*, where $xor(\pi_i,q.id)=\pi_i \oplus q.id$. The *first* link to a bucket is considered mandatory, as routing requires being able to resolve any prefix towards a destination key. Extra optional links are kept (2 in our implementation, hence a view size of $c=3$ links) in order to tolerate faults.

4.4 T-Kelips

Our last DHT is T-Kelips, which emerges the structure of Kelips [10] by using a total of $G=\sqrt{\tilde{N}}$ instances of T-Man, $\tilde{N}$ being an estimation of the number of nodes provided when bootstrapping the DHT. Such an estimate can be obtained using a gossip aggregation protocol [13] or using the interval density approach described in [15] on the PSS's view. The structure of Kelips is simple and allows

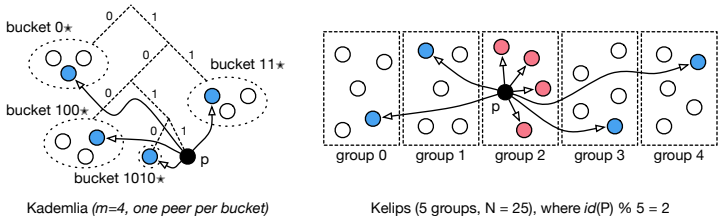
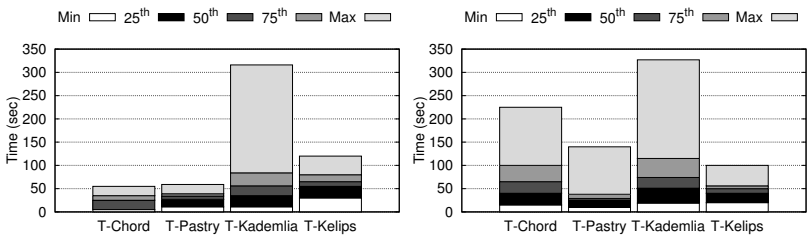


Fig. 4: Kademlia [20] and Kelips [10] DHTs.

O(1) routing at the cost of O(G) connections, more than for previous designs. The set of nodes is split into G groups. Each node must know all members of its group, as well as at least one representative in each of the other groups. Routing requires two steps: locating the appropriate group for target key k, as k mod G, and locating the node responsible for k in the group, this time using advertised responsibility ranges, as done in the O(1) DHT at the heart of S3/Dynamo [4]. The original Kelips construction mechanisms uses gossip principles for node detection but without the clear guarantees for convergence allowed by the use of a PSS and T-Man instances stack. We formalize T-Kelips as the combination of two views. The first view contains all members of p's group. The distance function is *same group*, or sg, where $sg(p.id,q.id)=0$ if $p.id$ mod $G=q.id$ mod G, and ∞ otherwise. Unlike previously mentioned T-Man instances, the size of this view is unbounded, and all links therein are mandatory. The remaining $G-1$ other instances contain each links to one group, excepting the one $p's$ belong to. Each instance $i \in [0..G-1]\backslash(p.id \bmod G)$ is equipped with distance sg_i, defined as $sg(i,q.id)=0$ if $i=q.id$ mod G, and ∞ otherwise. The view at each instance is bounded to $c=3$. We consider the first link to be mandatory and the remaining two to be optional.

4.5 Optimisations and Interactions with the PSS

So far we considered multiple, independent instances of T-Man interacting with the other corresponding instances on other nodes, as illustrated for T-Chord in Figure 3. As previously detailed, all instances rely on the existence of the PSS in order to guarantee convergence. This interaction happens in two forms. First, gossip partners can be selected from the PSS. We alternate between a selection in the T-Man view and the PSS view. Second, the nodes in the PSS view are automatically considered for inclusion in all the views of the locally running T-Man instances. The use of links provided by the PSS is crucial for some of the views, such as the RT entries in T-Pastry, where there is no transitivity in the node selection: either a node has a correct suffix or it does not, and it is not possible to gradually *navigate* towards closer and closer neighbors as shown in the example with distance *cw* in Figure 1. Exchanges for multiple T-Man instances are also grouped: an interaction between two nodes automatically exchanges views for *all* T-Man instances shared by the two nodes, and the gossip cycle Δ is the same for all instances. The implementation allows however defining each instance separately–the grouping is handled automatically by the T-Man runtime. This allows a simple and principled definition of the gossip stack while maximizing convergence speed, and is made possible thanks to the small payload (a single m bits identifier) associated with each of the entries.



(a) Mandatory links. (b) Optional links.

Fig. 5: Convergence of links in gossip-based DHTs.

5 Evaluation

We present in this section the evaluation of T-Chord, T-Kademlia, T-Pastry and T-Kelips using a prototype implementation. For Pastry, we include comparison figures against an implementation following the explicit construction and maintenance mechanisms of the original paper [24]. For space reasons, we leave the comparison of the other explicitly-constructed DHTs to future work.

We leverage the Splay framework [16] for our experiments. All protocols and the T-Man runtime are coded in Lua. We use a cluster of 12 dual-core Intel Core2 machines (24 cores in total), each with 2 GB of RAM and interconnected using a switched 1 Gbps network. We deploy up to 600 nodes over this cluster. The PSS we use is an implementation of the framework in [14]. The view size is $c=10$. Two parameters S and H allow setting the compromise between the randomness quality of the PSS overlay (S) and the ability to quickly discard failed peers (H). We use $S=c/2-1=4$ and $H=1$. The gossiping period Δ for both the PSS and T-Man instances is set to 5 seconds. We consider crash failures only, and leave the evaluation of the impact of other failures models such as omissions and fail-recovery failures, to future work.

We study the convergence of the four structures in Section 5.1 and their routing efficiency in Section 5.2. The network costs for constructing the four DHTs using gossip are evaluated in Section 5.3. Finally, we conclude in Section 5.4 by observing the behaviour of the protocols under churn and comparing against the explicitly-constructed Pastry implementation.

5.1 Overlay Convergence in a Stable Network

We start by analyzing the speed of convergence towards the target structure of the four DHTs. For each node, after each T-Man exchange, the set of neighbors in each of the views is compared to what would be selected by an omniscient observer. Mandatory and optional links, as defined in Table 1, are considered separately. This omniscient selection would pick the c peers with the smaller distance (e.g., for T-Chord's *successors* and *fingers*, or for T-Pastry's *leafsets*) or, when multiple peers are equally *close* to the node, consider valid any random set of c peers among the ones at this distance (e.g., for T-Pastry's *RT entries* of T-Kademlia's *buckets*).

Figure 5 presents convergence results for mandatory (5.a) and optional (5.b) links. We use a representation based on stacked *percentiles* throughout this section. The white bar at the bottom represents the minimum value, the pale grey bar on

top the maximal one. Intermediate shades of grey represent the 25th, 50th–the median–, and 75th percentiles. The median time for a T-Pastry node to learn all of its mandatory links is 5 cycles (25 seconds). For both T-Chord and T-Pastry, all nodes reach mandatory links convergence within 12 cycles (60 seconds). For T-Kelips, it takes twice as much time, which is explained by the higher number of mandatory links ($O(\sqrt{N})$ entries, $\approx$50 average in practice). T-Kademlia has the slowest convergence of mandatory links (up to 65 cycles, or 5.4 minutes in the worst case). Other experiments (not shown) show that this worst-case time increases linearly with the network size, leading to poor scalability. It is interesting to understand the reason for this slow convergence. The distance function used, *xor*, is discrete and transitive: if a node n_1 is close to node n_2, who is in turn close to node n_3, then there is a higher probability that n_1 be close to n_3, than it would be with a random node. This property typically allows speeding up the overlay convergence, as nodes *gradually* learn about closer and closer peers. However, the nature of the overlay built by T-Kademlia prevent this gradual selection from happening. Indeed, a peer in a bucket B_i of peer p contains neighbors starting with a prefix π_i whose last (i^{th}) bit differs from the i^{th} bit of $p.id$. These neighbors may themselves have neighbors that would minimize p's *xor* distance, but they are selected among all eligible neighbors and in a single bucket (a B_j for which π_j has the same $i-1$ bits as π_i and the same i^{th} bit as $p.id$). A result is that the probability for these peers to be of interest to p is the same as the one for peers drawn randomly from the PSS, explaining that the convergence performance is similar to what would happen by considering random peers only, and its duration evolves linearly in the number of nodes. Optional links in T-Kademlia are selected similarly to the mandatory ones, and their convergence shows similar performance. The same similar behaviour for the two types of links is displayed by T-Kelips, and for the same reason. Optional links differ in their definition from mandatory ones for both T-Chord and T-Pastry. In both cases, the latter are more numerous but can accommodate a lazy construction. Their convergence is indeed slower–while our evaluation of the percentage of correct links (not shown) shows that nodes converge quickly to a high number of correct links on average, the time to get *all* correct links, shown in Figure 5 depends on the few last missing links to be selected and increases the overall convergence time for the top of the distribution. It is also important to highlight the impact of the gossiping cycle on the convergence time. Shorter periods result in faster convergence. However, the bandwidth consumption will increase. Some solutions propose to dynamically adapt the gossip period [7].

The gossip partner selection strategy (from T-Man, PSS, or alternating between the two) impacts the convergence speed. Figure 6 presents a sample evaluation of this factor for the T-Chord DHT. We observe that the partner selection has a different effect depending on the optionality of links. For mandatory links, formed of the *successors* view, selecting partners from the PSS only is the fastest, leading to convergence in less than 8 cycles (40 seconds). Selecting partners from the view leads to up to 65 cycles (5.4 minutes) for convergence, with a linear distribution. The *combined* (alternating between the two) strategy has good general case convergence but also incurs outliers converging in up to 26

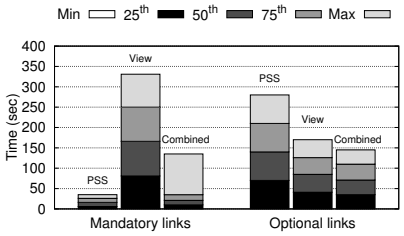


Fig. 6: Partner selection in T-Chord.

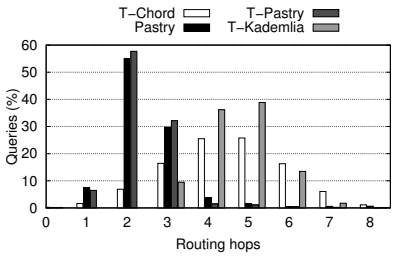


Fig. 7: Overlays routing efficiency.

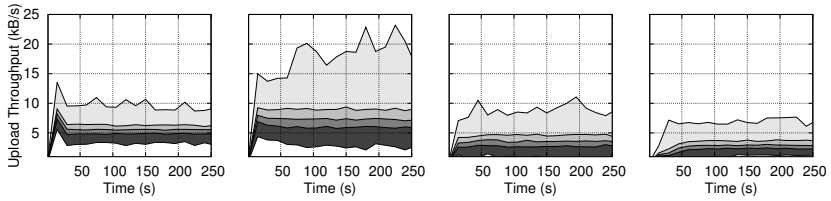
cycles (2.2 minutes). This behavior can be explained based on the nature of the successors lists. A node p who has node q selected as its current successor can ask for q's successors, but it is unlikely that those will be closer to p than they are to q, and will replace the current successors of p. It is only when p is connected to its very close neighbors that these will help to converge by sharing their views and in particular their predecessor. Random interactions are more efficient in this case for fast convergence. Interestingly, the opposite is true for optional links (fingers): random interactions through the PSS yield the slowest convergence, while using the combined approach is slightly faster than using the views only. For T-Kad we observe that selecting partners from the PSS only is the slowest option among the strategies, leading to a median convergence time of 24 cycles (2 minutes), while selecting from the view yields 9 cycles (45 seconds) and using the combined strategies requires 11 cycles (55 seconds). We observe the same lack of pattern for T-Pastry and T-Kelips but in the interest of space we do not present those results. This shows that there does not seem to be a one-size fits all for this parameter; we leave the automatic adaptation of gossip-based overlay construction protocol for future work and use the combined selection in the remaining of our evaluation.

5.2 Routing efficiency

Our next benchmark evaluates the efficiency of routing in the converged DHTs structures. We evaluate the routing costs by sending 50 queries per node to randomly generated keys for a total of 30,000 queries. Figure 7 shows the distribution of the number of hops in each of the structures, with the exception of T-Kelips where this number is always 1 (in the same group) or 2 (general case) by construction. We see that the routing performance is on par with the expectations, in particular, T-Pastry can deliver queries as efficiently as Pastry: 50% of the queries reach the destination in at most 2 hops, while outperforming T-Kademlia and T-Chord, where 50% of the queries require 4 hops.

5.3 Bandwidth Consumption

We evaluate the network bandwidth cost of the gossip-based construction of DHTs. This aspect has been one of the criticized aspects of gossip-based protocols, in particular for dissemination. With the constant and regular exchanges of information, even in a stabilized network, gossip-based protocols indeed incur a constant network cost for these exchanges. We do not consider techniques for auto-adapting the gossip cycle Δ such as [7] but present in Figure 8 the bandwidth

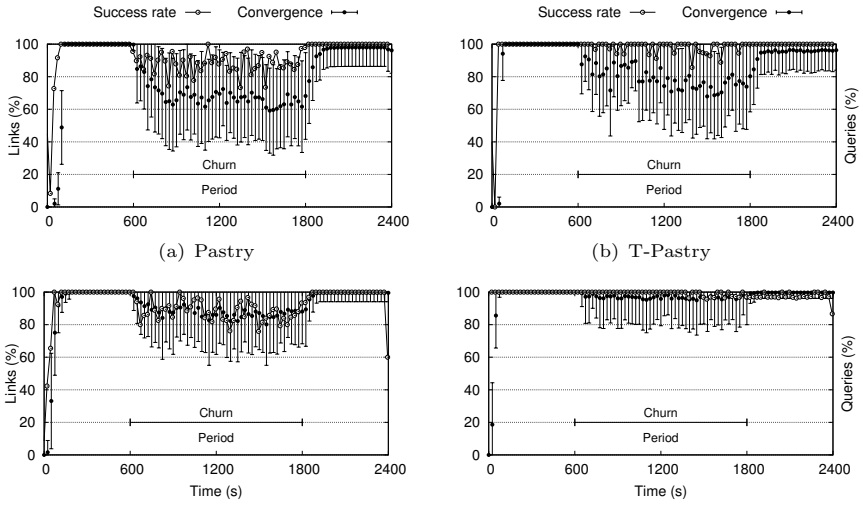


(a) T-Chord (b) T-Pastry (c) T-Kademlia (d) T-Kelips

Fig. 8: Upload throughput in a static scenario for the four DHTs.

consumed by the four DHT, as the distribution of upload costs per node, as a function of time. As expected, this distribution remains stable over time; with a small spread that is for the most part due to our fixed-time range aggregation. For all four DHTs, the bandwidth requirement is around 3 to 10 KB/s, up to 20 KB/s for T-Pastry. We believe these numbers remain reasonable in a wired environment; we leave the implementation of bandwidth capping mechanisms, in particular at the level of the aggregation of T-Man instances by the library, as future work (one solution being to send to a partner only the peers that have a potential interest for its own views rather than the whole set).

5.4 Behaviour under Churn



(a) Pastry (b) T-Pastry

(c) T-Chord (d) T-Kademlia

Fig. 9: Mandatory links convergence and lookup success under churn.

Figure 9 presents the evolution of the overlay structures (convergence of the mandatory links, compared to an offline trace of online nodes in the system) and the success rates for lookups, when the system is subject to churn. We compare the behavior of Pastry to three of the gossip-based DHTs. We leverage Splay's

ability to replay churn traces and orchestrating the creation and failure of peers. We use same churn trace for all four systems: the system starts in a stable state with 150 nodes. During the churn period of 20 minutes, indicated by the arrow on the figures and starting at time 10 minutes, every minute 15% of the nodes are *replaced*: one node goes down and another one goes up. New nodes are fresh nodes and we leave the consideration of recovering nodes to future work. This results in the arrival of 450 new nodes and the departure of the same amount of nodes in the duration of the churn period. The churn period is followed by a stable period of 10 minutes. We observe that T-Kademlia is the least affected by churn: the lookup success rate is close to 100% and the deviation from the correct state is insignificant. Once the system stabilizes, it immediately converges to the correct state again. The high success rate during churn is a result of the presence of redundant links in the routing table. T-Chord, likewise, converges to the ideal state right after the end of the churn period. Its lookup correctness is however more impaired compared to T-Kademlia during the churn. The performance of regular Pastry is more impaired by churn than that of T-Pastry, as the structure diverges more from the ideal and the lookup success rate is more impacted.

6 Conclusion

We have presented in this paper the design and evaluation of DHTs using gossip-based overlay construction principles. Following T-Chord based on T-Man [12], we presented the design of T-Pastry, T-Kademlia and T-Kelips. We evaluated the four DHTs using the Splay framework on up to 600 nodes, and under churn. The results indicate that gossip-based overlay construction is a sound approach for large and dynamic system: while it requires a constant bandwidth consumption, it is able to bootstrap overlays in very short times without concerns about the structure correctness or degradation. For one representative system compared against its explicitly-constructed counterpart (Pastry), the overlay resulting from the gossip-based construction shows better ability to handle and recover from churn.

Acknowledgments. The research leading to these results has received funding from CHIST-ERA under project DIONASYS, and from the Swiss National Science Foundation (SNSF) under grant 155249.

References

1. Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., and Singh, A. SplitStream: high-bandwidth multicast in cooperative environments. In *SOSP 2003*.
2. Castro, M., Druschel, P., Kermarrec, A.-M., and Rowstron, A. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC 2002*.
3. Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with CFS. In *SOSP 2001*.
4. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's highly available key-value store. In *SOSP 2007*.
5. Dolev, S. *Self-stabilization*. MIT Press, 2000.

6. Falkner, J., Piatek, M., John, J. P., Krishnamurthy, A., and Anderson, T. Profiling a million user dht. In *ACM IMC 2007*.
7. Felber, P., Kermarrec, A.-M., Leonini, L., Rivière, E., and Voulgaris, S. Pulp: an adaptive gossip-based dissemination protocol for multi-source message streams. *Springer PPNA 5*, 1 (feb 2012), 74–91.
8. Fraigniaud, P., and Gauron, P. D2B: A De Bruijn based content-addressable network. *Theorical Computer Science* (2006).
9. Gupta, A., Sahin, O. D., Agrawal, D., and Abbadi, A. E. Meghdoot: content-based publish/subscribe over p2p networks. In *Middleware 2004*.
10. Gupta, I., Birman, K., Linga, P., Demers, A., and Van Renesse, R. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *IEEE P2P 2003*.
11. Jelasity, M., Montresor, A., and Babaoglu, O. The bootstrapping service. In *ICDCSW 2006*.
12. Jelasity, M., Montresor, A., and Babaoglu, O. T-Man: Gossip-based fast overlay topology construction. *Computer Networks 2009*.
13. Jelasity, M., Montresor, A., and Babaoglu, O. Gossip-based aggregation in large dynamic networks. *ACM TOCS 23*, 3 (2005), 219–252.
14. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., and Van Steen, M. Gossip-based peer sampling. *ACM TOCS 2007*.
15. Kostoulas, D., Psaltoulis, D., Gupta, I., Birman, K. P., and Demers, A. J. Active and passive techniques for group size estimation in large-scale and dynamic distributed systems. *Journal of Systems and Software 80*, 10 (2007), 1639–1658.
16. Leonini, L., Rivière, E., and Felber, P. SPLAY: Distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze). In *NSDI 2009*.
17. Li, J., Stribling, J., Morris, R., Kaashoek, F., and Gil, T. M. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. In *INFOCOM 2005*.
18. Malkhi, D., Naor, M., and Ratajczak, D. Viceroy: A scalable and dynamic emulation of the butterfly. In *ACM PODC 2002*.
19. Matos, M., Schiavoni, V., Rivière, E., Felber, P., and Oliveira, R. LayStream: composing standard gossip protocols for live video streaming. In *IEEE P2P 2014*.
20. Maymounkov, P., and Mazières, D. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS 2002*.
21. Pouwelse, J. A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D. H., Reinders, M., Van Steen, M. R., Sips, H. J., et al. Tribler: A social-based peer-to-peer system. *Conc. and Comp.: Pract. and Exp.* (2008).
22. Rhea, S., Chun, B.-G., Kubiatowicz, J., and Shenker, S. Fixing the embarrassing slowness of OpenDHT on planetlab. In *WORLDS 2005*.
23. Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J. Handling churn in a DHT. In *USENIX ATC 2004*.
24. Rowstron, A., and Druschel, P. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Middleware 2001*.
25. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM ToN 2003*.
26. Voulgaris, S., and van Steen, M. Vicinity: A pinch of randomness brings out the structure. In *Middleware 2013*.
27. Wang, C.-C., and Harfoush, K. On the stability-scalability tradeoff of DHT deployment. In *IEEE INFOCOM 2007*.
28. Wu, D., Tian, Y., and Ng, K.-W. Analytical study on improving DHT lookup performance under churn. In *IEEE P2P 2006*.
29. Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., and Kubiatowicz, J. D. Tapestry: A resilient global-scale overlay for service deployment. *IEEE JSAC 2004*.