Authentication-free fault-tolerant peer-to-peer service provisioning

Wojciech Galuba[1], Karl Aberer[1],
Zoran Despotovic[2], Wolfgang Kellerer[2]

[1] Ecole Polytechnique Fédérale de Lausanne, Switzerland {karl.aberer,wojciech.galuba}@epfl.ch
[2] DoCoMo Communications Laboratories Europe, Munich, Germany
{despotovic,kellerer}@docomolab-euro.com

Abstract. The correct functioning of a peer-to-peer network relies on cooperative behavior of peers as service providers. Current approaches to detection and deterrence of non-cooperative behavior, such as reputation systems, rely on (1) global sharing of observations about service provisioning and (2) global authentication. These two factors severely impair the practical applicability. We propose a novel forward feedback protocol that is completely local and authentication-free and where peers locally and independently learn to avoid non-cooperative peers and routing failures.
We evaluate our system in a variety of failure scenarios. The convergence rates and failure resilience are close to those of a fully centralized reputation system. High churn is tolerated without significant drop in performance. For increased fault tolerance the system takes advantage of the service replicas existing in the network. The proposed protocol is lightweight and can readily be integrated into any architecture where service requests are recursively routed, which includes all modern structured overlays.

1 Introduction

In a peer-to-peer architecture peers provide various services to one another. Services include forwarding a message to some destination, storing a key-value pair in a distributed hash table or transferring a file. In a typical P2P network deployment each peer is owned and fully controlled by a different user. Selfish users may choose to modify the peer software to only request and receive services from other peers and never provide any services back. In this way the peer's resources (e.g. bandwidth) are conserved. Users may also be adversaries attempting to subvert the network by generating superfluous or malformed traffic or providing fake services. Independently of the failures caused by the user, a peer may also undergo an internal failure leading to a disruption in the message passing protocols and consequently to service provisioning failures. In this paper we focus only on protocol failures themselves regardless of their causes, be it selfish users, adversaries or chance. We propose an architecture that detects failures in a service provisioning P2P network and learns how to avoid them to maximize the fraction of successfully provided services and prevent service requests from reaching the faulty service providers.

A number of approaches to protecting the P2P networks against service provisioning failures rely on the universal authentication of the peers, i.e. the ability to reliably verify the identity of any peer by any other peer in the network. For example in reputation-based approaches [1, 2] each peer has a reputation associated with its identity and that identity has to be reliably verified by other peers. The typical distributed authentication systems based on PKI are centralized and have a single point of trust (e.g. the root certification authority) which limits the scaling and usability. On the other hand, any decentralized P2P-based implementation of peer authentication, such as PGP or other web of trust approaches, must deal with peers that fail to provide the authentication service to other peers. Tolerating this type of failures is precisely the problem we are solving, a circular dependency. In our approach we do not require any node authentication at all. When a connection between two nodes is opened none of the neighbors have to prove their identity. Each node maintains connections to a set of neighbors and exchanges feedback on the quality of service provisioning only with them. The node learns which of its neighbors are faulty by listening to feedback from other neighbors. No node identities are ever communicated over the network and thus never have to be verified. The crucial difference between ours and the reputation-based approaches is that when there is a service provisioning failure, in the reputation system the service provider is blamed for it, while in our system it is the neighbor who forwarded the service request that is blamed. We show in Section 4 that this simple strategy results in performance that closely trails that of a centralized reputation system with global peer authentication.

In most of the existing approaches a peer that is detected as faulty is isolated from the network, regardless of the extent or the type of the failure. Peers may be only partially faulty. For example, in a P2P file sharing network a peer might choose to respond to requests for small files, while dropping requests for large files to conserve bandwidth. Isolating such a peer completely instead of still routing requests for small files to it decreases the overall availability of files in the network. In our approach, the algorithm learns the nature of the partial failures and maximizes the use of the faulty peer's functionality that is still correct. We demonstrate this property in the context of selective dropping of service requests and service replication (Sections 4.5 and 4.6).

Before a service can be provided, the peer that provides the service needs to be located. In existing approaches service location is generally assumed to be a reliable black box, always perfectly locating the service. In practice this frequently made assumption is not valid, the service location as a distributed mechanism itself requires cooperation among the peers and tolerance to failures. For example, in DHTs locating the peer that stores a specific key requires the cooperation of all the peers that forward the key lookup message. If any of these peers drop the lookup message or mutate it, it may never be delivered to the peer responsible for the key. The modern DHT design mainly focuses on fault tolerance to crash-stop failures in which once a peer crashes or departs, no more messages are sent or received to or from it. However, there is typically lack of fault tolerance to selective message dropping or message mutation (Byzantine failures). Hence it is incorrect to assume that a DHT is a reliable black box when using the DHT to store the critical information in reputation-based approaches.

In this paper, instead of assuming the service location mechanism to be a reliable black box, we assume a specific preexisting *service location protocol* (Section 2.2) in which the service request originates at the requesting peer and is recursively routed hop by hop to a service provider that is capable of providing the requested service. The definition of the service location protocol is broad enough to encompass a large number of applications in which there exists a concept of recursive routing, and to which our forward feedback protocol can be applied, e.g. structured and unstructured overlays or ad-hoc networks.

A failure in the recursive service location may lead to the wrong service being provided or not being provided at all. We do not require the service location to be fault tolerant, instead we propose a system that learns how to ensure both (i) the successful delivery of the service requests despite routing failures and (ii) preventing service requests from reaching faulty service providers. The proposed *forward feedback protocol* (Section 2.3) integrates into the existing service location protocol. Based on the gathered feedback, peers learn which routes lead to service provisioning failures and the service location protocol selects alternative routes (Section 4.3). When services are replicated, our protocol is able to learn which replicas are correct and route towards them (Section 4.6).

2 The model

Let the digraph $G = (V, E)$ be the network of connections between the peers, where V is the set of peers and E is the set of peer pairs representing unidirectional connections. Let $outneigh : V \to 2^V$ and $inneigh : V \to 2^V$ be the set of peers pointed by respectively outlinks and inlinks of some $v \in V$. Let S be the set of services and $offeredServices : V \to 2^S$ be the set of services offered by a given peer $v \in V$.

2.1 Architectural overview

Figure 1 depicts the functional dependencies between the components of the system. We take some existing service location protocol (Section 2.2) that locates the requested service in a hop-by-hop manner and augment that protocol with feedback messages reporting the success or failure of service provisioning (Section 2.3). These successes and failures along with any deviations from the protocol are reported to the failure predictor. The failure predictor (Section 2.4) is used when routing service requests to avoid forwarding service requests to next hops that are likely to lead to a failure in providing the service. The failure predictor uses the p-concepts learning algorithm [3] to make its predictions (Section 2.5).

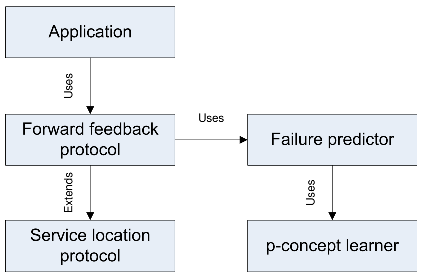


Fig. 1. The proposed system architecture

2.2 Service location

We assume the existence of a service location functionality which recursively routes service requests from the requestor to some provider that offers the service specified in the service request. Services are specified in the service request by means of a service descriptor. A single service descriptor can be matched against not only one but many different services. The set of service descriptors is D and $matchedServices : D \to 2^S$ returns the set of services that the specified service descriptor matches.

We represent service request routing as a function $selectNextHop : S \times V \times 2^V \to V$ that takes the service descriptor of the service that needs to be located, the current node (self) and the set of possible next hops and returns the chosen next hop. The routing paths generated by $selectNextHop$ are assumed to be loop-free. The set of next hops to which a request can be forwarded can easily be constrained on a per request basis by the failure predictor (Section 2.4).

2.3 Forward feedback protocol

When the service request successfully reaches the provider, the provider is expected to provide the service to the requestor. We do not make any assumptions about how the service is provided, it is only assumed that this must happen within a bounded time. After a provisioning failure or timeout, the service requestor initiates the forward feedback protocol described below to inform other peers about the success or failure of service provisioning.

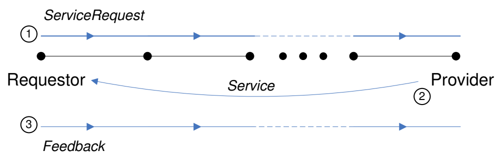


Fig. 2. The operation of the forward feedback protocol

The forward feedback protocol adds on to the existing service request routing protocol. After a service request is forwarded by some node on the routing path it expects to receive feedback from its previous hop. The feedback message contains a single bit describing whether the service was provided successfully or not. The first feedback message is sent by the service requestor after it has received the service. Then on, the service feedback message is forwarded following exactly the same path as the service request has followed until the service provider is reached. This process is depicted on Fig. 2 and Fig. 3 presents the complete algorithm with failure prediction.

From the point of view of the service requestor, when the application calls *requestService* (line 6) for some service descriptor, the peer generates the $ServiceRequest$ message and sends it to the appropriate next hop. It then expects a *serviceProvided* call (line 8) from the application that indicates whether the service has been provided or not. If that call does not happen within the timeout of $T_{service}$ the service provisioning is considered to have failed (line 38) and negative feedback is sent.

From the point of view of the service provider, when it receives a *ServiceRequest* (line 11) with a service descriptor for which it can provide the service, the service is provided to the origin of the service request (line 14).

From the point of view of some forwarding peer v_c, when it receives a service request from some previous hop v_{ph}, it forwards it to the next hop v_{nh} (line 16), v_c then expects feedback from v_{ph} (line 12) on whether the service provisioning was successful or not. If the feedback does not arrive within the timeout $T_{feedback}$ (line 34) the service provisioning is considered to have failed. If the feedback arrives it is forwarded to the same next hop that was taken by the corresponding service request before (line 32), same for the requestor when the service is provided (line 28). This is implemented by assigning a unique identifier to every service request (rid) and remembering the next hops taken in the nh array.

```
1  initialize
2      failurePredictor ← create new failure predictor
3      $nh$ ← empty array
4      $sd$ ← empty array
5
6  function requestService(serviceDescriptor)
7      rid← generateUniqueRequestID()
8      expect call serviceProvided(rid,*) in $T_{service}$
9      forwardServiceRequest(null,rid, self, serviceDescriptor)
10
11 receive ServiceRequest(rid, origin, serviceDescriptor) from src
12     expect Feedback(rid, *) from src in $T_{feedback}$
13     if offeredServices(self) $\cap$ matchedServices(serviceDescriptor) $\neq \emptyset$
       then
14         provide service to origin
15     else
16         forwardServiceRquest(src,rid, origin, target)
17     end
18
19 function forwardServiceRequest(ph,rid, origin, serviceDescriptor)
20     candidateNextHops←outneigh(self) \
       failurePredictor.getSuspectedNextHops(ph,serviceDescriptor)
21     nextHop ← selectNextHop(serviceDescriptor, self,
       candidateNextHops)
22     send ServiceRequest(rid, origin, serviceDescriptor) to nextHop
23     $nh_{rid}$ ← nextHop
24     $sd_{rid}$ ← serviceDescriptor
25
26 function serviceProvided(rid,success)
27     failurePredictor.reportObservation(null, $nh_{rid}$, $sd_{rid}$, success)
28     send Feedback(rid,success) to $nh_{rid}$
29
30 receive Feedback(rid,success) from src
31     failurePredictor.reportObservation(src, $nh_{rid}$, $sd_{rid}$, success)
32     send Feedback(rid,success) to $nh_{rid}$
33
34 timeout on Feedback(rid,*) from src
35     failurePredictor.reportObservation(src, $nh_{rid}$, $sd_{rid}$, false)
36     send Feedback(rid,false) to $nh_{rid}$
37
38 timeout on call serviceProvided(rid,*)
39     failurePredictor.reportObservation(null, $nh_{rid}$, $sd_{rid}$, false)
40     send Feedback(rid,false) to $nh_{rid}$
```

Fig. 3. Forward feedback protocol

2.4 Failure predictor

As the events happen in the protocol they are reported to the failure predictor as observations. An *observation* $o = (ph, nh, sd, b) \in V \times V \times D \times \{0, 1\}$ consists of the previous hop from which the service request was received, next hop to which the request was forwarded, the request's service descriptor and the boolean flag indicating whether the hop led to a successful service provisioning.

Given a set of reported observations, the failure predictor returns the suspected next hops that are likely to cause a failure in service provisioning. The suspected next hops are then excluded from the list of candidate next hops and not taken (line 20). A next hop may be suspected only temporarily, as observations are reported the set of suspected next hops returned by the failure predictor may change. When all next hops are suspected, there is a *next hop null* event (NHnull) and the service request is dropped.

2.5 Learning failures

The failure predictor accumulates observations and based on them makes predictions if a *context triple* of the form $(ph, nh, sd) \in V \times V \times D$ will result in successful service provisioning or not. The context triple describes the previous hop from which the currently forwarded service request has been received, the hypothetical next hop that can be chosen and the service descriptor contained in the service request. The learning task is to learn the probability of success given a context triple and the set of observations.

We employ the learning method called the probabilistic concepts (p-concepts) learning [3], which is based on Rivest's decision lists [4]. A *concept* is a boolean function defined over a domain set X. Rivest [4] has shown that given a set of observations (x, b) where $x \in X$ and $b \in \{0, 1\}$ is the label, an unknown concept can be learned efficiently if the concept belongs to the set kDL, i.e. decision lists with conjunctive clauses of size k. The set kDL is expressive enough to include decision trees and the sets $kDNF$ and $kCNF$, i.e. respectively the disjunctive and conjunctive normal forms with k literals per clause. Kearns et al. [3] extend this result to probabilistic concepts i.e. functions of the form $c : X \to [0, 1]$ where $c(x)$ is interpreted as the probability of observing a positive observation $(x, b = 1)$. Kearns defines the probabilistic decision list c over the set of predicates $\mathcal{F}_n$ to be $(f_1, r_1), \ldots, (f_s, r_s)$, where $f_i \in \mathcal{F}_n$ and $r_i \in [0, 1]$. The value of $c(x)$ is r_j for the smallest j such that $f_j(x)$ is true. The decision list is ω-converging if $|r_i - \omega| \geq |r_{i+1} - \omega|$ for $1 \leq i < s$. Such list is output by the Kearns' algorithm. We use a special case of the list, for $\omega = \frac{1}{2}$, which corresponds to the list with decreasing certainty. The predicates that are most certain (i.e. those with values of r_j closest to 0 or 1) are in the front of the list and are matched against first.

In our case the domain X is the set of all context triples, $X = V \times V \times D$. The set of predicates $\mathcal{F}_n$ has the following properties:

– each predicate is either a single literal or a conjunction between two or more literals
– there are three types of literals: (1) previous hop literals of the form $ph = v_i$, $v_i \in V$ (2) next hop literals $nh = v_i$, $v_i \in V$ and (3) service descriptor literals $sd = d_i$. $d_i \in D$
– a predicate can refer to at most one literal of each type, from this it follows that a predicate can have a maximum of 3 literals

For example, the predicate $ph = v_{12} \wedge sd = d_4$ belongs to $\mathcal{F}_n$ while the predicate $ph = v_{12} \wedge sd = d_4 \wedge ph = v_{15}$ does not.

The set of predicates $\mathcal{F}_n$ is induced from the observation set S before running the Kearns' p-concepts algorithm. For every observation $o = (ph = v_i, nh = v_j, sd = d_k, b) \in S$ there can be a maximum of 7 corresponding predicates generated: (1) $ph = v_i$, (2) $nh = v_j$, (3) $sd = d_k$, (4)$ph = v_i \wedge nh = v_j$, (5) $ph = v_i \wedge sd = d_k$, (6) $nh = v_j \wedge sd = d_k$, (7) $ph = v_i \wedge nh = v_j \wedge sd = d_k$. If either ph or the nh are null the corresponding predicates containing either the previous hop or next hop literals are not generated.

Given the set of observations S and the set of predicates $\mathcal{F}_n$ induced from it the p-concepts algorithm runs and produces the decision list (for the details of the algorithm the reader is referred to [3]). With the set of predicates defined above, thanks to the expressiveness of kDL the failure predictor can learn to predict the probability of success even if it depends on the context triples in obscure, non-trivial ways.

When the function *getSuspectedNextHops* is called with arguments ph and sd then for each possible nh a context triple (ph, nh, sd) is created and the the decision list is consulted to compute the probability of success for that particular context triple. If that probability is lower than the threshold $p_{min} = \frac{1}{2}$ then nh is deemed suspected. When there is no matching predicate found on the decision list, a probability of 1.0 is assumed. In particular this occurs for all context triples when the decision list is empty at the beginning when S has no elements. This *initial optimism* property of the failure predictor is necessary to bootstrap the system when there are no observations available.

The size of the set S of observations does not follow the minimal bounds described by Kearns, as these bounds are impractical. Even if relatively large errors are allowed ($\gamma = 0.1$, $\epsilon = 0.1$, $\delta = 0.1$) the minimum amount of required observations is on the order of 10^6. Instead we use a simple FIFO queue for the observations with the maximum size set to 3000, which is sufficient for good performance.

Recomputing the decision list every time a new observation is reported is costly. We recompute the decision list periodically every n_r-th observation.

3 Scaling & Performance

Despite the locality of the learning algorithm, we need to consider how it scales in terms of network size and number of available services. Our implementation of the learning algorithm has the time complexity of $O(k^2)$ and space complexity $O(k)$, where k is the number of unique context triples that are the input to the algorithm. The upper bound on the number of locally observable context triples is equal to d^2s, where d is the number of neighbors a node has and s is the size of the set of service descriptors, hence the learning time is $O(d^4s^2)$. The node degree d is typically logarithmic in terms of the network size N and the $\log^4 N$ factor still scales well with network size. We are left with s as the main scaling limitation. In practice, rarely can one node observe traffic for all possible destinations, especially in modern structured overlays in which the topologies are balanced and no single node carries most of the traffic. If the problem of a large set of service descriptors exists it can be greatly reduced by dividing the services into groups. Then the learning algorithm uses the reduced set of service groups instead of the services. The division into groups should be done in such a way that the services belonging to the same group fail in the same way (same correlations with previous and next hop). This grouping can be done based on the attributes of the services, e.g. location in some overlay identifier range, service type and QoS etc. The services can also be dynamically categorized into groups by the very same learning process as used for failure prediction. We leave this optimization as future work.

From the protocol performance standpoint, it may be argued that by making the feedback messages compulsory instead of sending only the negative feedback, we are adding an unnecessary overhead. This may be a problem if feedback is used for services that involve e.g. a delivery of a single message to some destination. However, service provisioning in general requires the exchange of a large amount of data between the requestor and the provider and the single bit feedback message is negligibly small in comparison.

4 Evaluation

4.1 Experimental setup

We use [5], an event-driven simulator, the protocols are implemented in their full extent including all types of messages and timeouts. $T_{feedback}$ and $T_{service}$ are both set to 10s.

For service location we use the small-world unit ring topology as in Chord [6]. The peers generate service requests in a Poisson process. The generation rates are identical across all the nodes and equal to one request per second. Each time a service request is generated its service descriptor is selected uniformly randomly from the set of all possible descriptors. This simple workload is sufficient to demonstrate the basic properties of our system and contrast it with other solutions.

We use the latency model based on King [7] measurements, the communication is lossless. For most of the simulations there are no node arrivals or departures (churn). In Section 4.7 we measure and discuss the impact of churn on the performance of our system.

Two different allocations of services to peers are considered: (i) **one-on-one** - each peer provides only one service, each service is provided by only one peer and (ii) **many-on-one** - services are replicated, the number of services is equal to the number of peers, each service is provided by an average of 4 peers (Poisson distributed).

4.2 Convergence rate

In reputation-based approaches the local observations about service provisioning are shared with all other peers in the network. In the forward feedback protocol the observations are only shared with the peers along the routing path. This brings up the question whether such limited observation sharing can ensure fast convergence of the system.

In the following experiment we compare how fast the system converges to the state in which there are no requests sent to faulty service providers. We consider four cases: (1) **LOCAL** - each peer runs an instance of its own simple local reputation system and learns to avoid faulty service providers. After a single faulty service provisioning from some peer A the service is never requested from A again. No reputation information is shared among the nodes. (2) **GLOBAL** - an idealized global reputation system. Each peer A can place one report about another peer $B \neq A$ in the system. The report is a binary flag indicating either a successful or faulty service provisioning. The report can be changed at any time by A. If the majority of reports about some B is faulty then B is deemed faulty by all the peers, otherwise it is deemed correct. The reputation system is centralized in our implementation and relies on global authentication. (3) **FF** - forward feedback - peers use the forward feedback protocol and local learning and (4) **DN** - do nothing - the case in which there is no fault-tolerance mechanism in place.

The service allocation is one-on-one. At the beginning of the simulation all the nodes are correct. At $200s$ 30% of the nodes become DEFECTORs, uniformly randomly. A DEFECTOR is a peer that behaves correctly in all aspects except it always provides a bad service when it is requested. The goal is to minimize the number of service requests reaching the DEFECTORs, to prevent faulty service provisioning. For each of the four cases we measure the fractions of service requests that have reached faulty nodes. Only the service requests sent by correct nodes are taken into account.

The results on Fig. 4 show that when there is no fault-tolerance mechanism in place (DN), then there is a stable failure rate of 30%. This is explained by two facts: (i) the destination service providers are chosen uniformly randomly and (ii) 30% of service providers are faulty. When nodes do not exchange any information about the success of their service requests (LOCAL) then the convergence is very slow, the curve for LOCAL reaches zero only after 6000s. When peers share the information about their observations (FF and GLOBAL) the convergence is significantly faster. It takes 200s for both mechanisms to "forget" the previous good behavior of the DEFECTOR nodes. GLOBAL's transition from trusting DEFECTORs to distrusting is slightly sharper than for FF but both mechanisms eventually prevent all service requests from reaching the DEFECTORs. This result demonstrates that limited feedback sharing in our forward feedback protocol is sufficient to have the same convergence rate as the centralized reputation system in which each feedback report is immediately available to all other peers.

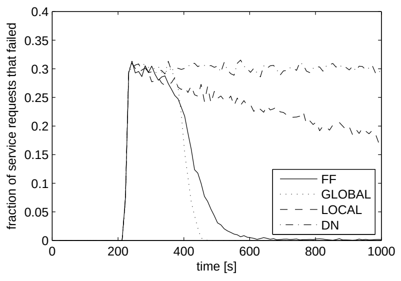


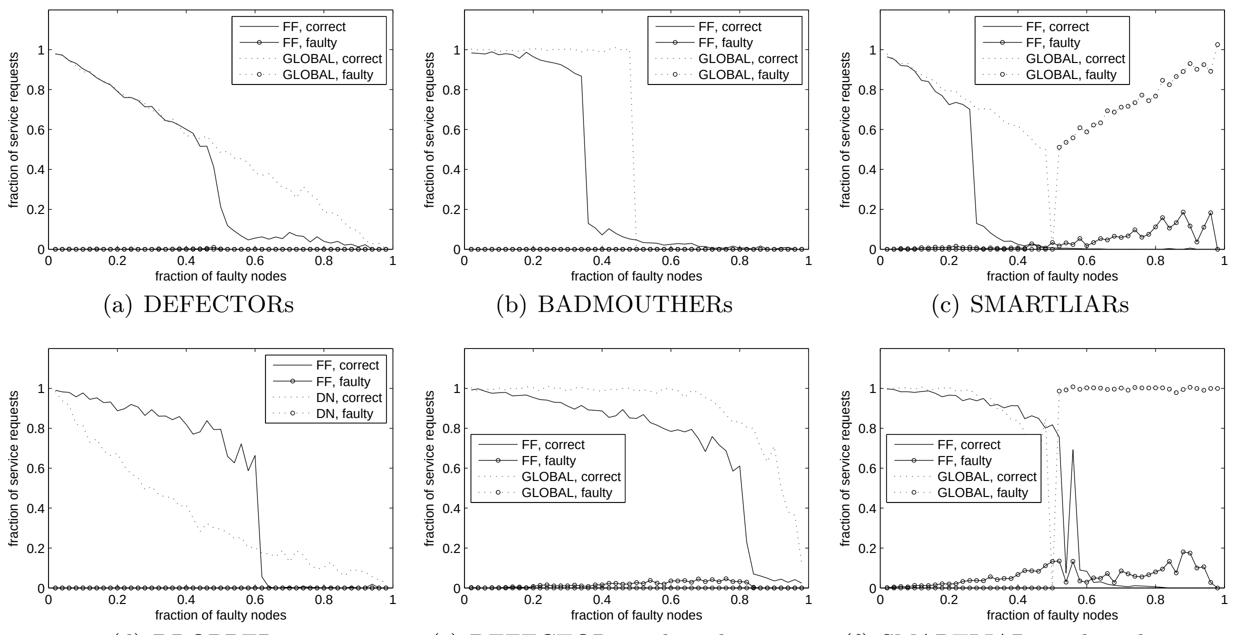
Fig. 4. Convergence rate of FF compared to other systems.

Similar convergence results were obtained in other setups, we now turn to asymptotic performance measurements.

4.3 Preventing service provisioning failures

In our measurement of the convergence rates we have fixed the fraction of faulty nodes to 30%. In the following experiments we vary the fraction f of faulty nodes and observe how it affects the performance of the system. We measure the asymptotic behavior of the system, i.e. after the measured values have converged,which takes between 500s to 3000s depending on the configuration. For each of the f values the simulation is run independently. We reduce the number of nodes to 256 due to the significant amount of simulation time required. We measure the fraction of service requests sent by correct nodes that resulted in either faulty or correct service provisioning. The

”faulty” and ”correct” fractions do not sum up to 1, since there can be service requests that never reach any provider. Figure 5(a) compares the performance of forward feedback (FF) with the



(a) DEFECTORs (b) BADMOUTHERs (c) SMARTLIARs

(d) DROPPERs (e) DEFECTORs with replication (f) SMARTLIARs with replication

Fig. 5. Resilience to different types of failures.

performance of the centralized reputation system (GLOBAL).

Until the fraction of DEFECTORs is lower than the half, the performance of forward feedback is identical to that of the centralized reputation system. The linear drop in the number of correctly provided services is related to the linear increase in the number of DEFECTORs in the population (one-on-one service allocation). In both cases, FF and GLOBAL, all requests for correct services are delivered. Both FF and GLOBAL successfully prevent the service requests from reaching any of the faulty service providers (the ”faulty” curves). When more than half of the nodes are DEFECTORs, GLOBAL is still able to tolerate this but the performance of FF decreases sharply. This is caused by a self-propelling effect in FF: once the nodes observe more faults the failure predictor deems some of the innocent neighbors as faulty, this leads to some routes being closed and causes additional false positives to happen.

4.4 Resilience to lying

In both cases, GLOBAL and FF, the peers receiving service can falsely report to the system about the success or failure of service provisioning. We first consider the simple case of a BADMOUTHER. A BADMOUTHER is a peer that behaves correctly in all respects but always reports a service provisioning failure to the system.

We rerun the resilience tests with the same conditions as previously (Section 4.3), but replacing DEFECTORs with BADMOUTHERs (Fig. 5(b)). BADMOUTHERs can collapse cooperation in the FF system when their fraction is larger than 0.33. For GLOBAL the collapse threshold is obviously 0.5 due to majority voting.

We next consider a more complex, collusive form of lying. Assume that an adversary controls a fraction f of SMARTLIARs. A SMARTLIAR always reports success when the service provider was another SMARTLIAR, otherwise a failure is reported. When a SMARTLIAR forwards a feedback message, the same rule is applied: the *success* flag in the feedback message is mutated to true if the service was provided by a SMARTLIAR, to false otherwise. The SMARTLIAR also provides a bad service whenever it is requested to. The adversarial strategy here is to use the SMARTLIAR population to convince the other peers by selective lying that they should use SMARTLIARS as service providers instead of correct peers.

Figure 5(c) presents the resilience test results. Collusive liars succeed in convincing peers to use defective services, though for the attack to succeed a fraction of liars greater than 50% is needed. For FF, there is a collapse in correctly provided services at 0.25, earlier than for BADMOUTHERs. Below 0.25 the performance of FF is the same as for the centralized reputation system, the fraction of correctly provided services is maximized and the faulty service provisioning is entirely eliminated.

4.5 Resilience to routing failures

While it is typically assumed in the state-of-the-art that the service location functionality is a reliable black box we do not require the underlying recursive service location mechanism to be reliable. First, let us consider message dropping.

When a service request is dropped, this causes a $T_{service}$ timeout at the requestor and negative feedback is propagated along the path. Let DROPPER be a node that behaves correctly, but drops all service requests. The resilience test results are on Fig. 5(d). We compare our system to the case where no fault-tolerance mechanism is in place (DN). Over time, the nodes learn to associate failure with their dropping neighbors, the droppers are routed around and the service requests are successfully delivered to providers. The network also learns that DROPPERs are correct when asked for services they are responsible for but are faulty for the services that require request forwarding. Even when half of the nodes in the network are dropping requests, 80% of the services can be reached even if some of them are provided by a DROPPER.

Other routing failures include mutating the fields in service requests and routing without making progress to the destination. These failures result in timeouts and subsequent negative feedback, if they occur consistently the can be tolerated with the same mechanism as for DROPPERs. The resilience curves are similar to those for DROPPER and we do not include them.

4.6 Service replication

So far we evaluated the scenario of one-on-one service allocation. We now consider the case of many-on-one service allocation in which services are replicated 4 times on average for increased fault tolerance. We rerun the resilience tests for the SMARTLIAR and DEFECTOR behaviors .

In the case of DEFECTOR (Fig. 5(e)), bad service provisioning is consistently eliminated. Replication dramatically shifts the point at which cooperation collapses as compared to one-on-one service allocation (Fig. 5(a)). Even when 80% of the nodes are providing bad services, the FF network learns to locate the correct replicas and routes towards them. The centralized reputation system has the advantage of global knowledge and is more effective at selecting correct replicas.

Collusive lying even with replication (Fig. 5(f)) lowers the fraction of good service provisioning and increases the fraction of faulty provisioning. Compared to the one-on-one case (Fig. 5(c)), the collapse of cooperation occurs much later, at 50% instead of 25% for FF, however FF convinced by SMARTLIARs fails to prevent faulty service provisioning which reaches its maximum of 10% at the collapse point. Again, below 20% both GLOBAL and FF are performing optimally, all service provisioning is correct.

4.7 Operation under churn

In all our simulations thus far there were no node departures or arrivals. Churn influences the performance of the forward feedback protocol in the following ways. First, as the neighbor sets change, the nodes must learn the behavior of their new neighbors. Second, we require in the protocol that the feedback messages follow the same path as the corresponding service request. The path must remain stable from the moment the service request is delivered to the provider to the moment the provider receives the feedback message. If during that time one of the nodes on the path departs, this causes feedback timeouts downstream from the departed node and consequently negative feedback is reported to the predictors. Lastly, when a node departs and the services it offered become unavailable, then all the currently routed requests for these services fail and negative feedback is propagated along the routing paths.

To test how the above effects influence the performance of the system we rerun the experiment with the same setup as in Section 4.2. We compare two cases, with and without churn. The churn model is based on the $Kad4$ data from the Stutzbach's et al. [8] study, which measured churn characteristics in live P2P networks. We increase the churn rate tenfold to accommodate for even the highest churn fluctuations. When a new node joins after the 200s mark it has 0.3 probability to take on the DEFECTOR behavior.

The results are on Fig. 6. Churn slightly lowers the fraction of correctly provided services, the main reason for this is that the destinations of service requests depart before the requests can reach them. Paradoxically, the convergence rate is faster under churn than without it. During the first 200s when there are no DEFECTORs, the network learns that some nodes are correct, it then takes some time before this is "forgotten", churn speeds up this process by making the DEFECTORs depart. Overall, even a ten times greater than realistic churn rate has minimal impact on the performance of the forward feedback protocol.

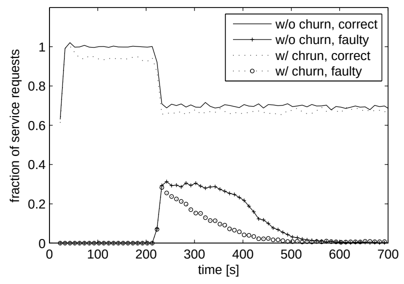


Fig. 6. The performance of the FF protocol under churn.

Similar effects of churn were observed in other experimental setups, we omit these result.

5 Discussion

Evaluation shows that the forward feedback protocol despite its limited observation sharing converges as fast as a fully centralized reputation system. We have also demonstrated our system to be robust in face of provisioning of bad services, collusive lying and different forms of routing failures, while also being able to tolerate churn. The performance of our system closely trails that of a fully centralized reputation system. The performance is lower only in cases when the fraction of faulty nodes is large (above 25%), which is unlikely to occur in Internet-scale systems.

In its current form the proposed system is vulnerable to the Sybil [9] and Eclipse [10] attacks, especially when absence of authentication is taken into account. An adversary can insert an arbitrary number of nodes that can open an arbitrary number of connections. We can either use some of the existing solutions to the problem [11, 12], or replace the initial optimism of the learning algorithm with limited initial trust so that newcomers are progressively probed before they become fully trusted neighbors.

The system we proposed is fault-tolerant but not fault-deterrent. Deterrence is necessary when failures are caused by selfish users trying to e.g. save on local peer resources such as network bandwidth. An effective form of deterrence is reciprocity (e.g. tit-for-tat). The neighbor behavior model learnt by p-concepts can be used for accurate reciprocation. We plan to build into the system different forms of adaptive indirect and direct reciprocity to enforce P2P cooperation.

Our solution can be customized to the needs of a specific application. The predicate set in the failure predictor can be tuned to be sensitive to any interdependencies between the attributes of the service request. Each peer can adjust its failure predictor independently of other peers, to suit the failure prediction to its own priorities and role in the network. The impact of such learning heterogeneity on the performance needs further research.

6 Applications

The forward feedback protocol is general enough to be integrated into any architecture in which there is recursive routing of service requests. This covers a wide range of applications. We focus here on data-related P2P applications.

6.1 Secure DHT routing

The proposed forward feedback algorithm can readily be applied to the securing routing in DHTs [13, 12]. The service is defined to be the delivery of an overlay message to the destination specified in the service descriptor and the service provisioning is simply the acknowledgment of the message delivery sent by the requestor. In this case the algorithm proposed in the paper can route around message droppers. This application can be extended in a number of ways. First, not only the droppers can be routed around but also nodes that do not ensure enough progress towards destinations in the overlay identifier space. This can be implemented by checking that every hop makes progress and penalizing the previous hops by bad feedback if the progress is slow. To go even further, if we use the round trip latency as the measure of progress instead of using the distance in the overlay identifier space, we essentially construct an overlay that learns the underlying latency model of the network and adapts to changing network conditions.

6.2 Preventing data corruption

Providing data is by far the most common modern application of a P2P network. Data could be files, key-value pairs in a DHT or data from a P2P sensor network. Data can be corrupted before it arrives at the peer that requested it. This can happen in a number of ways, ranging from simple corruption at the data source itself to mutation of the data by one of the forwarding peers. There are many existing solutions to this problem, most notably data replication and coding techniques. Forward feedback is a great addition to the existing methods in the following ways. When corruption occurs repeatedly at some peer then using feedback the network can pinpoint the faulty peer and either route around it when it is a forwarder or select alternative data providers when the corrupted peer is the data source. In contrast, to the existing mechanisms which only detect or mask the data corruption, the network running the forward feedback protocol actively prevents the corruption from occurring again once it is detected.

7 Related work

The reputation-based approaches [1, 2] to the problem of cooperation in P2P networks are top-down. The trust and reputation model is specified together with the mechanism of how peers exchange the reputation information, which is typically shared through a distributed data structure such as a DHT or propagated through gossiping. All these approaches strongly rely on persistent universally verifiable peer identities.

The second class of approaches is based on trust graphs [14, 15, 16, 17]. A trust graph is a digraph in which nodes represent peers and the arcs represent trust relationships. Aggregation algorithms are run on the trust graph to determine the trust towards the service provider. The trust graph is stored either in a DHT [14] or in the form of self-certifying reports stored locally by each node (e.g. cookies in [15], tokens or stamps in [18]).

Our solution may fall into the latter category, however, in our case the overlay graph is the trust graph and there is no need to store the trust graph in a distributed data structure, which greatly simplifies the design. Perhaps, the closest to our approach is the work of Repantis et al. [19] in which reputation information is stored in the neighbors and similarly to FF the neighbors of a peer A act as failure detectors of A. However, [19] relies on flooding and works only in unstructured networks.

8 Conclusions

The proposed architecture provides service provisioning P2P networks with fault tolerance to a wide range of message passing failures including mutating the service requests, collusive lying, and message dropping. Service requests are prevented from reaching faulty service providers, which protects the requestors from the unnecessary waste of resources during faulty service provisioning. When services are replicated the network learns to route to correct replicas and avoids the faulty ones.

Our solution places few requirements on the underlying system. Neither a DHT nor authentication are needed. Most loop-free, recursive service routing protocols can be made fault-tolerant using our solution, this widens the application domain beyond P2P networks and includes ad-hoc wireless networks, sensor networks and inter-domain routing.

References

[1] Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: CIKM. (2001) 310–317

[2] Xiong, L., Liu, L.: Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans. Knowl. Data Eng **16**(7) (2004) 843–857

[3] Kearns, Schapire: Efficient distribution-free learning of probabilistic concepts. In: Computational Learning Theory and Natural Learning Systems, Volume I: Constraints and Prospect, MIT Press. Volume 1. (1994)

[4] Rivest, R.L.: Learning decision lists. Machine Learning **2**(3) (1987) 229–246

[5] anonymized entry: Title

[6] Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM'01. (2001) 149–160

[7] Gummadi, P.K., Saroiu, S., Gribble, S.D.: King: estimating latency between arbitrary internet end hosts. In: Internet Measurement Workshop, ACM (2002) 5–18

[8] Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In Almeida, J.M., Almeida, V.A.F., Barford, P., eds.: Internet Measurement Conference, ACM (2006) 189–202

[9] Douceur: The sybil attack. In: International Workshop on Peer-to-Peer Systems (IPTPS), LNCS. Volume 1. (2002)

[10] Atul Singh, Tsuen-Wan Ngan, P.D., Wallach, D.S.: Eclipse attacks on overlay networks: Threats and defenses. In: INFOCOM. (2006)

[11] Yu, H., Kaminsky, M., Gibbons, P.B., Flaxman, A.: Sybilguard: defending against sybil attacks via social networks. In Rizzo, L., Anderson, T.E., McKeown, N., eds.: SIGCOMM, ACM (2006) 267–278

[12] Castro, M., Druschel, P., Ganesh, A.J., Rowstron, A.I.T., Wallach, D.S.: Secure routing for structured peer-to-peer overlay networks. In: OSDI. (2002)

[13] Moreton, T.D., Twigg, A.: Enforcing collaboration in peer-to-peer routing services. In Nixon, P., Terzis, S., eds.: iTrust. Volume 2692 of Lecture Notes in Computer Science., Springer (2003) 255–270

[14] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in P2P networks. In: WWW'03. (2003) 640–651

[15] Lee, S., Sherwood, R., Bhattacharjee, S.: Cooperative peer groups in NICE. In: INFOCOM. (2003)

[16] Mui, L., Mohtashemi, M., Halberstadt, A.: A computational model of trust and reputation for e-businesses. In: HICSS '02, IEEE Computer Society (2002) 188

[17] Singh, M.P., Yu, B., Venkatraman, M.: Community-based service location. Commun. ACM **44**(4) (2001) 49–54

[18] Moreton, T., Twigg, A.: Trading in trust, tokens, and stamps. In: First Workshop on Economics of Peer-to-Peer Systems. (2003)

[19] Repantis, T., Kalogeraki, V.: Decentralized trust management for ad-hoc peer-to-peer networks. In: MPAC '06: Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006), New York, NY, USA, ACM Press (2006) 6