A P2P-based Infrastructure for Adaptive Trustworthy and Efficient Communication in Wide-Area Distributed Systems

Haiying Shen*, *Senior Member, IEEE*, Guoxin Liu, *Student Member, IEEE*,
Jill Gemmill, *Member, IEEE*, Lee Ward[1]

Abstract—Tremendous advances in pervasive networking have enabled wide-area distributed systems to connect distributed resources or users such as corporate data centers and high-performance computing centers. These distributed pervasive systems take advantage of resources and enhance collaborations worldwide. However, due to lack of central management, they are severely threatened by a variety of malicious users in today's Internet. Current reputation- and anonymity-based technologies for node communication enhance system trustworthiness. However, most of these technologies gain trustworthiness at the cost of efficiency degradation. This paper presents a P2P-based infrastructure for trustworthy and efficient node communication in wide-area distributed systems. It jointly addresses trustworthiness and efficiency in its operation in order to meet the high performance requirements of a diversified wealth of distributed pervasive applications. The infrastructure includes two policies: trust/efficiency-oriented request routing and trust-based adaptive anonymous response forwarding. This infrastructure not only offers a trustworthy environment with anonymous communication, but also enhances overall system efficiency through harmonious trustworthiness and efficiency trade-offs. Experimental results from simulations and the real-world PlanetLab testbed show the superior performance of the P2P-based infrastructure in achieving both high trustworthiness and high efficiency in comparison to other related approaches.

Index Terms—Wide-area distributed systems, Peer to peer networks, Reputation systems, Anonymity, Efficiency

1 Introduction

The immense popularity of the Internet has provided a significant stimulus to the advancement of wide-area distributed pervasive systems that provide high scalability, efficiency, and other benefits not found in centralized models. These Internet-based distributed pervasive systems (e.g., corporate data centers and high-performance computing centers) attract Internet users from across the world to donate their computer resources for various objectives ranging from file sharing to distributed computing, from instant messaging to content distribution. For example, over the past year, the total traffic on the BitTorrent P2P file sharing application has increased by 12%, driven by 25% increases in per-peer hourly download volume [1], and 471.5 million users in one day connect to the Akamai P2P-assisted content delivery network (CDN) [2].

An open wide-area distributed system consists of many diverse and autonomous peers without preexisting trust relationships, in which malicious or selfish users may drop forwarding messages in node communication. Thus, it is important for nodes to communicate with each other efficiently (i.e., quickly with low overhead), trustworthily (i.e., reliably and anonymously). Currently, many technologies, including reputation- and anonymity-based approaches, have been proposed to prevent, detect, and counter malicious attacks and enhance system trustworthiness.

A reputation system collects, distributes, and aggregates feedback about participants' past behaviors to help peers decide whom to trust, encourage trustworthy behavior, and discourage uncooperative or dishonest behavior. In a distributed system, a node needs to choose a server from a candidate pool for a service such as forwarding a query or providing a file. A straightforward approach for nodes to use reputation metrics is to select the node with the highest reputation value as a providing server. However, biasing the nodes with the highest reputation may overload them and also prevent a system from fully taking advantage of all resources to achieve system-wide high performance.

Current research on reputation systems [3–15] mainly focuses on accurate reflection of node trustworthiness and high scalability of systems. However, even the accurate calculation of reputation values offered by a highly scalable reputation system may not be adequate as a mechanism to provide proper incentives for nodes to choose the best servers, and to improve the achievable efficiency of distributed systems. Reputation systems need to be complemented by an unbiased trust-based node selection policy that also considers efficiency for trustworthy and efficient node communication.

For anonymous transmission, tunnelled communication with a number of proxies between the two endpoints in the routing path provides anonymity for the two endpoints, since each node has no information about the message routing path other than the identity of its previous and succeeding node. Technologies such as Tor [16] randomly chooses the proxies. Directly using the randomized selection method in an open distributed system is not able to ensure successful routing, as the chosen nodes may be malicious nodes that drop or corrupt messages. Freenet [17] and Mute [18] achieve anonymity by requiring that all data is forwarded back hop-by-hop from a data provider to a requester rather than using direct communication. OneSwarm [19] provides anonymity by multi-path message forwarding and by providing users with configurable control over their data: data can be shared publicly or anonymously, with friends, with some friends but not others, or only among personal devices. However, these methods introduce high communication overhead. Efficiency for high performance should not be greatly compromised while exploring trustworthy technologies for distributed systems.

- * *Corresponding Author. Email: shenh@clemson.edu; Phone: (864) 656 5931; Fax: (864) 656 5910.*
- *Author[1] is with the Sandia National Laboratories, Albuquerque, NM 87185. Email: lee@sandia.gov.*
  Other authors are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634.
  E-mail: {shenh, guoxinl, gemmill}@clemson.edu

Unlike OneSwarm which relies on users' extra effort for configuration and high-cost multi-path message routing, we tackle this problem from another angle: leveraging reputation systems. Specifically, we propose an infrastructure built on a structured P2P middleware overlay to supply trustworthy and efficient node communications for wide-area distributed systems. It can be easily used by a variety of distributed systems such as P2Ps, grids, and collaborative clouds. The infrastructure incorporates two policies:

- **Trust/efficiency-oriented request routing policy (FairTrust).** *FairTrust* guides nodes to consider both reputation and capacity in node selection in order to choose trustworthy nodes while avoiding bottlenecks. It creates a trustworthy distributed communication environment, while providing flexibility in resource sharing between nodes. Furthermore, it facilitates taking full advantage of system resources for high performance of distributed systems.
- **Trust-based adaptive anonymous response forwarding policy (TrustAar).** *TrustAar* adapts the path length of response forwarding to node reputation in order to protect client-server connectivity and anonymity, while enhancing communication efficiency over current anonymity approaches.

In this paper, we use P2P file sharing systems as an example for a wide-area distributed system. This article combines the methods proposed in previous conference versions [20, 21], refines and enhances the methods and presents extensive experimental results. The rest of this paper is structured as follows. Section 2 introduces the P2P-based infrastructure design including the aforementioned two policies. Section 3 shows the performance of these policies in comparison with other related policies in simulation and on PlanetLab. Section 4 concludes this paper with remarks on possible future work. The supplemental document presents a concise review of related work and more experimental results.

2 Design of the P2P-based Infrastructure

2.1 Background and Overview

Structured P2P overlay networks are a class of decentralized systems that partition ownership of a set of objects among participating nodes and can efficiently route messages to the unique owner of any given object. The overlay network provides two main functions: `insert(key,data)` and `lookup(key)` to store the data to a node responsible for the key and to retrieve the data. The message for each function is forwarded from node to node through the overlay until it reaches the data's owner. Each node maintains a routing table, which records its neighbors in the overlay network. After a node receives a message, it forwards the message to one of its neighbors in the routing table based on the P2P routing algorithm.P2P overlays such as Tapestry [22] and Cycloid [23] have no rigidly defined topologies and routing algorithms (i.e., an entry in a node's routing table can have multiple node choices), which facilitates the implementation of anonymity in the infrastructure. Any such P2P overlay can be the basis for our system's infrastructure. We choose Cycloid [23] as an example to serve as the foundation of the infrastructure.

Anonymity allows data sharing between clients and servers in such a manner that no one can determine the identities of the clients and servers. In order to hide the client and server of a communication, it is necessary to form an anonymous path between the two endpoints and ensure that nodes on the path do not know where the endpoints are. A P2P overlay contributes to achieving anonymity in the application level. It obscures the physical locations of nodes from each other, and restricts a node's view only to its neighbors. In addition, initial messages and forwarded messages are constructed and processed similarly, so nodes cannot differentiate message forwarding neighbors from initial message generating neighbors. Furthermore, tunnelled communication provides certain protection to two endpoints.

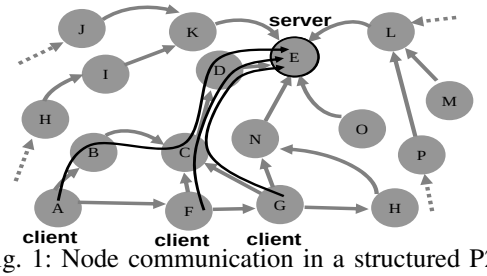


Fig. 1: Node communication in a structured P2P.

Figure 1 depicts a tunnelled communication between client A and server E in traditional structured P2P overlays. The request is routed along the path $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$, and the response is forwarded directly from E to A. We define a *client-server path* as a tunnelled path from a client to a server, and a *server-client path* as a tunnelled path from a server to a client. The client-server path in the figure is not directly between A and E; rather, it travels through three other nodes. Each node on the path does not know the source or the final destination of the messages transferred. For example, node C does not know if the communication is between nodes B and D or if they are passing the messages elsewhere. Nodes only have knowledge of their immediate neighbors in the proxy chain. This provides a certain degree of anonymity. However, the direct communication from E to A reveals their identities to each other. Also, in most traditional structured P2P overlay networks, because of their strictly controlled topologies and rigidly defined routing algorithms, there is only one node option in the system for each entry in a node's routing table, and hence only one neighbor option to forward a request. This enables a malicious node to analyze the message traffic to identify the client, server, and path. Also, the rigid routing algorithm cannot avoid overloaded nodes in routing, leading to low efficiency. For example, in Figure 1, requests from clients A, F and G towards E must travel through nodes C and D, which could easily become overloaded.

Cycloid's flexibility in topology construction and routing algorithm (that enables to select a neighbor from multiple options) facilitates the exploration of techniques for both anonymity and efficiency. We propose the *FairTrust* policy to prevent traffic analysis in the client-server path, and the *TrustAar* policy for data forwarding in the server-client path. These policies provide anonymity to both clients and servers with reduced overhead through a harmonious trade-off between reliability/anonymity and efficiency.

In this paper, we assume the existence of a decentralized reputation system [3, 4] that is scalable and can accurately and securely provide reputation values of nodes. We also assume that a node's reputation value can represent its general trustworthiness, and that a high-reputed node is unlikely to drop (or corrupt) a message or be a spy or unscrupulous organization that conducts censoring or analysis of data traffic. Please refer to papers [3–15] for the designs of scalable, accurate and secure reputation systems.

As our work does not focus on reputation system design, these works are beyond the scope of this paper. Specifically, we can use the reputation system designed for anonymous networks proposed in [15]. In this reputation system, nodes are represented by pseudonyms. The reputation of each node is closely related to its real identity rather than to its current pseudonym. This allows an honest node to switch to a new pseudonym, thus keeping its good reputation, while hindering malicious nodes from erasing their bad reputations with new pseudonyms. As a result, the reputation values can be reported and queried without disclosing node real identities.

2.2 Trust/Efficiency-Oriented Request Routing Policy

Problem Statement. This section handles the problem of forwarding a request to its destination trustworthily (i.e., reliably and anonymously) and efficiently. Cycloid has a flexible topology in that its routing table has a set of neighbors in each entry. For example, Cycloid node i (4,10111010) has one neighbor entry pointing to nodes (3,10100000), (3,10100001) and (3,10100010). If it receives a query that should be forwarded to this neighbor entry based on its original routing algorithm, there are three candidates to which the query can be sent. This topology flexibility enables a node to choose a neighbor from several options to forward a request along the client-server path routing. Then, how should a node be chosen to forward a query for trustworthy and efficient communication?

Discussion on Existing Methods. Randomized node selection in Tor [16] is not enough to ensure communication security with successful routing in an open distributed system. With malicious nodes, the requests may be dropped, corrupted, or delivered to a malicious node instead of a legitimate file owner. Also, the randomized node selection cannot guarantee that the selected nodes are never overloaded, possibly degrading the routing efficiency.

With a reputation system, a straightforward approach for nodes to utilize reputation metrics is to select the node with the highest reputation value as a providing server. Such an approach may give rise to unexpected problems. First, biasing high-reputed nodes may overload them and cause inefficiency. Second, some high-reputed but not-highest-reputed nodes are excluded from service offering. Consequently, their resources cannot be fully utilized, and they are deprived of opportunities to earn their reputation. Eventually, this will result in gradual decomposition of the nodes in the system. Therefore, as explained in Section 1, the reputation values have to be complemented with an unbiased trust-based policy that helps a node decide a candidate for service with consideration of both trustworthiness and efficiency.

To avoid overloading high-reputed nodes, the "peer-approved" policy [24] enables nodes to download files from others with lower or equal reputations, and the "comparable reputation" policy [25] restricts nodes to communicate only with other nodes at the same reputation level. These policies help high-reputed nodes to avoid low-reputed nodes, but also have a number of drawbacks. First, newly-joined nodes will be in a hazardous environment, surrounded by low-reputed or even malicious nodes for communication. Second, newly-joined nodes are provided with few opportunities to increase their reputation. Third, median-reputed nodes are deprived of access to services provided by high-reputed nodes, and also high-reputed nodes are deprived of access to services provided by median-reputed nodes. This prevents distributed systems from achieving their ultimate goal of wide resource sharing.

Our Proposed Policy. In choosing a forwarder from the options for request routing, biasing the highest-capacity nodes cannot guarantee reliable routing due to uncooperative participants. Similarly, biasing the highest-reputed nodes causes node overload and low efficiency. To address these problems, we introduce *FairTrust* that considers node capacity and reputation simultaneously to choose a providing node from a candidate pool. The policy contributes not only to the trustworthiness of the system but also to the overall high performance by making full utilization of all node resources and avoiding overloading high-reputed nodes.

To identify trustworthy nodes from untrustworthy nodes, many reputation systems [3–15] set a reputation threshold. If a node has a reputation no less than the threshold, it is considered a trustworthy node; otherwise, it is considered an untrustworthy node. Trustworthy nodes also provide trustworthy services, though their QoS may be lower than that of the highest-reputed node. In our daily life, when we buy a computer, we do not have to choose the one with the highest rating and highest price. We can choose a less expensive computer with high rating that still meets our requirements. Similarly, in a distributed system, different requests have different requirements for the levels of reputation. For example, while a node must choose the highest-reputed nodes for confidential data, it does not necessarily depend on such nodes when conveying public news with delay tolerance. Also, a node may prefer to download a file from a higher-reputed node rather than from the highest-reputed node in order to avoid flooding the highest-reputed node with requests. Therefore, *FairTrust* trades reliability for efficiency by mapping messages of different desired trust levels to nodes with corresponding reputation levels. Later on, we will introduce an economic payment method to provide incentives for nodes to choose trustworthy nodes based on their own needs instead of always choosing the highest-reputed node.

Algorithm 1 (in the supplemental document) shows the pseudocode of the *FairTrust* policy. Specifically, when node i needs to choose a server from a number of candidates, it firstly determines reputation level T based on its desire. It then acquires the reputation value of each candidate, and filters out candidates whose reputation values are below T. The rest of the candidates include both the highest-reputed nodes and nodes that are trustworthy enough based on the requester's desired trust level. Then, as an unbiased trust-based policy, *FairTrust* guides a node to sift the rest of candidates for the final choice for request routing with the consideration of both trustworthiness and efficiency.

For efficiency purposes, the policy takes into account load status and proximity to ensure that no node becomes a bottleneck for request routing for quick routing. Our previous work [26] introduced a method to calculate an integer for a node to represent its proximity. We define *congestion* of node i as l_i/c_i, where l_i represents the load of node i and c_i represents the capacity of node i. We refer to the node whose actual load is larger than its capacity (*i.e.* $l_i > c_i$) as an *overloaded node*; otherwise, it is a *lightly loaded node*. To select a forwarding node, node j first considers load status by identifying lightly loaded nodes. From the identified candidates, it then considers proximity, in which nodes logically closer (in hops) to the destination and then physically closer to node j have higher priority to be chosen.

Checking every server's load status and proximity is not an efficient method. To reduce overhead, instead of probing all of the neighbors to find the best candidate, we restrict the search space to a small set of size b. That is, b nodes are randomly chosen first and then the nodes in the set are probed to retrieve load status and proximity information. If all nodes are overloaded, then the node with the lowest congestion is selected. If only one node is lightly loaded, then this node is selected. If multiple nodes are lightly loaded, a node selects the best candidate with two extra criteria explained previously: closeness to the target ID by logical distance in the overlay network and closeness to the node by physical distance. In the case that the candidates have the same logical distance, the node with shorter physical distance to the selecting node is chosen. In the case that the nodes have the same physical distance, then a node is randomly chosen. For example, if $b = 2$, node i receives a query with key (2,10100011). It first identifies its query forwarding neighbors according to the Cycloid routing algorithm and chooses neighbors with reputations no less than the required trust level. Node i then randomly chooses two options (3,10100010), (3,10100001) among the identified neighbors. It then probes each node's congestion and physical distance and finally chooses one node based on the algorithm in the *FairTrust* policy.

In addition to message forwarding, the proposed method for choosing a server can also be used in server selection for other purposes, such as file server selection and routing neighbor selection. Rather than always biasing on the highest-reputed nodes, *FairTrust* enhances system efficiency by distributing the load among trustworthy nodes with reputation values higher than the client's required level; thus, it will not overload the highest-reputed nodes while avoiding untrustworthy nodes. Also, offering high-reputed (not-highest-reputed) nodes opportunities to provide services enables them to further increase their reputations and takes advantage of their resources. Unlike rigid routing, *FairTrust* prevents a malicious node from identifying the routing path to find the client and the server while improving routing trustworthiness and efficiency by relying on randomized selection with consideration of trustworthiness and efficiency.

A node may tend to specify a desired reputation level higher than what it really needs in order to receive high QoS. Also, low-reputed nodes may not be motivated to provide high QoS since their low reputations do not prevent them from receiving high QoS. To handle these problems, *FairTrust* adopts the economic payment method in [27]. That is, a server prices its services based on its reputation and service quality, and a client pays a server for the service it requests. The payment is in the form of virtual credits. This economic payment method encourages nodes to provide high QoS in order to gain higher reputations that allow them to earn more credits to buy services, while also preventing the situation where higher-reputed and lower-reputed nodes receive the same QoS for the same cost. Trustworthiness aside, efficiency can also be controlled by the economic payment method. Servers can adaptively raise their service prices to discourage clients from asking for service from them when overloaded, and offer discounts when lightly loaded. The adaptive load control not only prevents servers from being overloaded, but also helps to take full advantage of server capacities. To address the newly joined node problem mentioned above, we can offer each node a certain number of credits on first joining the system that can be used for transactions to increase its reputation. This economic payment method makes service accessible to all nodes and provides freedom in service requesting and offering while providing protection from malicious nodes and leech nodes.

2.3 Trust-based Adaptive Anonymous Response Forwarding Policy

Problem Statement. Most traditional P2P overlay networks use direct connections to download files, thus disclosing the identities of the server and the client. A general approach to achieving anonymity on overlay networks is to construct an indirect path between a client and a server. For instance, in Freenet [17], Mute [18] and OneSwarm [19], after a file is located, instead of using a direct connection between two nodes, the data is sent back along the nodes in the request routing path. Figure 2 depicts an example of such anonymous communication. However, the hop-by-hop file forwarding comes at the cost of high communication overhead and leads to inefficient communication considering the fact that file downloading constitutes most Internet traffic. For instance, measurement results [28] showed that only 2.24% of all connections were download connections, but they were carrying 70.5% of the total traffic. P2P resource sharing applications have evolved into one of the major traffic sources, approximately 80% of Internet traffic. To reduce the overhead for request forwarding, Mantis [29] allows clients to exchange anonymity for download efficiency. It uses anonymous communication to search for files and send control signals, while letting the data be sent directly from the server to the client using return address spoofed UDP as shown in Figure 3. Instead of providing full anonymity, Mantis only protects the privacy of servers.

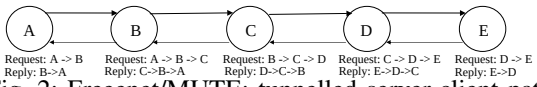


Fig. 2: Freeenet/MUTE: tunnelled server-client path.

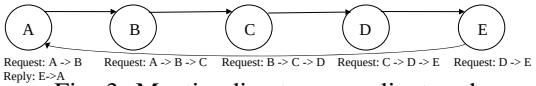


Fig. 3: Mantis: direct server-client path.

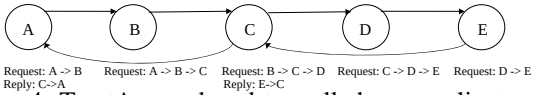


Fig. 4: TrustAar: reduced tunnelled server-client path.

Our Proposed Policy. We propose the *TrustAar* policy to provide anonymity for both clients and servers, and to reduce communication overhead by shrinking path lengths based on endpoint reputation. In order to prevent proxies from seeing the contents of the queried data, a client generates a public and private key pair for its request and sends the public key along with its request. The server uses the public key to encrypt the queried file before forwarding it back to the client and the client uses the private key to decrypt the file. Since proxies on the path do not know the private key, they are not able to decrypt the file.

The server needs to prevent the client from identifying that it is the source of the file, and the client needs to prevent the server from identifying that it is the requester of the file. The level that an endpoint i needs protection from another endpoint j is determined by the trustworthiness of endpoint j. That is, endpoint i needs higher protection from lower-reputed endpoint j and needs lower protection from higher-reputed endpoint j. Also, more tunnelled transfers along

the server-client response forwarding path provide higher anonymity but lower efficiency. We use t_c and t_s to denote the reputation of the client and of the server, respectively. To provide anonymity while guaranteeing high efficiency, *TrustAar* adapts the extent to which a tunnelled server-client path length shrinks to the value of $t = \min\{t_c, t_s\}$. Particularly, the higher the value of t, the fewer hops are needed in the server-client path, and vice versa. If the reputation values of two endpoints are very high, i.e., both of them can be trusted, then it is acceptable not to use proxies. We describe this policy in more detail in the following.

As mentioned previously, we use the reputation system designed for anonymous networks proposed in [15]. In this system, a node's reputation is queried based on its pseudonyms, and nodes always change their pseudonyms to avoid being tracked. As with other structured P2Ps, Cycloid requires $\log n$ hops per lookup request on average, where n is the network size [23]. The reputation system normalizes node reputation values to the range of $[1, \log n]$. After a server receives a request from a client, it queries the reputation values of the client and itself (t_s and t_c) from the reputation system, using the client's pseudonym contained in the file request and its own pseudonyms. Let l denote the path length of the client-server path. Then, the response is approximately passed l/t ($t = \min\{t_c, t_s\}$) hops along the original client-server path from the server to the client.

For example, as shown in Figure 4, if $t = 2$, E sends data to C, and C repeats the same process to send data to A. One question is, how can E know C's IP address, and can C know A's? We first introduce a simple method to address this question and then present a method for higher anonymity protection. In the simple method, the server passes an address request including its own IP address and a counter equal to t along the server-client path. The counter is decremented each time before it is passed to the immediate preceding node, and the node with counter equal to 0 is the node t hops away from the server. The node then sends its IP address back to the server, and repeats the operation after it receives the file from the server.

However, if using a certain rule to determine the forwarding proxies, malicious nodes along the path may tamper with or falsify the counter for traffic analysis, and the server may tamper with or falsify the counter to find proxies along the path. Also, disclosing the IP addresses of the server and relay proxies in the server-client path to every group of t nodes in the path segment leaks partial path information, which could be used by malicious nodes for traffic analysis. We use a cryptography technique to handle these problems. Algorithm 2 (in the supplemental document) shows the pseudocode of the *TrustAar* policy. The reputation system [15] signs the responses for $\{t_c, t_s\}$ before replying to the server in order to prevent nodes from tampering with the values. The server generates a public and private key pair, and sends the public key and the signed $\{t_c, t_s\}$ along the server-client path. Each message receiver first checks the authority of the signature of $\{t_c, t_s\}$, and then decides if it becomes a proxy in the server-client path with $1/t$ probability. Specifically, a message receiver i generates a random number v from [0,1]. If $v \leq 1/t$, it forwards the message to the next hop; otherwise, it becomes a server-client path proxy. In this case, proxy i uses the server's public key to encrypt its own IP address, and sends it with its public key along the client-server path successors until reaching the node that has the private key to decrypt the message to derive proxy i's IP address (i.e., the server). The server then encrypts the file using proxy i's public key and sends it to proxy i. Node i decrypts the message to derive the file. It conducts the same operation as the server and sends the file encrypted by the next proxy j's public key to j in the server-client path. This process repeats until the client receives the message from a proxy k. The client then passes its encrypted IP address to node k, and receives the file encrypted by the client's public key from node k.

Rather than relying on static hop-by-hop or direct communication in response forwarding, *TrustAar* dynamically adjusts path lengths based on the endpoints' trustworthiness, ensuring high anonymity and efficiency. Admittedly, there are problems that need to be solved for strong anonymity protection. For example, the server and the relay proxies in the server-client path may collude to find the identity of the client, the client and the proxies in the client-server path may collude to find the identity of the server, or the server may try to find all the proxies in the path by tampering with t_c or t_s. Also, as high-reputed endpoints have fewer proxies in file transfers, high-reputed nodes become weak points in the network for malicious nodes to compromise in order to find identifies of file owners. We leave these interesting problems as our future work.

3 Performance Evaluation

We conducted experiments on a simulator developed by ourselves and on the PlanetLab real-world testbed [30]. We compared the performance of *FairTrust* with the *Random*, *MaxTrust*, and *MaxCap* policies. Given a number of nodes, *Random* chooses a node randomly, while *MaxTrust* and *MaxCap* choose the node with the highest reputation and available capacity, respectively. We also compared the performance of *TrustAar* with *Freenet* [17], *Mute* [18] and *Mantis* [29], and include the results of the "comparable reputation" policy in [25] denoted by *SameTrust*, in which a requester randomly selects a server from the servers with the same reputation as itself.

TABLE 1: Environment and algorithm parameters.

Environment parameter	Default value
Cycloid dimension d	8 (simulation), 6 (PlanetLab)
Number of nodes n	2048 (simulation), 384 (PlanetLab)
Number of options per hop or file servers per request	10 (simulation), 6 (PlanetLab)
r (node reputation)	Negative Bounded Pareto: shape = 10 lower bound = 0.1 upper bound = 1
Node routing capacity	Bounded Pareto: shape = 2 lower bound = 1 upper bound = 100
Node file service capacity	Bounded Pareto: shape = 2 lower bound = 0.5 upper bound = 50
Forwarding latency in a light/overloaded node	0.02/0.1 second (simulation) Actual time (a)/a + 0.25s (PlanetLab)
File service latency in a light/overloaded file server	0.2/1 second (simulation) 1/5 second(s) (PlanetLab)

File requests were consecutively generated with a random source node and a random target. As in [31], file requests are generated according to a Poisson process at an expected rate of one per second. Table 1 lists the parameters and their default values in simulation and on PlanetLab respectively, unless otherwise specified. We assumed a bounded Pareto distribution for the capacity of nodes [32]. In the experiment, a node with capacity c_i can handle c_i queries at one time. We define node i's load (l_i) as the number of requests in its request processing queue. If node i has more than c_i queries in its queue, it is overloaded. In order to make sure

some high-capacity nodes have low reputations, we set the reputation of half of the nodes with capacity higher than ten times of the capacity lower bound to 0.2. We assume a bounded Pareto distribution for nodes' reputation (i.e., a majority of nodes are not malicious). The probability that a node provides a requested service equals to its reputation.

The metrics to evaluate the performance are listed below:

• *Success rate.* After a request arrives at a server, the probability that a request can be resolved depends on the server's reputation. The success rate is the successfully resolved service requests over the total requests. This metric shows the trustworthy performance on receiving successful service from servers.

• *Number of failures.* A request routing fails if one node in the routing fails to forward a request. This metric represents the effectiveness of a routing policy in trustworthy routing.

• *Congestion.* Recall that node i's congestion is defined as l_i/c_i. Ideally, congestion should be no more 1, which means that the node is lightly loaded. The higher the congestion is greater than 1, the less efficient performance. We record the maximum congestion of each node during the experiment process, and take the 99.9th (99th) percentile of these maximum congestions as *the 99.9th (99th) percentile maximum congestion* to show the performance.

• *Number of overloaded nodes encountered.* This metric shows the effectiveness of a policy in avoiding overloaded nodes in directing traffic flow in order to reduce request routing latency.

• *Lookup path length.* This is defined as the number of hops in a routing path. We use this metric to evaluate the performance of *FairTrust* on reducing lookup path length.

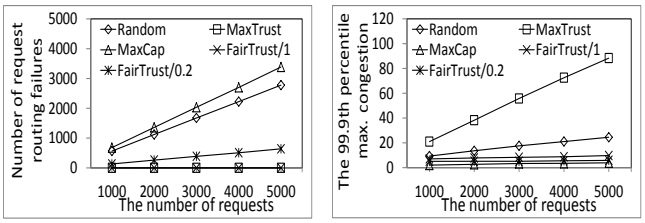
• *Request routing time.* This is the time used for a request routing. The number of overloaded nodes in request routing and the lookup path length are main factors for request routing time. This metric represents the efficiency of routing.

• *Request processing time.* This is the time used for processing file requests in servers. This metric represents the efficiency of server selection policies in avoiding overloaded file servers to achieve fair load distribution, and ultimately speeding up service provision.

• *File retrieval time.* This is the sum of the routing time, request processing time and response forwarding time. The metric reflects the performance of efficient and trustworthy file retrieval.

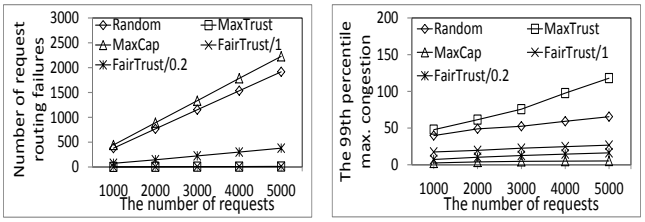
3.1 Trust/Efficiency-Oriented Request Routing Policy

This experiment aims to show the performance of *FairTrust* in achieving both high trustworthiness and high efficiency in request forwarding. We varied the number of requests from 1000 to 5000 with a step size of 1000. In *FairTrust*, we set two desired trust levels, 1 and 0.2, and represent them as *FairTrust/1* and *FairTrust/0.2*, respectively. Figure 5(a) and Figure 6(a) show the number of failures in routings for different policies in simulation and PlanetLab experiments, respectively. We observe that the number of failures of *MaxCap* grows rapidly when the number of requests increases, followed by *Random* and *FairTrust/0.2*. We also observe that *MaxTrust* and *FairTrust/1* generate the least failures, and that they exhibit similar behaviors. *MaxCap* considers node available capacity, but does not consider the trustworthiness when choosing a forwarding node. Thus, it performs the worst among the different policies in terms of trustworthy routing. *Random* also does not consider trustworthiness, but it may choose high-reputed nodes, leading to fewer failures than *MaxCap*. In contrast, *MaxTrust* and *FairTrust* take into account node trustworthiness in routing, and achieve the best trustworthy performance. *FairTrust/0.2* has relatively more failures. Considering its desired success rate is 0.2, it still performs effectively in terms of routing trustworthiness.



(a) Number of failures in routings (b) The 99.9th percentile max. cong.

Fig. 5: Trustworthy and efficient routing in simulation.

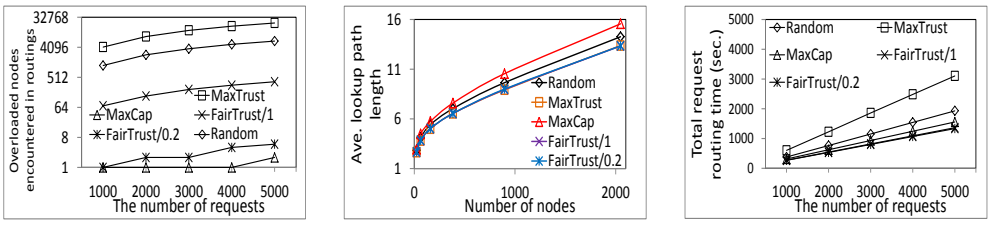


(a) Number of failures in routings (b) The 99th percentile max. cong.

Fig. 6: Trustworthy and efficient routing on PlanetLab.

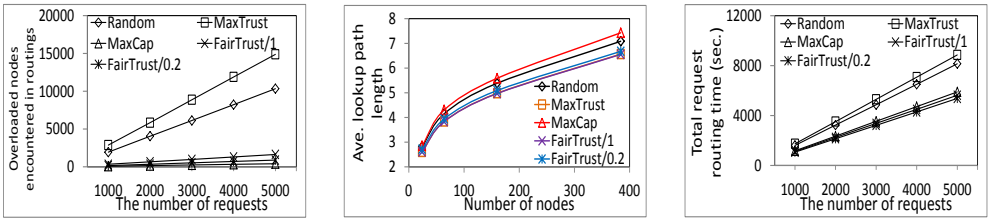
Trustworthiness aside, efficiency is also important in measuring the overall performance of the different routing policies. Figure 5(b) and Figure 6(b) demonstrate the 99.9th percentile and 99th percentile maximum congestion in each policy in simulation and PlanetLab experiments, respectively. We see that the congestion rates increase as query load increases. Also, *MaxTrust* has the highest congestion. This is because *MaxTrust* biases on the highest-reputed nodes for routing. By comparison, *Random* has lower congestion than *MaxTrust*, because it is comparatively efficient in load distribution due to its randomness. *MaxCap* and *FairTrust* lead to much lower congestion rates. They tend to direct requests to lightly loaded nodes such that nodes are not likely to be overloaded. *FairTrust/0.2* has a slightly lower congestion rate than *FairTrust/1* because *FairTrust/0.2* has more options for choosing a lightly loaded node. The results also show that *FairTrust* has marginally higher congestion than *MaxCap*. This is the cost for its high trustworthy performance. However, the low congestion benefit of *MaxCap* is outweighed by its poor trustworthy performance as shown in Figure 5(a) and Figure 6(a).

Routing latency is determined by two factors: the number of overloaded nodes encountered in routings and lookup path length. Figure 7(a) and Figure 8(a) show the number of overloaded nodes encountered in request routings. The number increases as the query load increases, and the number of *MaxTrust* increases faster than others. This is because *MaxTrust* overburdens the highest-reputed nodes by biasing them. This result confirms that *MaxTrust* cannot distribute query load among nodes in balance, while *MaxCap* and *FairTrust* avoid generating overloaded nodes. Fewer overloaded nodes in routing leads to higher efficiency in routing. *FairTrust* has marginally higher results than *MaxCap*. This is because *FairTrust* considers node reputation when choosing a forwarding node, reducing the number of available options. The relative performance between *MaxTrust*, *Random*, *MaxCap* and *FairTrust* is consistent with the 99.9th percentile maximum congestion in Figures 5(b) and 6(b).



(a) Overloaded nodes in routings (b) Lookup path length (c) Total request routing time

Fig. 7: Efficiency of request routing policies in simulation.



(a) Overloaded nodes in routings (b) Lookup path length (c) Total request routing time

Fig. 8: Efficiency of request routing policies on PlanetLab.

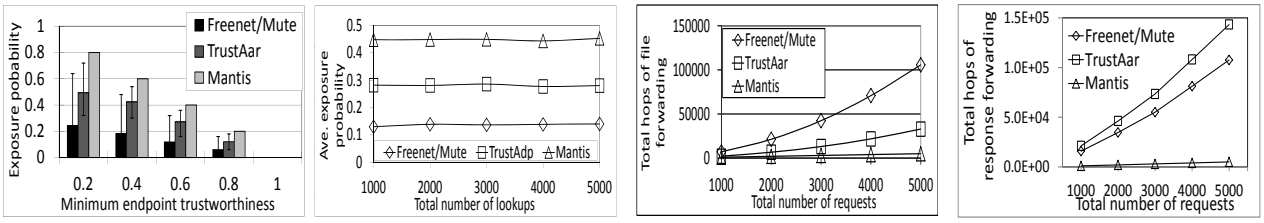
In this experiment, if a request was dropped due to low reputation of a forwarder, the sender resent the request and one more hop was added to the lookup path length. This process was repeated until the request was successfully forwarded. Figure 7(b) and Figure 8(b) show that average lookup path length over different numbers of nodes in simulation and on PlanetLab, respectively. The path length follows *MaxCap*>*Random*>*FairTrust/0.2*≈*FairTrust/1*≈*MaxTrust*. Successful forwarding in each hop is a main factor in determining the path length as dropped requests are resent. Since the failure probability in forwarding follows *MaxCap*>*Random*>*FairTrust/0.2*>*FairTrust/1*≈*MaxTrust* as in Figures 5(a) and 6(a), the lookup path length follows a similar trend. These results confirm that *FairTrust* has more efficient and trustworthy routing performance than others. Note that the average number of contacts to the reputation system per request of *MaxTrust* and *FairTrust* is similar to the average lookup path of *MaxTrust*, while *MaxCap* does not need to contact the reputation system.

Figure 7(c) and Figure 8(c) show the total request routing time of successfully delivered requests as the combined effect of overloaded nodes and lookup path lengths in simulation and on PlanetLab, respectively. Three main observations can be made from the figures: (1) *MaxTrust* has the highest routing latency, followed by *Random*, due to their neglect of node load status; (2) *MaxCap* has much lower routing latency because relying on lightly loaded nodes speeds up the routing process; (3) *FairTrust* significantly decreases routing latency because it directs query load to lightly loaded nodes, as does *MaxCap*, and reduces lookup path length, although *FairTrust* does not perform as well as *MaxCap* in reducing overloaded nodes and congestion rates. The results show that the combined effect of overloaded node reduction and lookup path length reduction result in great savings of routing latency, causing *FairTrust* to outperform *MaxCap*. We calculated that *FairTrust*'s average routing time per request is around 1s on PlanetLab.

3.2 Trust-based Adaptive Anonymous Response Forwarding Policy

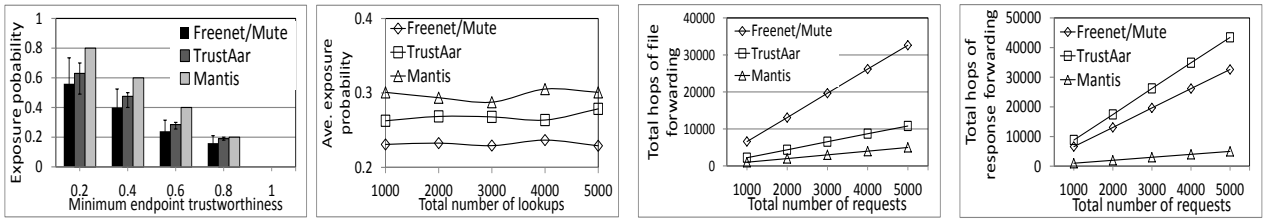
Anonymity is important in building a trustworthy environment. The probability that an endpoint is at a risk of ID exposure to the other endpoint depends on three factors. First, whether the other endpoint is a malicious node or not. Second, the number of proxies forwarding a file back from a server to a client; more forwarding proxies provide higher anonymity protection to the endpoints. Third, whether the proxies are malicious or not. Since the third factor is determined by the request routing policy and *FairTrust* chooses trustworthy nodes, we did not consider it in the evaluation of *TrustAar*. Because the first factor depends on the node trustworthiness, we define the exposure probability due to the first factor as the minimum endpoint trustworthiness, and define the probability due to the second factor as $1-d\times 0.1$, where d denotes the number of proxies between a server and a client. The product of the two factors is the probability that a server ID is disclosed to a malicious client. *Freenet* [17] and *Mute* [18] achieve anonymity by requiring that all data be forwarded back hop-by-hop along the server-client path, while *Mantis* [29] allows the data to be sent directly from a server to a client. *TrustAar* provides a different degree of anonymity based on the minimum endpoint trustworthiness by tuning the number of forwarding hops to adapt to the minimum endpoint trustworthiness. In this experiment, we compare the anonymity protection and efficiency performance of these policies. We assume that there are 5 levels of trustworthiness from 0.2 to 1 with a 0.2 increase for each level. We assume that the server has trustworthiness 1, so the minimum endpoint trustworthiness is always the client's trustworthiness. In simulation, as file encryption and decryption dominate the latency, we assumed that the latency of finding a proxy and sending a file to the proxy takes 0.4s. In the PlanetLab experiments, the size of a file and a key was set to 64KB and 1KB, respectively.

Figure 9(a) and Figure 10(a) show the average and the 1st and 99th percentiles of exposure probabilities versus the minimum endpoint trustworthiness level in simulation and PlanetLab experiments, respectively. A number of important observations can be made from this figure: (1) As we expected, the exposure probability of each policy decreases as the minimum endpoint trustworthiness increases. When clients have the highest trustworthiness, the exposure probability is 0, which means there is no response forwarding that



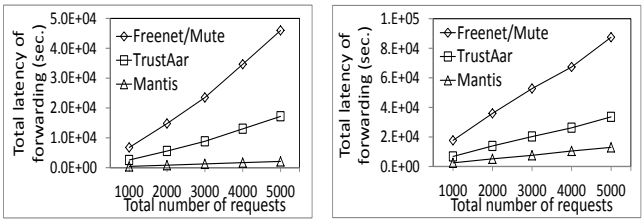
(a) Exposure probability (b) Average exposure probability (c) Total hops of file forwarding (d) Total hops of forwarding

Fig. 9: Efficiency of different anonymity response forwarding policies in simulation.



(a) Exposure probability (b) Average exposure probability (c) Total hops of file forwarding (d) Total hops of forwarding

Fig. 10: Efficiency of different anonymity response forwarding policies on PlanetLab.



(a) Simulation (b) PlanetLab

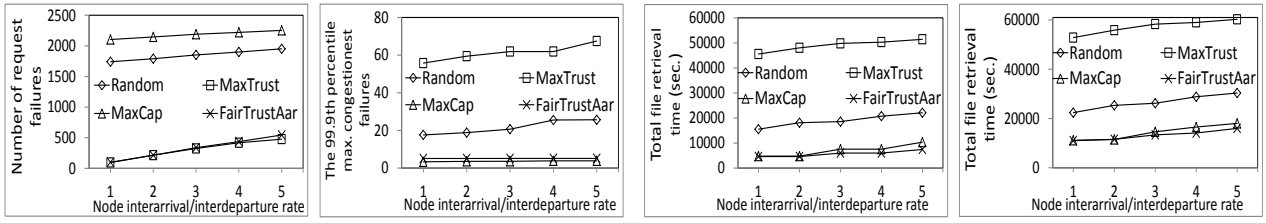
Fig. 11: Total latency of forwarding.

makes a server unsafe; (2) *Mantis* has the highest exposure probability, *TrustAar* reduces the probability more than one third, and *Freenet* generates the least exposure probability; (3) *Freenet/Mute* exhibits a larger variance than *TrustAar*, while *Mantis* has no variance at all. This is because the exposure rate of *Mantis* depends on the minimum endpoint trustworthiness directly without the involvement of proxies. Figure 9(b) and Figure 10(b) show the average exposure probability of the policies versus the number of requests in simulation and PlanetLab experiments, respectively. They demonstrate similar behaviors for each policy as in Figure 9(a) and Figure 10(a). As predicted, more hops along the server-client path result in higher server privacy protection. *Mantis* has the highest exposure probability due to its direct communication between a client and a server. On the other hand, *Freenet/Mute* has the lowest exposure probability due to its static hop-by-hop routing along the server-client path. However, its benefits are outweighed by its high overhead and efficiency degradation in forwarding. By adapting the number of hops to the minimum endpoint trustworthiness elastically, *TrustAar* improves the privacy protection of *Mantis* significantly, and reduces the overhead of *Freenet/Mute* considerably.

Next, let's see the efficiency performance of different policies. Figure 9(c) and Figure 10(c) show the total number of hops for file forwarding along the server-client paths to the clients versus the number of requests in simulation and PlanetLab experiments, respectively. We can observe that *Freenet/Mute* has the highest results, *Mantis* has the lowest results, and *TrustAar* lies in the middle. The results imply that *Freenet/Mute* incurs a high forwarding cost, leading to efficiency degradation; *TrustAar* decreases the overhead dramatically, and *Mantis* achieves the highest efficiency in file forwarding. Figure 9(d) and Figure 10(d) show the total number of hops for forwarding files and messages along the server-client paths to clients using our proposed cryptography technique versus the number of requests. *TrustAar* produces the highest number of forwarding hops since it first transmits a message to find a file transmission proxy hop by hop and then transmits a message from the proxy to the server or the previous proxy hop by hop. Then, the file is transmitted from the server or the previous proxy to the identified proxy. However, only around $1/4$ of forwarding hops in *TrustAar* are for file forwarding, which generates longer latency than message forwarding. Figure 11(a) and Figure 11(b) show the total latency of forwarding along the server-client paths to clients versus the number of requests. We used file encryption and decryption for each file transmission in each method. Because file transmission takes long time than message transmission, *Freenet/Mute* produces the highest latency. *TrustAar* significantly reduces the latency of *Freenet/Mute* due to much less file forwarding operations. These experimental results show that *Freenet/Mute* provides high anonymity protection at a high cost, while *Mantis* does not provide sufficient anonymity though it has high efficiency. *TrustAar* achieves a harmonious trade-off between anonymity and efficiency by reducing overhead while maintaining a high level of anonymity for both clients and servers.

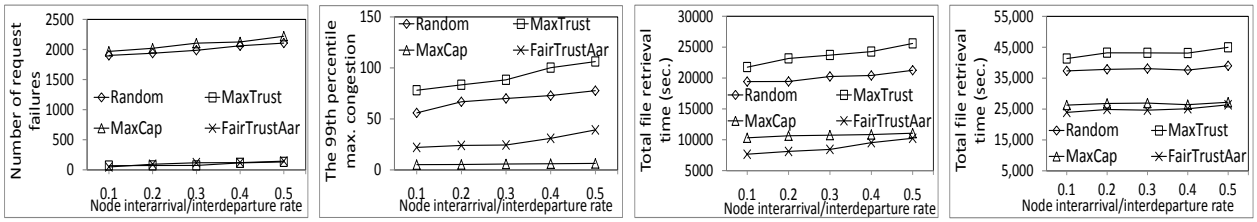
3.3 Performance of the P2P-based infrastructure

It is known that some Internet-based distributed systems such as P2P systems are characterized by dynamism, in which nodes join, leave, and fail continually and rapidly. This experiment aims to evaluate the behavior of the P2P-based infrastructure integrating *FairTrust* and *TrustAar* policies under different levels of dynamism. We use *FairTrustAar* to denote the infrastructure for a harmonious trade-off between trustworthiness and efficiency. We compared the performance of *FairTrustAar* with *Random*, *MaxTrust* and *MaxCap*. These policies use *TrustAar* for response forwarding. In this experiment, the desired trust level was set to 1. There were 3000 queries, and the query rate was modeled by a Poisson process with a rate of 1; one request per second. The node join/departure rate was



(a) Total request failures (b) The 99.9 percentile max. cong. (c) Total file retrieval time without file encryption/decryption (d) Total file retrieval time with file encryption/decryption

Fig. 12: Effectiveness of combination of *FairTrust* and *TrustAar* in dynamism in simulation.



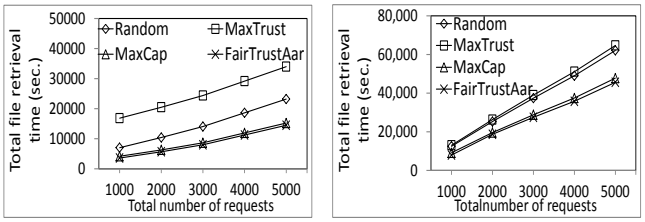
(a) Total request failures (b) The 99 percentile max. cong. (c) Total file retrieval time without file encryption/decryption (d) Total file retrieval time with file encryption/decryption

Fig. 13: Effectiveness of combination of *FairTrust* and *TrustAar* in dynamism on PlanetLab.

also modeled by a Poisson process [33]. We varied the node interarrival/interdeparture rate from 1 to 5 per second with a step increment of 1 in simulations, and varied it from 0.1 to 0.5 per second with a step increment of 0.1 on PlanetLab. A rate of 1 corresponds to one node joining and leaving per second on average. A higher rate corresponds to higher dynamism. Our results are collected from all nodes including the departed nodes and current nodes in the system when all query processing operations complete.

A query failure occurs when it is dropped by a routing node or it cannot be routed due to dynamism. We assume that if a query arrives at its file server successfully, the subsequent time that routing nodes stay in the system is long enough for response forwarding. Therefore, we did not consider the failures in response forwarding. Figure 12(a) and Figure 13(a) show the total number of query failures in simulation and on PlanetLab, respectively. We see that the number of failures increases as the node interarrival/interdeparture rate increases. This is because a faster node join/departure rate leads to higher failures. Another factor that affects the number of failures is trustworthiness. Since *Random* and *MaxCap* do not consider node trustworthiness in routing, they have the highest failures. *MaxTrust* and *FairTrustAar*, which consider node trustworthiness in routing, significantly reduce the number of failures. This implies that the reputation system provides important guidance for node selection based on trustworthiness, and *FairTrustAar* generates a high success rate for file queries.

We then tested the efficiency performance of each policy. Figure 12(b) and Figure 13(b) illustrate the 99.9th percentile and 99th percentile maximum congestion of each policy in simulation and PlanetLab experiments, respectively. The rate of *MaxTrust* still is the highest because it biases on the highest-reputed node. *Random* has a lower congestion rate because it distributes load among nodes by randomized node selection. *MaxCap* achieves the best congestion rate as it considers node available capacity. *FairTrustAar* has a marginally higher rate than *MaxCap* because it chooses the node with the lowest congestion among the trustworthy nodes and offers higher trustworthy performance.



(a) Simulation (b) PlanetLab

Fig. 14: Total file retrieval time without node dynamism.

In this experiment, we set the same resending policy as Figures 7(c) and 8(c). Figure 12(c) and Figure 13(c) show the total file retrieval time of each policy in simulation and PlanetLab experiments without the file decryption and encryption operations, respectively. Figure 12(d) and Figure 13(d) show the total file retrieval time of each policy in simulation and PlanetLab experiments with the file decryption and encryption operations, respectively. We can see that the time grows as the node interarrival/interdeparture rate increases. This is because node departures and failures result in invalid entries in node routing tables. Consequently, detour routings lead to longer routing path lengths, and thus longer transmission latencies. The results show that the latencies of *MaxTrust* are much higher than those of *Random*, and the latencies of *Random* are much higher than those of *FairTrustAar* and *MaxCap*. These observations are consistent with those of Figure 12(b) and Figure 13(b). This implies that *FairTrustAar* and *MaxCap* perform the best in reducing overloaded nodes and query processing even in dynamism. The results also show that the latencies of *FairTrustAar* are slightly lower than those of *MaxCap* because *FairTrust* has a shorter routing time than *MaxCap*, as shown in Figures 7(c) and 8(c). We calculated that the average file retrieval time per request of *FairTrustAar* is approximately 7.9s and 2.93s on PlanetLab with and without file decryption and encryption, respectively, which is acceptable. From the results, we can determine that *FairTrustAar* performs the best with dynamism in term of trustworthiness and efficiency compared to the other policies.

Figure 14(a) and Figure 14(b) show the total file retrieval time without dynamism versus the number of requests in simulation and on PlanetLab, respectively. We see that *FairTrustAar* has the lowest file retrieval latency, followed by *MaxCap*, *Random* and *MaxTrust* due to the same reasons as in Figure 12(d) and Figure 13(d). The results confirm that *FairTrustAar* performs the best in term of trustworthiness and efficiency in a stable environment without dynamism.

4 Conclusions

Most of today's advanced technologies for Internet-based distributed systems gain trustworthiness at the cost of overhead or performance degradation. This paper presents a P2P-based infrastructure that jointly considers trustworthiness and efficiency in node communication in order to meet the high trustworthiness and efficiency requirements of a wealth of diverse distributed applications. The infrastructure includes two policies: a trust/efficiency-oriented request routing policy and a trust-based adaptive anonymous response forwarding policy. Depending upon the reputation system, the infrastructure not only offers a trustworthy environment, but also enhances overall system performance with a harmonious trade-off between trustworthiness and efficiency. It provides a high probability of successful node communication, protects node privacy, and improves node communication efficiency by making full use of each node's capacity while keeping each node's load below its capacity. Simulation and PlanetLab experimental results illustrate the superior performance of the infrastructure compared with other related approaches in both static and dynamic environments, and show the effectiveness of each policy component in the infrastructure. In the future work, we will study the impact of the proposed policies on node behaviors and investigate strategies to combat malicious behaviors that try to take advantage of these policies.

Acknowledgment

This research was supported in part by U.S. NSF grants CNS-1254006, CNS-1249603, CNS-1049947, CNS-1156875, CNS-0917056 and CNS-1057530, CNS-1025652, CNS-0938189, CSR-2008826, CSR-2008827, Microsoft Research Faculty Fellowship 8300751, and Sandia National Laboratories grant 10002282.

References

[1] J. S. Otto, M. A. Snchez, D. R. Choffnes, F. E. Bustamante, and G. Siganos. On Blind Mice and the Elephant-Understanding the Network Impact of a Large Distributed System. In *Proc. of Sigcomm*, 2011.

[2] B. Maggs. A first look at a commercial hybrid content delivery system. http://research.microsoft.com/apps/video/default.aspx?id=154911/ [Accessd in Feb. 2012].

[3] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *TKDE*, 16(7):843–857, 2004.

[4] R. Zhou and K. Hwang. PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. *TPDS*, 2007.

[5] A. Satsiou and L. Tassiulas. Reputation-Based Resource Allocation In P2P Systems of Rational Users. *TPDS*, 2010.

[6] A. Gutowska and K. Buckley. Computing Reputation Metric in Multi-Agent E-Commerce Reputation System. In *Proc. of ICDCS*, 2008.

[7] K. Chen, K. Hwang, and G. Chen. Heuristic Discovery of Role-Based Trust Chains in Peer-to-Peer Networks. *TPDS*, 2009.

[8] L. Lu, J. Han, Y. Liu, L. Hu, J. Huai, L. Ni, and J. Ma. Pseudo Trust: Zero-knowledge Authentication in Anonymous P2Ps. *TPDS*, 2008.

[9] T. G. Papaioannou and G. D. Stamoulis. Achieving Honest Ratings with Reputation-Based Fines in Electronic Markets. In *Proc. of INFOCOM*, 2008.

[10] P. Dewan and P. Dasgupta. P2P Reputation Management Using Distributed Identities and Decentralized Recommendation Chains. *TKDE*, 22(7):1000–1013, 2010.

[11] Z. Li, H. Shen, and K. Sapra. Leveraging Social Networks to Combat Collusion in Reputation Systems for Peer-to-Peer Networks. In *Proc. of IPDPS*, 2011.

[12] H. Shen and G. Liu. An Efficient and Trustworthy Resource Sharing Platform for Collaborative Cloud Computing. *TPDS*, 2013.

[13] H. Shen, Y. Lin, and Z. Li. Refining Reputation to Truly Select High-QoS Servers in Peer-to-Peer Networks. *TPDS*, 2012.

[14] Z. Li and H. Shen. Game-Theoretic Analysis of Cooperation Incentive Strategies in Mobile Ad Hoc Networks. *TPDS*, 2011.

[15] E. Androulaki, S. G. Choi, S. M. Bellovin, and T. Malkin. Reputation Systems for Anonymous Networks. In *Proc. of PETS*, 2008.

[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. of USENIX Security*, 2004.

[17] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[18] The mute file sharing systems. http://mute-net.sourceforge.net/.

[19] T. Isdal, M. Piatek, A. Krishnamurthy, and T. E. Anderson. Privacy-preserving P2P Data Sharing With OneSwarm. In *Proc. of Sigcomm*, pages 111–122, 2010.

[20] H. Shen and Y. Zhu. Fairtrust: Toward secure and high performance p2p networks. In *Proc. of ICPADS*, 2007.

[21] H. Shen. A security/efficiency-optimized infrastructure for wide-area distributed systems. In *Proc. of CIC*, 2008.

[22] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: An Infrastructure for Fault-tolerant Wide-Area Location and Routing. *J-SAC*, 12(1):41–53, 2004.

[23] H. Shen, C. Xu, and G. Chen. Cycloid: A scalable constant-degree P2P overlay network. *Performance Evaluation*, 63(3):195–216, 2006.

[24] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster. To share or not to share: An analysis of incentives to contribute in collaborative file sharing environments. In *Proc. of P2PECON*, June 2003.

[25] G. P. Thanasis and D. S. George. Effective use of reputation in peer-to-peer environments. In *Proc. of GP2PC*, 2004.

[26] H. Shen and C. Xu. Locality-aware and churn-resilient load balancing algorithms in structured peer-to-peer networks. *TPDS*, 2007.

[27] Z. Li and H. Shen. Game-Theoretic Analysis of Cooperation Incentive Strategies in Mobile Ad Hoc Networks. *TMC*, 2011.

[28] K. Tutschku. A measurement-based traffic profile of the edonkey filesharing service. In *Proc. of PAM*, 2004.

[29] S. Bono, C. Soghoian, and F. Monrose. Mantis: A lightweight, server-anonymity preserving, searchable P2P network. Technical report, TR-2004-01-B, Information Security Institute, Johns Hopkins University, 2004.

[30] PlanetLab. http://www.planet-lab.org/.

[31] D. Qiu and R. Srikant. Modeling and performance analysis of Bittorrent-like Peer-to-Peer networks. In *Proc. of SIGCOMM*, 2004.

[32] K. Psounisa, P. M. Fernandezb, B. Prabhakarc, and F. Papadopoulosd. Systems with multiple servers under heavy-tailed workloads. *Performance Evaluation*, 2005.

[33] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *TON*, 2003.

[34] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proc. of NSDI*, 2005.

[35] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proc. of ACM SIGCOMM*, 2003.

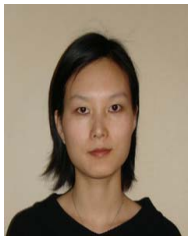
[36] H. Shen and C. Xu. Elastic Routing Table With Provable Performance for Congestion Control in DHT Networks. *TPDS*, 2009.

[37] M. Gunes, U. Sorges, and I. Bouazzi. Ara – the ant-colony based routing algorithm for manets. In *Proc. of IWAHN*, 2002.

[38] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE JSAC*, 16:482–494, 1998.

[39] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untranceability. *Communications of the ACM*, 1988.

[40] J. Han, Y. Liu, L. Xiao, R. Xiao, and L. M. Ni. A mutual anonymous peer-to-peer protocol design. In *Proc. of IPDPS*, 2005.



Haiying Shen Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the ECE Department at Clemson University. Her research interests include distributed computer systems and computer networks with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.

Guoxin Liu Guoxin Liu received the BS degree in BeiHang University 2006, and the MS degree in Institute of Software, Chinese Academy of Sciences 2009. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include distributed networks, with an emphasis on Peer-to-Peer, data center and online social networks. He is a student member of IEEE.

Jill Gemmill Jill Gemmill holds a PhD in Computer and Information Sciences and has over 30 years of experience in building university research computing and infrastructure and support organizations of new academic research support organizations at the University of Alabama at Birmingham and Clemson University. She is currently Research Professor in Clemsons Electrical and Computer Engineering Department. Dr. Gemmill's expertise includes trust relationship management frameworks for federated virtual organizations, network application performance and three-dimensional computer graphics. Her past experience includes university Data Security Officer, network applications specialist, and Technical Director for a Neurobiology Image Reconstruction Facility. Dr. Gemmill has published 33 articles and 2 technical books; and is cited for technical contributions in 16 publications.

Lee Ward Lee Ward is a principal member of technical staff in the scalable systems computing department at Sandia National Laboratories. As an inveterate student of operating systems and file systems, his interests have provided the opportunity to make contributions in high performance, parallel file systems, IO libraries, hierarchical storage management and compute cluster integration/management systems.

5 Pseudocode of Two Algorithms

Algorithm 1: Pseudocode for the *FairTrust* policy.

```
Determine the set of neighbors for the request
routing based on the P2P routing algorithm
//choose trustworthy neighbors with required reputation level
Determine the required reputation level T
Query reputation value (R) of each neighbor
Choose a set of options J = {j_1, j_2 ...} with R > T
Randomly choose b nodes J_b = {j_1, j_2 ... j_b} from J
//choose a node from J_b
Probe each node in J_b for its congestion
if all nodes in J_b are overloaded nodes then
  Select the node with the lowest congestion
else
  if only one node is lightly loaded then
    Select this lightly loaded node
  else //many nodes in J_b are all lightly loaded nodes
    Compute the logical distance from each lightly loaded
    node in J_b to destination respectively
    if all nodes do not have the same logical distance then
      Choose the node with the lowest logical distance
    else
      Acquire the physical distance from each lightly
      loaded node in J_b to destination respectively
      if all nodes do not have the same physical distance then
        Choose the node with shortest physical distance
      else
        Randomly choose a lightly loaded node in J_b
Ask the selected node for routing service
Pay virtual credits to the server for its received service
```

Algorithm 2: Pseudocode for the *TrustAar* policy executed by node i.

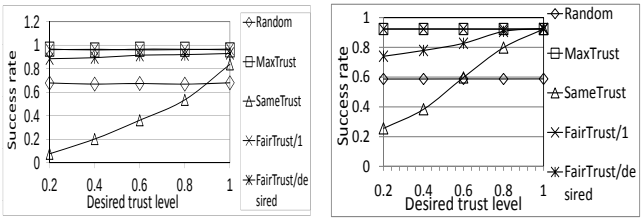
```
/*i_{-1}: the predecessor of node i in the client-server path*/
/*i_{+1}: the successor of node i in the client-server path*/
  if isServer() then {
    Query the reputations of client and itself, {t_c, t_s},
    based on pseudonyms
    Generate a pair of public and private keys (K^s_pub, K^s_pri)
    Send Msg({t_c, t_s}, K^s_pub) to node i_{-1} }

  if receive Msg({t_c, t_s}, K^s_pub) from node i_{+1} then {
    Check the authority of the signature of {t_c, t_s}
    Decide if it becomes a proxy with probability 1/t
    (t = min{t_c, t_s})
      if it becomes a proxy then {
      //notify its previous proxy in the server-client path
      Use K^s_pub to encrypt its IP address: (i)_{K^s_pub}
      Generate a pair of public and private keys (K^p_pub, K^p_pri)
      Send Msg(K^p_pub,(i)_{K^s_pub}) to node i_{+1}}
      //look for its successor proxy in the server-client path
      Generate a pair of public and private keys (K^s_pub, K^s_pri)
    Send Msg({t_c, t_s}, K^s_pub) to node i_{-1}}
      else //it is not a forwarding proxy
        Forward received Msg({t_c, t_s}, K^s_pub) to node i_{-1} }

  if receive Msg(K^p_pub,(j)_{K^s_pub}) from node i_{-1} then
    if it does not have K^s_pri to decrypt the message then
      Forward Msg(K^p_pub,(j)_{K^s_pub}) to node i_{+1}
    else //send the file to the next proxy in the server-client path
      Decrypt (j)_{K^s_pub} using its K^s_pri and retrieve IP address j
      if isNotServer() then
        Decrypt the file using its own K^p_pri
      Encrypt the file using K^p_pub and send it to node j}
```

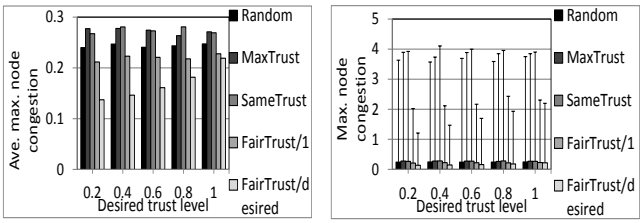
6 Performance Evaluation on the Effectiveness in File Server Selection

We now evaluate the effectiveness of *FairTrust* in file server selection. We assume that a client can locate the five servers successfully. We assume there are five levels of reputation, with the probability of successfully providing service ranging from 0.2 to 1, with a 0.2 increase for each level. Each node is randomly assigned one of the five reputation levels.



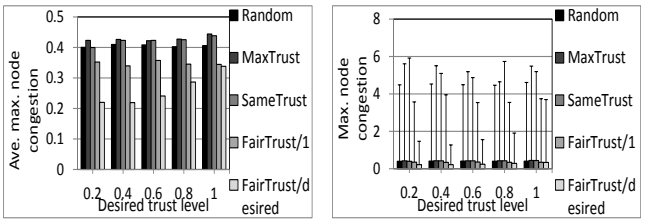
(a) Simulation (b) PlanetLab

Fig. 15: Success rate of service requests.



(a) The average of max. congestions (b) The 99.9th percentile max. cong.

Fig. 16: Node congestion in simulation.

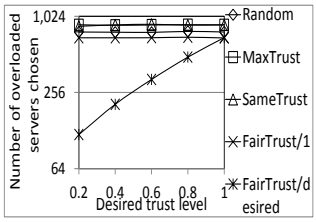


(a) The average of max. cong. (b) The 99th percentile max. cong.

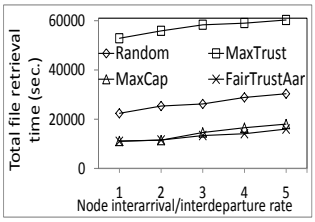
Fig. 17: Node congestion on PlanetLab.

An efficient server selection policy should distribute the load among the servers while considering trustworthiness and efficiency. In this experiment, we study the effect of *FairTrust* in trustworthiness and efficiency compared to other related policies. In the experiment, 1000 different files were requested, with each file requested 10 times. We varied the desired trust level in *FairTrust* from 0.2 to 1, with a 0.2 increase for each step. We also took this rate as the reputation level of requesters in *SameTrust*. We compare the success rate (i.e., the percent of the successfully resolved service requests) between *FairTrust*, *Random*, *MaxTrust*, *FairTrust/1*, and *FairTrust/desired*. *FairTrust/desired* represents the policy with different desired trust levels as set in the experiment. Figure 15(a) and Figure 15(b) plot the success rate versus the desired trust level in the simulation and PlanetLab experiments, respectively. Both figures show that the success rates of *FairTrust/1* and *MaxTrust* are the highest compared to the others, and the rates are close to 1. The rate of *FairTrust/desired* is marginally lower than the rates of *FairTrust/1* and *MaxTrust*, and it increases as the desired trust level increases. The results imply that *FairTrust* and *MaxTrust* achieve highly trustworthy performance. We

expected that the curve of *FairTrust/desired* would be y=x. It is surprising to see that *FairTrust/desired* achieves a much higher success rate than expected. This is because *FairTrust/desired* guarantees the desired trust level first, and then seeks a higher trust level while considering efficiency. *Random* does not consider reputation in node selection, so it has a lower success rate. Unlike the others, the success rate of *SameTrust* increases dramatically with the desired trust level, i.e., the trust level of requesters. Recall that *SameTrust* restricts communication to nodes in the same reputation level. Therefore, in addition to the server reputation, whether or not there exists a server with the same reputation level as the requester also determines the success of a request. For example, if a requester has reputation level 0.2, but there is no server of the requested file within reputation level 0.2, the requester has nowhere to request the file. If there is such a server, the probability of successful file provision is only 0.2. Consequently, the success rate increases as the requester reputation level increases, and is much lower than others due to the constraint of server reputation. The experimental results show the effectiveness of *FairTrust* in trustworthy performance in comparison to the other related policies. Next, we evaluate the performance of each policy with regards to efficiency.

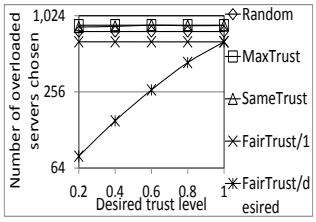


(a) Number of overloaded servers

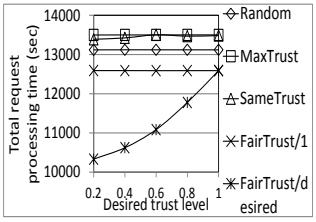


(b) Service request processing time

Fig. 18: Request processing efficiency in simulation.



(a) Number of overloaded servers



(b) Service request processing time

Fig. 19: Request processing efficiency on PlanetLab.

We measured each node's maximum congestion during all test cases and calculated the average and 99th percentile of the maximum node congestions. Figure 16(a) and Figure 17(a) show the average of maximum congestion rate versus desired trust level in simulation and PlanetLab experiments, respectively. We can see that *MaxTrust* and *SameTrust* generate the highest rates. This is within expectations because *MaxTrust* and *SameTrust* strongly bias on the highest-reputed or same reputation level nodes for service provision and those nodes may turn out to be overloaded. The results indicate that *MaxTrust* and *SameTrust* have poor efficiency performance, although *MaxTrust* has high trustworthy performance as illustrated in Figure 15(a) and Figure 15(b). In contrast, *FairTrust/1* and *FairTrust/desired* have much lower congestion rates. Since *FairTrust/1* has the highest reputation level as its desired trust level, how can it achieve a lower congestion rate than *MaxTrust*? This is because *FairTrust/1* distributes load between nodes among the highest-reputed nodes rather than biasing on a single node. On the other hand, with a lower desired trust level, *FairTrust/desired* expands the server candidate pool and achieves a more balanced load distribution, thus achieving higher efficiency.

Figure 16(b) and Figure 17(b) show the 99.9th percentile and 99th percentile maximum congestion of each policy based on the average maximum congestions in simulation and PlanetLab experiments, respectively. We can observe that the rates of *MaxTrust* and *SameTrust* are much higher than the others, and that *FairTrust/1* has lower rates than *MaxTrust* and *SameTrust* due to the same reason as in Figure 16(a) and Figure 17(a). *Random* keeps the rate around 1 in simulation but around 4.5 on PlanetLab since PlanetLab provides fewer servers, so a low-capacity server is more easily to be overloaded. *FairTrust/desired* keeps congestion low, which slightly increases as the desired trust level increases. This is because *FairTrust/desired* expands the server candidates to the servers above the desired reputation level rather than biasing on the highest-reputed node, which increases the possibility of selecting a lightly loaded server. The results confirm the high efficiency performance of *FairTrust*.

We also tested the service request processing efficiency in different server selection policies. Figures 18(a) and (b), and Figures 19(a) and (b) show the number of overloaded servers chosen and service request processing time versus the desired trust level in simulation and PlanetLab experiments, respectively. The figures show that *MaxTrust* and *SameTrust* have significantly higher results than the others. This is due to the side-effect of high trustworthy performance by biasing the highest-reputed or same reputation level nodes. The biasing policy overburdens those nodes, generating more overloaded nodes and longer request processing times. In contrast, *FairTrust* and *Random* have much fewer overloaded servers and processing times. The figures also illustrate that *FairTrust/desired* has lower results than *Random* and *FairTrust/1*, and it has fewer overloaded servers chosen in most cases. *FairTrust/desired* trades surplus trustworthiness for efficiency by mapping tasks of different desired trust levels to nodes with corresponding trustworthiness levels. The results imply the efficiency of *FairTrust* in avoiding overloaded nodes and speeding up service request processing.

The experimental results show that in server selection, *MaxTrust* and *SameTrust* can achieve high trustworthiness, but have difficulty in improving efficiency. *Random* can achieve high efficiency, but does not consider trustworthiness. *FairTrust* has superior performance over the others with both high trustworthiness and efficiency.

7 Related Work

Routing algorithms and reputation systems. In order to improve the routing efficiency in P2P networks, some previously proposed routing algorithms choose the highest-capacity nodes in routing. Castro *et al.* [34] proposed a neighbor selection algorithm to direct most traffic to high capacity nodes. Some algorithms [34–36] forward queries to high capacity nodes in routing to enhance communication efficiency. However, these algorithms cannot guarantee reliable routing because of uncooperative participants.

To provide incentives that promote cooperative (or discourage uncooperative) behavior, many reputation systems [3–15] have been proposed in the last few years. Most works aim to design a reputation calculation method to

more accurately reflect node trustworthiness (e.g., [3]) and (or) realize decentralized reputation management for higher scalability (e.g., [3, 4]).

Though many reputation systems help to achieve accuracy of reputation values, higher scalability, and security of reputation systems, effectively using reputation values is also critical for building a trustworthy environment for distributed systems. Most of the approaches for using reputation metrics are to select the node with the highest reputation as a providing peer. However, this may lead to unexpectedly low efficiency of high-reputed nodes and prevent P2P systems from making full use of system resources. To address this problem, Ranganathan *et al.* proposed a "peer-approved" policy [24] in which nodes can download files only from others with a lower or equal rating. This policy encourages a node to provide high-quality service in order to improve its reputation. However, a node's received quality is questionable, as it may select services from lower reputed-nodes. To handle this problem, Papaioannou [25] proposed a "comparable reputation" policy aiming to restrict nodes to communicate only with others at the same reputation level. Restricting the eligibility of a node to interact with others prevents nodes from sharing resources freely, which is a main goal of P2P systems. We develop the *FairTrust* server selection policy that not only enables nodes to share resources freely, but also ensures trustworthy resource allocation. It avoids overloading high-reputed nodes and takes full advantage of all resources in the system.

Anonymity protocols. Anonymity protocols aim to hide the relationship between an observable action and the identity of the users involved with this action. The Mute [18] and Mantis [29] anonymity systems implement the Ants protocol [37], in which a requester broadcasts a query, and the response is forwarded back along its query route. Tor [16] provides anonymity using the Onion routing protocol [38], in which messages are randomly routed, and each router obtains no information about the message routing path other than the identity of the following router through the use of encryption technology. Freenet [17] is a searchable P2P system which makes it impossible for an attacker to find all copies of a particular file by allowing each node to store all the files that pass across it. Freenet uses response forwarding along its query route to enable provider anonymity, and uses broadcast to obscure the intended recipient. Some systems including DC-nets [39] and Secret-sharing-based mutual anonymity protocol [40] rely on broadcasting to achieve anonymity.

These protocols and systems are effective in building a secure environment with anonymity protection. However, approaches such as broadcasting in Ants, random router selection in Tor, tunnelled communication in Freenet lead to system scalability and efficiency penalties. Motivated by the observation that most current anonymity approaches focus on enforcing security while neglecting efficiency, we aim to explore an approach that considers both.

OneSwarm [19] also aims for a trade-off degree of anonymity for communication efficiency. OneSwarm uses multi-path message forwarding and provides users with explicit configurable control over the amount of trust they place in peers and in the sharing model for their data: the same data can be shared publicly, anonymously, or with access control, with both trusted and untrusted peers. Though sharing similar goals with OneSwarm, our work is different in that it leverages reputation systems to tackle the trade-off problem for providing both high anonymity protection and efficiency.

Our proposed request routing and response forwarding policies are unique in that they provide trustworthy (i.e., reliable and anonymous) communication between nodes while enhancing the efficiency of node communication based on the reputation system.