Self-organized fault-tolerant routing in peer-to-peer overlays

Wojciech Galuba, Karl Aberer
Ecole Polytechnique Fédérale de Lausanne (EPFL)
firstname.lastname@epfl.ch

Zoran Despotovic, Wolfgang Kellerer
DOCOMO Euro-Labs, Munich, Germany
lastname@docomolab-euro.com

***Abstract*—In sufficiently large heterogeneous overlays message loss and delays are likely to occur. This has a significant impact on overlay routing, especially on longer paths. The existing solutions to this problem rely on message redundancy to mask the loss and delays. This incurs a significant bandwidth cost.**

We propose the Forward Feedback Protocol (FFP) which only routes a single copy of the message and detects the message loss and excessive delays while routing. Failures are signalled along the routing paths. Based only on the simple binary signals, each overlay node locally and independently learns to route to avoid failures. The local node interactions lead to the emergence of fast reliable overlay routes. This is a continuous process, the system constantly self-organizes in response to changing delay and loss conditions.

We evaluate the protocol in the Internet deployment and in simulation. Our system uses 2-5 times less bandwidth than the existing overlay routing approaches that rely on high message redundancy for fault-tolerance. Despite its marginal bandwidth investment in reliability, FFP achieves up to a 30% higher delivery success rate in comparison to the existing solutions. The protocol is scalable with local state size of $O(\log^2 N)$ in terms of the network size and is universally applicable to all recursively routing overlays.

I. Introduction

To ensure scalability of message delivery in peer-to-peer (P2P) systems, the nodes are typically interconnected to form an overlay network. The messages are then routed to their destinations hop-by-hop along the links of the overlay topology. Peer-to-peer systems are mostly deployed in heterogeneous environments with resource availability varying not only across the nodes but also over time. If any of the computational, storage or network resources are exhausted, message loss and delays might occur. Persistent message loss, usually caused by peers' departure from the network, is trivial to detect and detection typically triggers the appropriate protocols that repair the P2P overlay topology [24], [2], [27], [23] after the failure.

However, any sufficiently large P2P system will also experience non-persistent, intermittent message loss due to various factors, including peer overload [21], transient network connectivity problems [14] or DDoS attacks [12]. These problems cannot be solved by using the same techniques as persistent failures, since permanent changes in overlay topology incur substantial additional repair costs such as the overlay search for new neighbors or additional key replication in distributed hash tables [22].

Transient failures pose a challenge. When message loss occurs at some peers or network links, even with low probability, it may substantially decrease the overlay routing reliability, especially for longer routing paths. This can be demonstrated using a simple Bernoulli trial argument as in [12], [8]. Moreover, some peers might introduce message forwarding delays. The delays accumulate over the whole path and the delivery time may exceed application timeouts.

A simple local solution to the problem could use hop-by-hop acknowledgments. After some peer receives the message it sends an acknowledgement back to the previous hop indicating that it has processed the message successfully. By keeping track of the acknowledgements the peer can eliminate the faulty peers from the routing paths. However, a receipt of an acknowledgement does not guarantee that its sender has already received the acknowledgment from the next hop on the path. The acknowledgment thus signals only that the message performed one hop and not that it successfully traversed the whole path and reached the destination. All the nodes on the path must be functioning correctly to complete the message delivery. The loss probability and delays are cumulative over the whole routing path and such failures are challenging to detect by relying only on local ack signalling. Path-wide mechanisms are necessary to achieve this.

In practice, all the widely used solutions to the problem of reliable message delivery rely on redundancy for masking the message loss and delays. In one approach [8], messages are sent along several disjoint routing paths. If one path fails, the messages on the other paths can still potentially reach the destination. Another approach [19] uses iterative routing that keeps several parallel RPCs (remote procedure calls), each independently asking peers on the routing path for the next hops that are closer to the destination. When one RPC fails the other can still make progress. These solutions heavily rely on message redundancy and thus have a high bandwidth cost. Moreover, they do not in any way remember where the failures happen and are likely to repeat the same routing mistakes again.

In this paper we propose an overlay routing protocol that routes each message only along a single routing path for bandwidth-efficiency, detects message loss along the paths and over time learns to avoid the lossy and slow peers in an entirely decentralized and self-organized way. In our Forward Feedback Protocol (FFP) each routed message is followed on its routing path by a binary feedback message (§III-B). The feedback is positive when the message is delivered on time and negative when routing path delays exceed the application timeout or when messages are lost. Based on the feedback, each node on the path locally and independently learns to avoid slow or lossy parts of the overlay. Despite the independence of its nodes, the overlay as a whole converges on reliable loop-free routes. The process of learning from feedback is continuous, the system constantly self-organizes in response to changing delay and loss conditions.

The proposed FFP protocol combines several properties in a novel way:

- **path-wide fault-tolerance** - FFP's failure detection covers the whole routing process: from the moment the source sends the message, until the final destination acknowledges the receipt. Thanks to that FFP can respond to failures that can only be detected at the routing path level, such as when the delay accumulated over the whole path exceeds the application timeouts.
- **self-organized routing** - FFP peers do not require any prior knowledge about their neighbors, including their location in the overlay address space (§III-C). Routing is topology-oblivious. Through feedback each peer individually learns which of its neighbors is a reliable forwarder for which destination. The network as a whole converges on efficient routing paths (§V-C) in a decentralized self-organized way.
- **low overhead** - peers in our system continuously learn through feedback and remember past failures. This is in contrast to some of the existing approaches which rely on multiple redundant paths for increasing fault-tolerance and do not learn from routing mistakes. In §V we demonstrate that despite the 2-5 times lower bandwidth usage our solution achieves the success rate comparable to the existing approaches.
- **scalability** - FFP is fully decentralized and scalable, we show the local state size to be only $O(\log^2(N))$ in terms of the network size (§IV).
- **universality** - FFP is simple and general enough to be used as a fault-tolerance mechanism in any recursively routing overlay or in any recursively routing network (§VI).

The remainder of the paper is organized as follows. In §II we go over the existing work. Section III describes the basic system setup followed by the specification of the Forward Feedback Protocol (§III-B). We then analyze the protocol's scalability and propose several optimizations (§IV). FFP is then evaluated (§V) together with the existing approaches under a range of failure scenarios and workloads and the paper concludes in §VII.

II. Related work

Considerable attention has been devoted to overlay fault tolerance to churn, i.e. constant peer arrivals and departures. Many designs have been proposed [24], [20], [2], [27], [23], [22] each with different topologies and overlay maintenance algorithms. The problem of message loss is solved either by retries or by falling back on multipath or iterative routing, which we explain next.

Although our solution is based on recursive routing, many of the widely deployed systems such as Coral [13] or Kademlia [19] rely solely on iterative routing. Iterative routing puts the control over the routing process in the hands of the source of the message and works as follows. A source S picks the neighbor A closest to the destination and sends a *next hop request* to A asking for A's neighbor that is closest to the destination. Suppose that A responds with B, then S in turn sends the next hop request to B asking for B's neighbor closest to the destination and so on until the destination is reached. Because of the round trips between the source and the nodes on the routing path, iterative routing increases the latency, it is on average up to 60% higher than recursive routing according to some measurements [10] and since more messages are exchanged bandwidth consumption is also higher [18]. However, by centralizing the control over the progression of routing, iterative routing is more fault tolerant, especially in its parallelized form (e.g. in Kademlia [19]) when the source S instead of sending the request for next hops to only one of its neighbors, it sends the request to k neighbors and at any time maintains k outstanding next hop requests to different peers until the destination is reached. We contrast the performance of our approach with parallelized iterative routing in §V.

Another solution to message loss in overlays is multipath routing. Messages are routed along many, possibly node-disjoint paths to lower the loss probability and lower the delay [5], [4], [8]. In our system, failures are actively detected and routed around using only a single path. In §V we compare the performance of our system with the routing path diversification approach from [8].

The concepts of trust and reputation have been used to detect and then isolate the faulty or selfish peers in non-cooperative environments [16], [3], [26]. Most of this work considers the problem of selecting a reliable peer to perform a transaction with, e.g. a file transfer. Up to our knowledge there has been no direct application of the trust and reputation concepts to fault-tolerant overlay routing, so there are no grounds for comparison with our approach.

The stigmergic routing protocols are inspired by ant colonies and their ability to form robust short paths between the nest and the food sources. In AntNet [11], routing is divided into two phases. The forward ant first attempts to reach the destination. Once the destination is reached a backward ant returns on the same path as the forward ant depositing pheromone at each node which tells the subsequent forward ants in which direction is the destination. Our protocol uses a similar design pattern. When the routing path is traversed, it is either reinforced or weakened depending on weather the delivery to the destination was successful or not. However, unlike AntNet, which is designed for fixed networks, our protocol specifically addresses the dynamic nature and larger scales of the peer-to-peer overlays.

An early evaluation of the Forward Feedback Protocol (§III-B) used in this paper has been performed in [15]. In the current work we replace the complex learning algorithm with a more efficient and robust success estimation technique (§III-C). We apply the Forward Feedback Protocol to solve a specific fault-tolerance problem and for the first time the protocol's effectiveness is demonstrated in an Internet deployment.

III. The protocol

A. Service provisioning

We assume an overlay that offers some *service* that is addressable in the peer ID space. For example in the distributed hash tables (DHTs), key-value pairs are stored and retrieved by specifying the location of the key in the peer ID space. When a peer (the source) requests a service it sends a *service request* to some destination in the peer ID space. We assume that even though the service request specifies a unique location in the ID space, the requested service can be provided by more than one peer (e.g. through key replication in DHTs). The service request is routed hop-by-hop until it reaches the service provider (i.e. the destination).

The source, after it has sent the service request, expects to receive a *service response*. In a correctly functioning overlay the response always arrives. Even in the case when the source asks for a non-existent service, a negative response is sent back to the source. If the response does not arrive within a timeout, this indicates an overlay routing failure and may happen when either the request is forwarded too slowly or is entirely lost. We next define a protocol that informs all the peers on the routing path that a routing failure has occurred. By using that failure information, request forwarders can improve their routing decisions and route around faulty peers.

Our protocol is entirely agnostic to the details of the service response. Service provisioning may be just a single message from the destination back to the source or it may be a complex exchange of messages involving the source, the destination and possibly other peers. The only assumptions we make are (i) that the source can determine the success or failure of service provisioning and (ii) that any service provisioning completes within the T_s timeout. Applications are expected to adjust T_s to their needs.

We define the *lookup* to be the whole process of issuing and routing the service request until the service response is received.

B. Forward Feedback Protocol (FFP)

The *Forward Feedback Protocol* (FFP) is to disseminate the routing failure information to all the peers on the routing path. There are two outcomes possible when routing a service request: either success, when the source receives the service response or failure when the source does not receive the response within the timeout T_s. The source is the only peer that can determine the outcome of the routing process. When the outcome is known the source sends a *feedback message* that is recursively routed along exactly the same routing path as that taken by the service request (Fig. 1). The feedback message is either *positive* or *negative* depending on whether

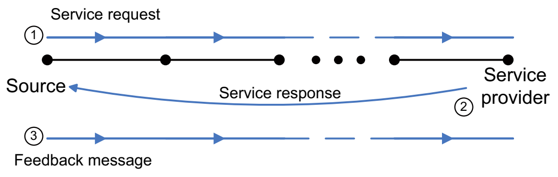

Fig. 1. **The Forward Feedback Protocol (FFP).** The service request is recursively routed in the overlay until it reaches the service provider (the destination). The destination sends the response back to the source. When the service response is successfully received or the T_s timeout fires the source sends a feedback message along the same routing path that the service request took. After receiving the feedback, the peers on the routing path learn whether their routing decisions led to the successful lookup.

the outcome was a success or a failure. As the peers along the routing path receive the feedback they learn whether their routing decisions led to the successful service provisioning.

Each peer on the routing path other than the source or destination after forwarding the service request expects to receive a feedback message from the previous hop. All service requests and their corresponding feedback messages contain the same unique identifier. When a service request is received a node allocates the state containing the unique identifier from the incoming the service request and the address of the sender. This state is used to match the incoming feedback messages with the previously forwarded service requests and to forward the feedback backward on the path. The state is kept at a node i until either: (1) the feedback arrives at i or (2) the service response arrives at i and i is the source or (3) T_s fires at i and i is the source or (4) T_f fires at i and i is not the source. The $T_f > T_s$ is a timeout specifying how long a non-source node waits for arrival of the feedback message.

C. Fault-tolerant routing

Each peer locally keeps a set of *success estimators*. The success estimators reflect the history of the past lookup outcomes. When a service request arrives and needs to be forwarded the peer selects the next hop for which the success estimate is the highest. The arriving feedback messages are used to update the success estimators. As the peer is forwarding service requests and receiving feedback it improves its routing decisions.

Success estimators. Assume that some peer i has n neighbors to which it can send service requests. Let the identifier of i be p_{ID}. The neighbors are indexed by $j = 1 \dots n$. The peer i keeps one success estimator $\hat{\theta}_{j,d_z(d_{ID})}$ for every (neighbor, destination zone) pair. The *destination zone* $d_z(d_{ID})$ is the function of the destination ID d_{ID} defined as $\lfloor -\log_2 d(p_{ID}, d_{ID}) \rfloor$, where $d(p_{ID}, d_{ID})$ is the distance in the ID space between p_{ID} and d_{ID} specified in the service request. The properties of the zoning function and the rationale behind it are illustrated on Fig. 2.

The success estimator $\hat{\theta}_{j,z}$ kept by node i models the neighbor j's reliability of delivering service requests to zone z. Each $\hat{\theta}_{j,z}$ is represented by two numbers a and b which are the exponential moving average counts of the observed positive (a) and negative (b) feedback. The value of $\hat{\theta}_{j,z}$ is

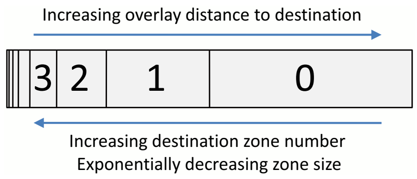
Increasing overlay distance to destination
3
2
1
0
Increasing destination zone number
Exponentially decreasing zone size

Fig. 2. **Destination zoning.** Zoning is key to FFP's scalability. Rather than keeping per-destination success estimators which would not scale we divide the destination space into zones and only keep per-zone success estimators. The further the zone is from the peer the larger the portion of of the destination space it covers. Since the zone sizes decrease exponentially there are $O(\log N)$ of them in terms of the network size, which dramatically reduces the protocol's local state size.

simply defined as $\frac{a}{a+b}$. The a and b counts are initialized to 1. When positive feedback arrives the updates are: $a \leftarrow \gamma a + 1$, $b \leftarrow \gamma b$, on negative feedback: $a \leftarrow \gamma a$, $b \leftarrow \gamma b + 1$.

The parameter $0 < \gamma < 1$ provides a single knob for controlling how sensitive the system is to failures. Low values of γ result in a system that readjusts its routing paths quickly in response to failures but the routes are less stable since even a few lost messages can affect them. In a high γ system more feedback needs to accumulate before the peers change their routing decisions but the decisions are more reliable.

Next hop selection. Given a service request r with a destination d_{ID}, the node i selects such next hop h that the predicted success probability is maximized, i.e. $h = \arg\max_{j \in 1 \dots n} \hat{\theta}_{j,d_z(d_{ID})}$, where $j \in 1 \dots n$ are all the neighbors of i in the overlay.

While forwarding the request each node decrements the request's TTL (time-to-live) counter. When it reaches zero the request is dropped.

Self-organized fault-tolerance. The next hop selection rule makes the service requests follow the most reliable paths. When the lookup fails all the peers on the routing path decrease their success estimates for their downstream neighbors on the path, the routing path is weakened. In the same way, when a lookup succeeds, the path is positively reinforced and more likely to be used in the future. As lookups are performed, each node locally and independently accumulates reliability information about its overlay neighbors. The network as a whole converges on reliable routes in a completely decentralized self-organized way. This process is continuous and when delay or loss conditions change, the network detects that and reroutes the traffic to increase reliability (Fig. 3).

Topology-oblivious routing. In contrast to existing overlay routing protocols, FFP's next hop selection does not rely on the overlay neighbors' peer identifiers but only on the neighbors' reliability history. In that sense, FFP is entirely oblivious to the topology of the network it routes in. This is key to FFP's universal applicability to a wide range of topologies.

Consider an overlay that has never forwarded any service requests. Initially all success estimators on all peers are $(a = 1, b = 1)$. Due to the properties estimator updates and the next hop selection the service requests either tend to visit the next hops that have not been visited before or

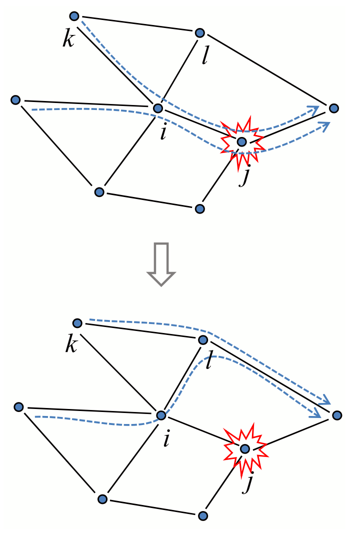

Fig. 3. **Self-organized fault-tolerant routing.** Two routes are passing through peer j. When j starts failing the two routes are weakened by negative feedback. Peer i starts associating failure with its neighbor j and switches to l as the next hop. Similarly, node k routes around i, however, in this k's next hop i is not faulty, it only happens to lay on a faulty route. In general, it does not matter which node on the routing path reacts to failure as long as some node does. Both i and k made their decisions to reroute independently and based only on local observations. This illustrates how an FFP-based network of independent nodes self-organizes and finds reliable routes.

those that have already been successful at delivering to a destination zone. Given enough service request traffic the system eventually by trial and error learns which next hops are best for which destination zones and routing becomes efficient. Despite starting with no knowledge about the next hops at all, this process converges relatively fast (§V-C). Successful routing paths are rapidly reinforced and service request traffic is redirected to use them.

Optimized bootstrap. Initially, when no feedback information is available the peers make essentially random routing decisions. Over time, as more successful paths are found and reinforced with positive feedback the routing becomes reliable for all source-destination pairs. This process can potentially be slow, but only takes place when a large group of uninitialized peers suddenly joins the system, which is a rare case in practice. However, to account for this we introduced a performance optimization: the *warmup phase*. The warmup phase starts after the peer joins and ends when the peer has received at least $f_{min} = 100$ feedback messages. During the warmup phase the peer uses the usual greedy distance-minimizing rule for routing in the overlay. When the warmup phase has finished the peer switches to FFP routing and starts using the feedback gathered during the warmup to make routing decisions. We found the simple $f_{min} = 100$ rule to perform well in all the workloads and failure scenarios used in the evaluation (§V).

Loop-freedom. Most of the existing protocols follow the greedy approach in which each hop attempts to minimize the distance to the destination in the peer ID space. FFP routing is driven by the success estimators and not the ID space distance. To increase the routing reliability the peers in our system deviate from the greedy routing paths. Because of these deviations, one desirable property of greedy routing is lost: loop-freedom. However, routing loops are extremely rare in FFP. There are two reasons for that. First, any loops that occur lead to exhaustion of the service request TTLs, which leads to failures and negative feedback. The negative feedback is then received by the peers that are part of the loop and they start using alternative routes most likely breaking the loop. Second, when loops occur they increase the lookup latency, which leads to the T_s timeouts followed by negative feedback and the breaking of the loop.

Achieving near loop-freedom without explicitly enforcing it is a good illustration of FFP's path-wide fault tolerance in action. It does not matter how the service requests are routed as long as they do not exhaust the TTL and the response arrives at the source within T_s. Any routing inefficiencies such as loops are eliminated by peers changing the routing paths in response to negative feedback.

IV. Scalability, overheads and optimizations

Each peer needs to store the success estimators. Assume a system of N peers and some peer i with d neighbors. The destination zoning function divides the whole set of peers into $O(\log(N))$ zones. Hence, there are $O(d \log N)$ $\hat{\theta}_{j,d_z(d_{ID})}$ success estimators. Assuming $d = O(\log N)$, the total state size for the success estimators is $O(\log^2 N)$, which scales well with the network size. A peer must also keep the state between arrival of the service request and either receipt of the corresponding feedback message or the T_f or T_s timeout. Assuming the rate λ of incoming service requests and $T_s < T_f$, the expected state size is $O(\lambda T_f)$, which in practice does not pose a problem since request rates in P2P systems are low and overlays are typically well load-balanced in terms of forwarded traffic.

The bandwidth overhead in our system consists of the feedback messages and the unique identifier field that needs to be added to the service requests. A feedback message consists of the unique identifier and a Boolean flag indicating either success or failure. The naive approach is to send each feedback message separately, but this can be optimized in a number ways. First, feedback can be piggybacked on other service requests sent to the same neighbor. Second, the largest part of the feedback message is the unique identifier. If the identifiers are assigned sequentially and independently for each ordered pair of connected peers, then feedback can be sent for contiguous ranges of identifiers, which can simply be specified by their left and right bounds. If the ranges are big enough the total amortized bandwidth overhead is reduced to a single bit per feedback message.

V. Evaluation

The system is implemented using the ProtoPeer[1] toolkit and evaluated in a 400-node PlanetLab[2] deployment.

We consider the following routing protocols: (1) **BASE** - the baseline without any routing fault-tolerance mechanisms, (2) **MULTIk** - multipath routing as in [8], in the first hop the source chooses k neighbors closest to the destination instead of one, (3) **ITERk** - iterative routing scheme based on Kademlia [19] with k simultaneous lookups, we set a low timeout of 500ms on the hop-by-hop responses so that peers can recover from local failures and resume routing on-the-fly and (4) **FFP** - system running the FFP.

We use a bidirectional Chord [24] implementation. Each peer sends one service request for some key every 500-1500ms. We create a set of 1024 keys uniformly randomly distributed in the ID space. The service request keys are drawn from this set according to a power-law distribution such that 50% of the requests are for the top 5% of the keys. This constitutes a realistic workload for a large number of P2P applications [6].

The request is considered delivered when it reaches either the peer responsible for the key or one of the two of the peer's immediate Chord successors or predecessors, i.e. in the DHT terms, there is a 5-fold key replication. The timeouts are $T_s = 3s$ and $T_f = 6s$ and the TTL is set to 20. Note that T_f applies only to FFP, while T_s and TTL apply to all four routing protocols. A lookup fails when either the TTL is exhausted or T_s fires before the service response is received by the source. FFP's γ parameter (§III-C) is set to 0.95, which offers a good balance between the sensitivity to failures and the tolerance to feedback noise.

Each deployment is approximately 400 PlanetLab hosts in size depending on the availability. The standard practice in PlaneLab experiments is to exclude highly loaded hosts from the deploy list. We have included all the hosts that we could log on to provide a more challenging environment for evaluating the fault-tolerance. The hosts were polled for their 15-min UNIX load averages. The median was 5.11 and the 90^{th} percentile was 16.12. Under these conditions CPU starvation was causing considerable delays on some peers.

A. Message loss

To simulate peer overload some fraction of peers are droppers, which drop messages with probability 0.5. Enough messages get through such that the neighbors of the droppers do not consider them as departed from the network and keep them as their overlay neighbors. Just as the other peers, the droppers periodically send their own service requests, but they might get dropped on sending as any other message.

In our failure scenario, every 5 minutes a new 10% batch of peers becomes droppers adding to the existing set of droppers. New droppers stop arriving when half of the peers become droppers.

Success rate. We measure the lookup success rate, i.e. the number of service responses that arrived before the T_s timeout as the fraction of all service requests that have been sent by the sources. Figure 4 summarizes the results.

As more droppers arrive the lookup success rate decreases sharply for the system without any routing fault-tolerance mechanism (BASE). The multipath (MULTI4) and iterative (ITER4) routing approaches improve the lookup success rate

[1]http://protopeer.epfl.ch/

[2]http://www.planet-lab.org/

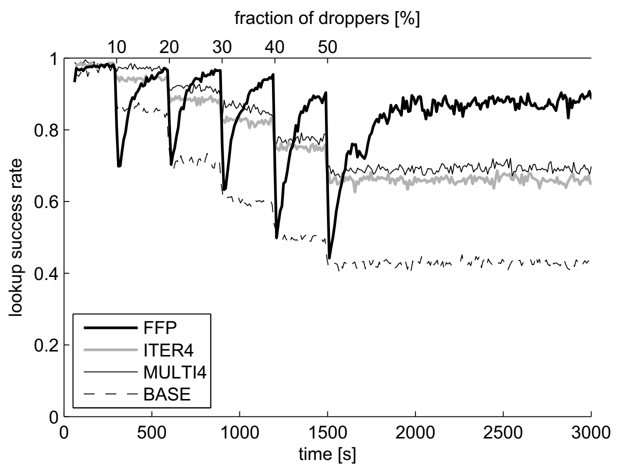

Fig. 4. **Resilience to message dropping.** Every 5mins. a new 10% batch of peers starts to drop all messages except their own lookup traffic. FFP responds to the failures and routes around the droppers, which allows it to reach the delivery success rates up to 30% higher than the existing approaches.

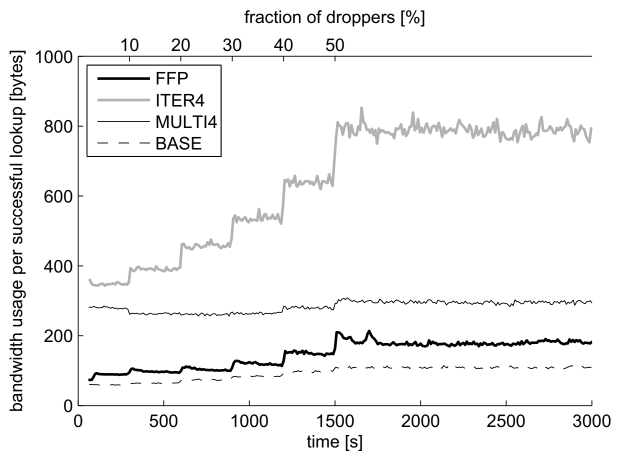

Fig. 5. **Per-lookup bandwidth usage.** We sum up the total send and receive bytes from all the peers. This value is then divided by the number of successful lookups and plotted. MULTI4 routing relies on 4-way message redundancy for fault-tolerance, which results in 4 times the bandwidth cost compared to the baseline. ITER4 has a 1s timeout on the RPCs and when many peers are dropping messages there are many RPC retries which increases the bandwidth cost considerably. FFP's only overhead are the feedback messages (§III-B), which add up to only a marginal bandwidth increase compared to the system without any routing fault-tolerance mechanism (BASE). As the droppers arrive in the system, FFP routes around them, this increases the path lengths and thus the per-lookup bandwidth cost.

by 30%. An FFP-based system, when it converges, is able to achieve a high 90% success rate despite the fact that half of the peers in the system are dropping messages.

The droppers do not send any feedback messages even though they are the sources of service requests. Although there is less feedback exchanged in the network, the available feedback provides the peers with enough information to effectively adjust the overlay routes.

Bandwidth usage. In the dropper experiment we have also measured the total number of bytes that have to be transmitted by the peers per successfully completed lookup (Fig. 5).

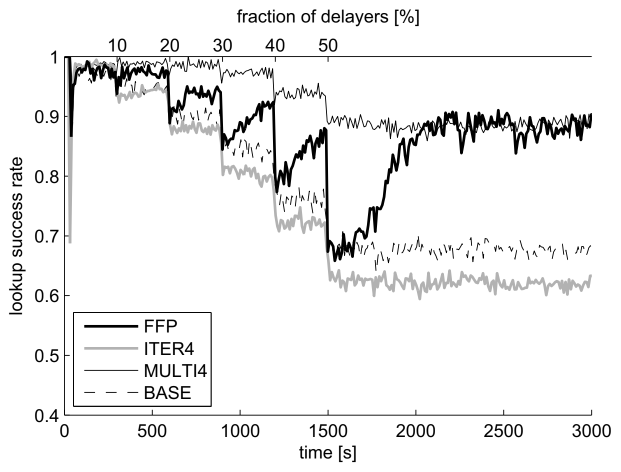

Fig. 6. **Tolerance to message delays.** Every 5mins a new 10% batch of peers starts delaying all outgoing messages by 500-2000ms. The multipath algorithm performs well, while the source-controlled iterative routing is significantly slowed down by the delaying sources. The feedback that FFP peers are receiving is noisy due to the high variability of the path delay. FFP needs 500s to detect and route around the delayers after the last failure injection at 1500s. When it converges, FFP reaches the high performance level of multipath routing.

Both the multipath and the iterative routing approaches rely on redundantly sent messages to tolerate failures, which considerably increases the bandwidth cost. ITER4 is worse than MULTI4 in two ways. First, every time an RPC times out in ITER4 it is retried, which in presence of many droppers results in many retries, which increase the bandwidth cost considerably. Second, because of the round trips back to the source the total lookup latency is much greater than in the case of MULTI4 and many ITER4 lookups take longer than T_s and fail (Fig. 4).

Although we have not investigated this, our implementation can be further optimized by piggybacking the feedback messages on other messages or sending feedback messages in batch (§IV), thus avoiding message header overhead, which is the dominating component in case of the simple feedback messages.

B. Tolerance to message delays

We turn to evaluating the tolerance of our system to peers delaying the messages. Apart from the inherent PlanetLab delays we introduce artificial ones. Similarly to the case of droppers a new 10% batch of peers becomes delayers every 5mins. Each delayer delays all messages by an interval uniformly randomly chosen from between 100 and 2000ms.

Figure 6 shows the lookup success rate. Multipath routing with its four independent routing paths is able to reach the destination at least with one of them. Iterative routing fails entirely in this case. When the sources are also the delayers they significantly slow down the whole iterative routing process. Due to the high delay variability FFP needs 500s to gather enough feedback information to route around the delayers. Once it converges it reaches the same high lookup success rate as the multipath routing.

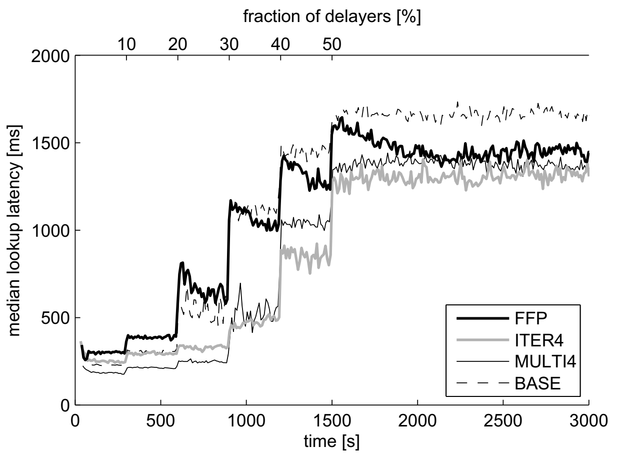

Fig. 7. **Median lookup latency under forwarding delays.** The same setup as in Fig. 6. Multipath routing with its four independent routing paths achieves the lowest latency. Even though FFP was not designed for latency minimization, it removes slow peers from the routing paths when the lookup latency exceeds the T_s threshold, which over time leads to latency decrease.

We have also measured the median lookup latency for the successful lookups (Fig. 7). The multipath routing method besides increasing the fault-tolerance to delays also lowers the lookup latency. Our system is designed to maximize the fraction of lookups completed within a timeout $T_s = 3s$ and not the lookup latency median. However, as a side effect of rerouting for reliability FFP also slightly lowers the median delay.

C. Topology-oblivious routing & the bootstrap

One of the novel properties of our protocol is its ability to route in overlays while using only the reliability feedback and ignoring any overlay topology information (§III-C) such as the locations of the neighbors in the peer ID space. In this section we demonstrate this property experimentally.

All the peers initially have no previously recorded feedback, i.e. all their success estimators are $(a = 1, b = 1)$. At the one minute mark all the peers simultaneously start their workloads. Initially, the routes are explored randomly, there are many failures, but over time each node learns which of its next hops are reliable forwarders for which destinations.

To create a more challenging scenario we use a workload in which peers send service requests to destinations that are at least 0.25 away on the unit Chord ring. Such workload has a larger fraction of long multi-hop routing paths.

We compare the systems with and without warmup. Warmup is an optimization that speeds up convergence by initially relying on topological information and then subsequently switching FFP's purely reliability-based routing (§III-C).

The results on Fig. 8 shows that in a system without warmup the uninitialized FFP peers are still able to converge on efficient routes despite ignoring any topological information. Moreover, the created routes are loop-free, despite the fact that loop-freedom is not explicitly enforced by FFP. Loops lead to failures and the peers in a self-organized way learn to avoid them.

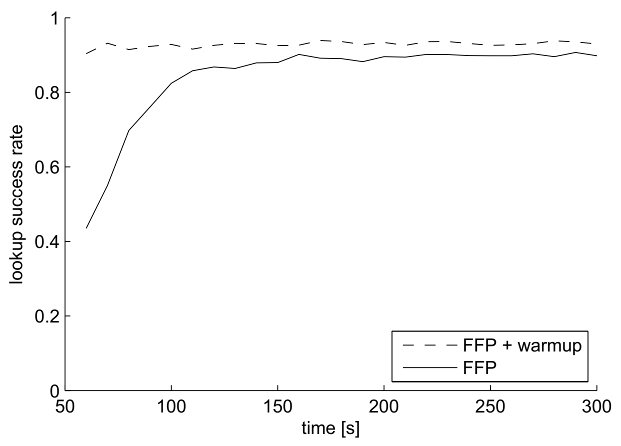

Fig. 8. **Topology-oblivious routing & the bootstrap.** Two instances of FFP are compared, with and without warmup (§III-C). All the peers' success estimators are initially $(a = 1, b = 1)$. At 60s all the peers start their workloads simultaneously. FFP without warmup converges on efficient loop-free routes relying solely on the reliability feedback and ignoring any overlay topology information. The warmup shortens the initial convergence time as well as slightly improves the overall performance.

The measurements also confirm that the warmup process significantly speeds up the initial convergence.

D. The influence of the workload on convergence time

Until now our FFP evaluation has been based on a power-law workload, we next explore how the workload influences the FFP's performance. We are next going to consider four workload variants: multi-source/multi-destination (MS/MD), single-source/multi-destination (SS/MD), multi-source/single-destination (MS/SD) and single-source/single-destination (SS/SD). In the single-destination workloads peers issue lookups for a single fixed key. In the multiple-destination workloads destination keys are chosen uniformly at random from the ID space. In the single-source workloads there is a single high-rate peer sending lookups, the other peers are passive. The workloads are normalized such that the lookup rates summed across all the sources in each workload are the same.

Figure 9 compares how the system recovers from a failure under different workloads. The general observation that can be made is that the fewer the source-destination pairs in the workload the faster the failure recovery. This is particularly evident in the SS/SD case. When traffic is constrained to fewer paths, the peers along these paths receive more feedback and can adapt faster. Moreover, the failure recovery is the faster the fewer there are destinations in the workload. The peers in the neighborhood of the destinations receive more feedback and with it they can determine which of the destination ID replicas are more reliable.

E. Tolerance to churn

So far we have evaluated our system in setups with all peers permanently on-line, however in P2P systems peers are constantly joining and leaving the overlay (churn). Peer

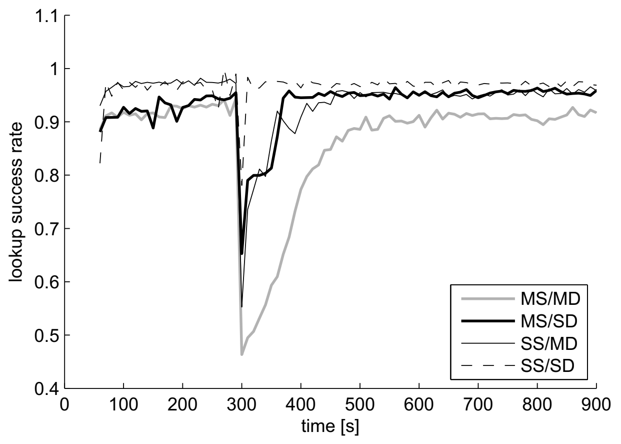

Fig. 9. **FFP performance under different workloads.** The four workloads are: multi-source/multi-destination (MS/MD), single-source/multi-destination (SS/MD), multi-source/single-destination (MS/SD) and single-source/single-destination (SS/SD). At 5mins 25% of the peers become droppers (§V-A). The fewer paths the traffic is confined to the more feedback the peers on these paths receive and the faster the recovery after the failure.

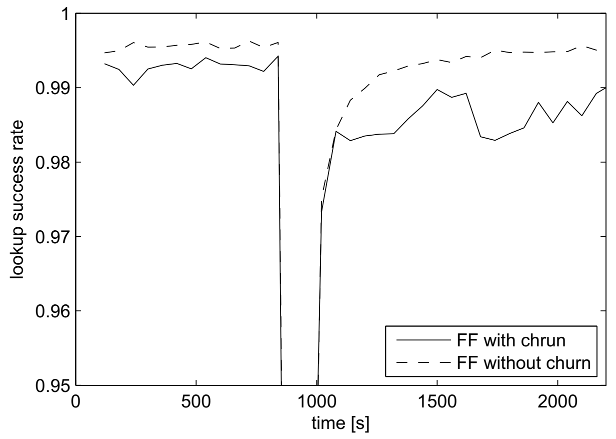

Fig. 10. **FFP performance under churn.** An FFP system is running a multi-source multi-destination workload. At 800s 40% of the 1024 peers become droppers. The time resolution is 60s. Feedback message loss caused by churn and the constant joining of new uninitialized peers do not affect FFP's ability to recover from the failure.

departures in our system cause forwarded messages to be lost and also break the feedback flow, which affects the state of the success estimators, which in turn might disrupt FFP's next hop choices. In addition, churn constantly adds new peers which initially are in the warmup mode (§III-C) and make routing decisions that do not account for the lookup failures. In the following experiment we test to what extent these effects affect the system's performance.

We take the same application code as used in the PlanetLab deployment and evaluate it in the ProtoPeer simulator. The Internet latencies are taken from the King dataset[3]. We are using the churn model based on the Kad4 data from the Stutzbach's et al. study [25], which measured churn characteristics in live P2P networks. There are 1024 live peers on average. At 800s into the simulation, 40% of the peers become droppers. The results of the simulation with and without churn are compared on Fig. 10. The ability to detect and route around failures is largely unaffected by churn. There is only a $1-2\%$ drop in the lookup success rate, but this is mainly caused by the unpreventable message loss due to the peer departures.

VI. Discussion

Self-organized fault-tolerance. In the Forward Feedback Protocol each node locally and independently of the other nodes learns to route based on the simple binary feedback. As we have confirmed in the evaluation, an overlay composed of independent FFP nodes continuously self-organizes to increase the routing reliability. When message loss (§V-A) or excessive delays (§V-B) occur in the network, the paths are rapidly readjusted to route around the faulty peers and overlay links.

Bandwidth efficiency. The performance data accumulated at each peer about its neighbors is key to FFP's reliability. The other approaches do not keep track of their neighbors' performance and instead rely on message redundancy to deal with the failures. As we have confirmed in the measurements this comes at a significant bandwidth cost. Knowing the neighbors' performance allows FFP to choose more reliable paths which allows our protocol to reach higher success rate than the existing approaches.

However, despite the high bandwidth cost, the usage of redundant routing paths in iterative and multipath routing allows for reduction in the median lookup latency. In contrast, FFP is better suited for bandwidth-constrained environments and applications in which latency minimization is not the highest priority.

Convergence time & suitability for high-rate traffic. Convergence time in FFP is directly dependent on the rate at which the peers are forwarding the traffic. The higher the the traffic rate the more feedback is available and the more reliable the routing decisions are. As we observed (§V-D) convergence is significantly faster when the traffic is less diverse, with fewer source-destination pairs . The peers are able to respond to changing conditions faster since the feedback accumulates in a smaller set of success estimators. The fact that the more peers forward the quicker they converge on reliable routes as well as FFP's low bandwidth overhead make this protocol particularly suitable for applications with high-rate traffic.

Although the biggest factor affecting FFP's convergence time is the workload, it is not the only factor. We are currently running a more detailed quantitative study to determine the other factors (success estimator update rules, different destination zoning functions etc.).

Combining the routing approaches. So far we have looked at our solution to fault-tolerant overlay routing as an alternative to the existing approaches. However, an interesting possibility is the integration of our solution with the multipath or iterative routing protocols. Every time the next hop is chosen in one of these protocols, it is chosen greedily to minimize the distance

[3]http://pdos.csail.mit.edu/p2psim/kingdata/

to some destination ID. This greedy next hop choice can be replaced with our failure-aware choice (§III-C) taking the feedback into account. This would combine the fault-tolerance gains from both the message redundancy of iterative and multipath routing and the failure-awareness of our routing approach.

Under abundant bandwidth the system could reap the full benefits of all approaches, when bandwidth became scarce the system could reduce the amount of message redundancy (the k parameter of MULTIk and ITERk) or entirely fall back on routing only with our bandwidth-efficient method.

Feedback suppression & incremental deployment. In FFP a service request is always followed by a feedback message. But does it have to? We have indirectly measured that moderate feedback loss does not prevent FFP from converging on reliable routes (§V-A). This brings up an interesting question: How much of the feedback is necessary for reliable routing? The system could suppress feedback generation to conserve bandwidth. It would be possible in such a system to precisely control the tradeoff between reliability and the amount of bandwidth invested in maintaining reliable routes.

If the system can tolerate feedback dropping then it will tolerate a fraction of peers that do not support the FFP protocol and ignore the feedback messages instead of forwarding them. This opens up the possibility of incremental deployment of FFP in already running overlays, which could proceed as follows. Initially, the isolated FFP peers do not receive any feedback and rely on the traditional overlay routing. As more FFP-compatible peers are deployed and start forming larger connected components in the overlay topology they begin exchanging feedback, soon they accumulate enough feedback to exit the warmup stage and begin routing using their success estimators.

Unstructured vs. structured vs. FFP overlay routing. Unstructured overlays (e.g. [7]) do not assign any identifiers to the peers and do not maintain any global overlay topology. Instead of routing they rely on random walks or flooding for message delivery. Similarly to the unstructured overlays, the FFP peers do not need to know their neighbor's identifiers to be able to route. Also, just as the unstructured overlays, FFP can operate in the topology-oblivious mode in which it finds the destinations using random walks, however unlike the unstructured overlays, FFP learns from its routing mistakes and the random walks are over time replaced with more reliable faster paths.

Even though FFP makes no use of the neighbor identifiers each peer locally uses its own identifier as the input to the zoning function (§III-C). The zoning function makes the assumption that the destination IDs close to one another in the ID space are also likely to be topologically close in the overlay graph. This is a property shared by all structured overlays [24], [2], [27], [23], [20] and allows FFP to scale.

Richer signalling. The proposed feedback mechanism is general enough to be used in applications other than overlay routing fault-tolerance. Feedback messages can be used to signal any Boolean property of the routing path, not only when it satisfies the T_s latency constraint. Forwarders can route around those peers that consistently cause the property of interest to become false. For example, when service requests pass through forwarders or service providers close to reaching their throughput saturation point, these peers can signal it by setting a congestion control bit in the service requests (as in [17]). The source then finds out about the congestion via the service response and sends negative feedback along the path to notify the forwarders that some peers on the path are close to being overloaded and should be avoided.

Feedback can be used to signal not only the Boolean properties of the routing path but others as well. For example, if feedback messages contain the delay information, then the success estimators can be modified to be delay estimators and the system can potentially be turned into a latency-minimizing overlay [1], [9].

Beyond overlay routing. We have applied FFP to solve the problem of fault-tolerance in recursively routing overlays. However, FFP is general enough to be layered on top of any recursively routing network. For example, we have already obtained several promising results from applying FFP to multi-hop wireless networks. In such networks, FFP provides both the routing functionality and the route discovery functionality, which uses FFP's random walks to locate the destinations. In wireless networks we do not need to rely on zoning (§III-C) for scalability, success estimators can simply be stored per-destination. In general, FFP can be adapted to the needs of a specific application or a network by modifying the granularity of the success estimators and assigning some meaning to the Boolean success flag carried by the feedback.

VII. Conclusions

We have presented a solution to the problem of message loss and excessive delays during lookups in structured overlays. We have shown how nodes acting completely independently can self-organize into a reliably routing overlay using only the binary feedback messages as the signalling mechanism. We have confirmed the effectiveness of our solution in an Internet deployment under a variety of failure scenarios and workloads. The FFP protocol has up to 30% higher delivery success rate than the existing approaches while using 2-5 times less bandwidth.

Our protocol can readily be integrated into most of the modern structured overlays to provide fault-tolerance at low bandwidth cost. For additional fault-tolerance the protocol can be combined with either the iterative or multipath approaches. However, fault-tolerance is not the only area of application. FFP can be viewed as a general signalling protocol and can be used for overlay congestion control or lookup latency minimization. FFP can also potentially be adapted for use in other networks such as multi-hop wireless or the Internet.

References

[1] "Meridian: a lightweight network location service without virtual coordinates," in *SIGCOMM'04*. ACM, 2005, pp. 85–96.

[2] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-grid: a self-organizing structured P2P system," *SIGMOD Record*, vol. 32, no. 3, pp. 29–33, 2003.

[3] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in *CIKM*. ACM, 2001, pp. 310–317.

[4] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan, "Best-path vs. multi-path overlay routing," in *IMC '03*. New York, NY, USA: ACM, 2003, pp. 91–100.

[5] M. S. Artigas, P. G. Lopez, and A. F. G. Skarmeta, "A novel methodology for constructing secure multipath overlays," *IEEE Internet Computing*, vol. 09, no. 6, pp. 50–57, 2005.

[6] S. Bianchi, S. Serbu, P. Felber, and P. Kropf, "Adaptive load balancing for DHT lookups," in *ICCCN*, 2006, pp. 411–418.

[7] F. Bustamante and Y. Qiao, "Friendships that last: Peer lifespan and its role in p2p protocols," 2003.

[8] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach, "Secure routing for structured peer-to-peer overlay networks," *ACM Operating Systems Review*, vol. 36, no. si, p. 299, 2002.

[9] F. Dabek, R. Cox, M. F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," in *SIGCOMM'04*. ACM, 2004, pp. 15–26.

[10] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *NSDI'04*. USENIX, 2004, pp. 85–98.

[11] G. Di Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research*, vol. 9, no. 2, pp. 317–365, 1998.

[12] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel, "Denial-of-service resilience in peer-to-peer file sharing systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 38–49, 2005.

[13] M. J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing content publication with coral," in *NSDI'04*. USENIX, 2004, pp. 239–252.

[14] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica, "Non-transitive connectivity and dhts," in *WORLDS'05*. Berkeley, CA, USA: USENIX Association, 2005, pp. 10–10.

[15] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, "Authentication-free fault-tolerant peer-to-peer service provisioning," in *DBISP2P 2007*. Springer, 2007.

[16] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in P2P networks," in *WWW*, 2003, pp. 640–651.

[17] F. Klemm, J.-Y. Le Boudec, D. Kostic, and K. Aberer, "Improving the Throughput of Distributed Hash Tables Using Congestion-Aware Routing," in *IPTPS*, 2007.

[18] Li, Stribling, Gil, Morris, and Kaashoek, "Comparing the performance of distributed hash tables under churn," in *IPTPS, LNCS*, vol. 3, 2004.

[19] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *IPTPS*, ser. LNCS, vol. 2429. Springer, 2002, pp. 53–65.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Computer Communication Review*, vol. 31, Berkeley, CA, USA, 2001, pp. 161–172.

[21] S. Rhea, B. Chun, J. Kubiatowicz, and S. Shenker, "Fixing the embarrassing slowness of opendht on planetlab," 2005.

[22] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *USENIX*. USENIX, 2004, pp. 127–140.

[23] A. Rowstron and P. Druschel, "Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems," in *Middleware'01*, Nov. 2001.

[24] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM'04*, 2001, pp. 149–160.

[25] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *IMC'06*, J. M. Almeida, V. A. F. Almeida, and P. Barford, Eds. ACM, 2006, pp. 189–202.

[26] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Trans. Knowl. Data Eng*, vol. 16, no. 7, pp. 843–857, 2004.

[27] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: a fault-tolerant wide-area application infrastructure," *Computer Communication Review*, vol. 32, no. 1, p. 81, 2002.