TrustCode: P2P Reputation-Based Trust Management Using Network Coding

Yingwu Zhu[1] and Haiying Shen[2]

[1] Department of CSSE, Seattle University,
Seattle, WA 98122, USA, zhuy@seattleu.edu
[2] Department of CSCE, University of Arkansas,
Fayetteville, AR 72701, USA, hshen@uark.edu

Abstract. Trust management is very important for participating users to assess trustworthiness of peers and identify misbehaving peers in the open P2P environment. In this paper, we present TrustCode, a framework for P2P reputation-based trust management. Leveraging random network coding, TrustCode spreads coded feedbacks massively among peers, thereby achieving bandwidth-efficient dissemination, ensuring data availability, and yielding efficient feedback retrieval. Our simulations show that TrustCode is resilient to failures and robust against malicious nodes. To exhibit applicability of TrustCode, we also present two applications that can be built on top of TrustCode.

1 Introduction

While P2P systems have become an appealing platform to build a wide range of large-scale distributed applications (e.g., file sharing, content delivery, multimedia streaming), the open and anonymous nature of P2P systems opens the door to possible misuses and abuses of the overlay network by selfish, dishonest and malicious peers. For instance, malicious peers exploit the Gnutella overlay to spread tampered with information such as unauthenticated files and malwares (i.e., Trojan horses and viruses). Thus, reputation-based trust management, which builds trust by utilizing community-based feedbacks about past experiences of peers, has been proposed to suppress peer misbehaving, enabling peers to gauge trustworthiness of others and to selectively interact with more reputable ones.

Challenges for P2P reputation-based trust management include feedback expression, computation of trust, and storage and dissemination of trust data. Prior work [1–3] has well addressed the first two challenges, if not perfectly. In this paper, we focus our work on the last challenge. That is, we aim to provide efficient and robust trust data management in the P2P network where node churn is the norm and malicious peers are present. Current solutions are either centralized or distributed. Centralized solutions like eBay and Amazon, using a central server to store user feedbacks and compute user reputation, while simple, pose the problem of scalability. Prior work [1–3] stores (or aggregates distributed) feedbacks and computes a global trust value for each peer over P2P overlays

like DHTs [4–6], through the notion of *trust manager* or *score manager*. For example, a trust manager for a peer i is the peer node that is responsible for the hash of i's ID. To compute the column trust vector for peers, the trust managers collaborate to aggregate distributed feedbacks and repeatedly compute the trust vector until it converges.

However, the distributed solutions have some limitations: (1) A misbehaving trust manager or peer storing feedbacks could intentionally ignores/drops some feedbacks, resulting in a "distorted" trust vector. (2) Malicious nodes could launch attacks against the trust manager or the node storing the feedbacks for the specific peer, attempting to discredit the peer or simply void the peer's reputation. (3) The number of trust managers is proportional to the number of peers in the large-scale system since the hashes of each peer's IDs are very likely uniformly distributed over the DHT. Consequently, computation of the trust vector incurs high message overhead among trust managers. Worse, if the feedbacks for a peer are stored in a different node other than its trust manager, the feedback retrieval traffic is massive. (4) Node churn could make unavailable the trust manager for a peer or the node storing the peer's feedbacks, resulting in unavailability of trust information and failure of trust computation. Although replication (e.g., by using multiple trust managers or feedback storage nodes for each peer) is able to alleviate the problem, the problem of ensuring data availability in the presence of node churn itself is still nontrivial in the P2P environment [7, 8]. (5) The trust manager or the node storing the feedbacks of a popular peer (which provided a high amount of services to others) may be overburdened, incurring load imbalance.

With these research problems in mind, we propose a P2P reputation-based trust management framework called *TrustCode*, which addresses the issue of dissemination, storage, and retrieval of feedbacks in the P2P network. TrustCode has three main design goals. First, availability of trust data is emphasized because a *complete* set of feedbacks is key to computation of the trust vector that closely characterizes the peers' reputation, while node churn and presence of malicious nodes in the P2P environment are very likely to make some feedbacks unavailable. Second, retrieval of feedbacks for trust computation needs to be efficient by contacting only a small number of nodes if the feedbacks cannot be found locally on the computing node. Finally, the dissemination of feedbacks needs to be efficient and robust against node/link failures and node misbehavior; in the meantime, it needs to ensure data availability and facilitate the retrieval of feedbacks for trust computation.

To meet the goals, TrustCode exploits *random network coding* [9] to disseminate feedbacks over the underlying overlay network. In particular, we make the following contributions: (1) To the best of our knowledge, TrustCode is the first to exploit network coding in P2P trust management. By using network coding, TrustCode makes feedback dissemination bandwidth-efficient and resilient to failures. With massive distribution of coded trust information, TrustCode improves data availability and facilitates retrieval of feedbacks for trust computation. (2) TrustCode is independent of feedback expression, trust computation

models, and overlay structures. It can be easily adapted to any reputation-based system as the trust information management layer. Our simulations show TrustCode has good performance in terms of dissemination speed, bandwidth cost, feedback retrieval, data availability, and failure resilience. (3) We present example applications that can be built on top of TrustCode, exhibiting applicability of TrustCode.

The remainder of the paper is structured as follows. Section 2 provides background on network coding and review of related work. We discuss TrustCode's design and its two applications in Section 3. Section 4 provides experimental setup and results. We conclude the paper in Section 5.

2 Background and Related Work

Background: Network Coding Network coding [10] was first proposed to improve multicast session throughput. It allows an intermediate node to *create* outgoing packets by encoding its received packets instead of simply *repeating* the received packets. With random network coding [9], the encoding generates a new coded packet by the linear combination of the received packets over a Galois field $GF(2^s)$ (s with a typical value of 8 or 16), where coefficients are randomly chosen, and addition and multiplication are performed over the Galois field. Linear combination is not concatenation: the resulting encoded packet is of length of L if it is combined from a set of packets of length L. A receiver which wants to receive all the original packets, needs to receive a sufficient number of coded packets, and then performs Gaussian elimination over a matrix constructed from the coefficients and data blocks contained in each received encoded packet, to decode the original packets. The most compelling benefits of network coding are bandwidth efficiency, data dissemination simplicity, and failure resilience. Due to space constraints, please refer to [11] for more detail on network coding.

Related Work Aberer et al. [12] proposed storing complaints as trust data in a P2P overlay P-Gid, and using replication to handle malicious nodes. In P2Prep [13], each peer aggregates others' opinions about a servant by flooding requests across the network for votes, and computes a reputation value for the servant by considering votes and credibility of the voters. Prior proposals such as EigenTrust [1], PeerTrust [2], and PowerTrust [3], are similar in the sense that they all compute a global trust vector iteratively by taking into account both feedbacks and credibility of feedback sources. Trust data are stored (and replicated) into peer nodes by the use of the underlying DHT data location mechanism. As mentioned earlier, such trust management is vulnerable to node failures and misbehavior. TrustCode differs from the prior proposals in that it exploits random network coding to disseminate coded feedbacks and stores them massively among peers. TrustCode and the prior proposals complement each other. TrustCode can provide a robust trust data management layer for the prior proposals to access the data and compute the trust vector.

3 System Design

3.1 Overview

TrustCode disseminates feedbacks across the overlay network using random network coding, which increases *diversity* of coded feedbacks among nodes and improves resilience to failures during dissemination, while achieving bandwidth efficiency. Leveraging slack storage capacity of peers, TrustCode spreads coded feedbacks massively among peers to provide data availability guarantee in the presence of node churn and malicious nodes; in the meanwhile, TrustCode is very efficient in retrieval of feedbacks for trust computation by contacting only a small number of nodes. While *informed* feedback dissemination is a plausible alternative [3], it requires coordination among nodes by exchanging information to make informed decisions. Leveraging network coding, TrustCode does not require such coordination. Each node independently makes *local* distribution decisions (i.e., simply distributing encoded packets each of which is an arbitrary combination of some packets cached locally) while still achieving fast and efficient feedback dissemination. Not defying the informed dissemination alternative, TrustCode takes a different, complementary approach to resilient and efficient dissemination over the dynamic P2P environment.

TrustCode manages feedbacks in an *epoch*-based manner. An epoch is referred to a time window of length T (e.g., days, weeks or months). The feedbacks generated within an epoch are aggregated for trust computation [4]. Consequently, dissemination, storage and retrieval of feedbacks are all epoch-based. A feedback is locally generated when two peers have done a transaction. For example, if peer A requests a service from peer B. After B provides the service, A will produce and submit a feedback for B regarding the satisfaction of the transaction. In this paper, we term peer A *feedback source* and peer B *feedback target*. A feedback contains a unique ID, epoch, timestamp, feedback source and feedback target. Due to space limitation, please refer to our technical report [14] for details. It is worth pointing out TrustCode is insensitive to feedback expression. Mentioning feedbacks here is only to ease exposition of subsequent subsections

3.2 Packet Structure

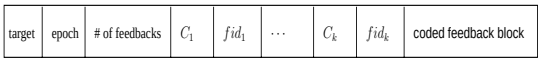


Fig. 1. Packet format.

[3] E.g., nodes exchange Bloom filters that contain their feedback information to make informed feedback distribution, thereby reducing bandwidth consumption over blindly random distribution.

[4] The feedbacks of multiple epochs may also be aggregated for trust computation.

When a peer wants to send feedback data to a neighbor node, the peer uses random network coding to create a packet by *encoding* its cached data which may include original feedbacks (i.e., locally generated feedbacks or decoded feedbacks) and coded feedbacks contained in the received packets. TrustCode makes two restrictions on feedback encoding: (1) Only feedbacks within the *same* epoch can be combined; and (2) Only feedbacks for the *same* feedback target can be combined. Figure 1 shows the packet format. As an example, M is a packet for feedback target B produced by using network coding: $M = \{h(P_B), epoch, k, C_1, fid_1, \cdots, C_k, fid_k, CB\}$, where $h(P_B)$ uniquely identifies the feedback target (P_B is B's public key and $h()$ is a hash function), k is the number of feedbacks encoded in the packet, and C_i is the coefficient randomly chosen from a Galois field of a proper size for the feedback with ID fid_i. *Coded feedback block* CB is a data block encoded from the k feedbacks. Assume all original feedbacks have a same length of L. The coded feedback block also has the length of L due to network coding. Note that the TrustCode packet incurs overhead due to the list of coefficients and feedback IDs encapsulated in the packet header.

How is a packet created from received packets by using network coding? Suppose a peer caches two received packets M_1 and M_2 for feedback target B within epoch e_1: $M1 = \{h(P_B), e_1, 2, C_1, fid_1, C_2, fid_2, CB_1\}$ and $M2 = \{h(P_B), e_1, 2, C_3, fid_3, C_4, fid_4, CB_2\}$. By combining the two packets with randomly chosen coefficients a and b for M_1 and M_2 respectively, the peer produces a packet M from the coded feedbacks contained in the two packets: $M = \{h(P_B), e_1, 4, C'_1, fid_1, C'_2, fid_2, C'_3, fid_3, C'_4, fid_4, CB\}$, where $C'_1 = C_1 \otimes a$, $C'_2 = C_2 \otimes a$, $C'_3 = C_3 \otimes b$, $C'_4 = C_4 \otimes b$, and $CB = (CB_1 \otimes a) \oplus (CB_2 \otimes b)$ ($\oplus$ and $\otimes$ are addition and multiplication operations over a Galois field). CB is the coded feedback block, containing only part of information about the four original feedbacks, which means that CB alone cannot decode any of the four feedbacks. In particular, if a packet M is encoded only from a single original feedback F, then we have: $M = \{h(P_B), e, 1, C, F_c\}$, where $F_c = C \otimes F$. If $C = 1$, then $F_c = F$.

3.3 Local Storage Structure on Peers

Leveraging abundant storage capacity, each peer caches (coded) feedbacks massively to combat node failures and node misbehaving. Each peer maintains two buffers for each epoch [5]: *decoding buffer* and *decoded buffer*. Each entry in both buffers contains feedback data for a particular feedback target. When a peer A receives a packet M, it inserts M into the corresponding entry of the decoding buffer. A then may perform Gaussian elimination on this entry [6]: If there are one or more original feedbacks decoded in this process, the original feedbacks are inserted into the corresponding entry of the decoded buffer. If the cache size

[5] A peer may remove the feedback data in past epochs.

[6] In our implementation, the Gaussian elimination is actually performed on a decoding matrix which is composed of the coefficients and coded feedback blocks contained in the packets in this entry.

for each entry is limited, we may randomly pick a packet in the corresponding entry of the decoding buffer, producing a new packet from the chosen packet and received packet using network coding. Then we replace the chosen packet with the new packet in the decoding buffer. Note that we assume an infinite buffer size on nodes in this paper and leave finite buffer sizes to our future work.

3.4 Dissemination Protocol

When a new epoch starts, TrustCode spreads coded feedback data for the current epoch [7]. For feedbacks in the past epochs, TrustCode allows a peer to retrieve them from other peers. For example, a newly joined peer can not only immediately participate in coded feedback dissemination for the current epoch, but also retrieve coded feedbacks in past epochs it misses.

The dissemination protocol is fully distributed. Each peer independently makes local decisions about what to spread to its neighbors. Moreover, the protocol is an iterative network coding approach. That is, each node further divides an epoch of length T into multiple *time slices* of length t ($t \ll T$). In each time slice, each peer generates a packet encoded from the data in the decoding and decoded buffers for each neighbor and sends the packet to the neighbor. Algorithm 1 outlines the dissemination algorithm on peer x. Note that in Line 7, TrustCode sets a limit on the number of received packets and decoded feedbacks that can be encoded in one packet, in order to limit the number of coefficients and feedback IDs in the packet header and thus the packet overhead. Note that in Algorithm 1, a peer sends each neighbor only one packet for each time slice. In practice, we allow multiple packets (for different feedback targets) to be included in a single message to each neighbor peer, to speed up the dissemination process and reduce message overhead (i.e., TCP/IP header). Once an epoch ends, each peer stops dissemination of the feedbacks for this epoch [8]. However, a peer may inform its upstream neighbors of stopping sending packets if the peer deems it has already cached sufficient data; Or a peer may decides to stop dissemination if it has not received any new (or innovative) feedback data for a certain number of consecutive time slices. Algorithm 2 shows the algorithm of receiving a packet.

During dissemination, TrustCode treats locally generated feedbacks differently. If a peer generates a feedback after a transaction, the peer immediately floods the feedback (in the form of packets but with a typical coefficient of 1) to all its neighbors disregarding the time slice concept. The intuition is that TrustCode makes a few replicas of the feedback to the direct neighbors right away in case that the failures of the feedback source peer extinct the newly created feedback.

[7] We assume the Network Time Protocol(NTP) is used in the overlay network to synchronize clocks.

[8] We may allow a *grace period* for last epoch because the feedbacks generated in the very end of last epoch may not have been spreaded massively among peers.

Algorithm 1 x.disseminate()

1: $targets \leftarrow$ gather all distinct feedback targets from the decoding and decoded buffers for the current epoch
2: **if** $targets$ is empty **then**
3: return
4: **end if**
5: **for** each neighbor node n_i **do**
6: randomly choose a target $k \in targets$
7: generate a packet p for the target k by combining randomly chosen packets and decoded feedbacks in two buffers
8: send p to n_i which in turn calls $receive(p)$
9: **end for**

Algorithm 2 x.receive(Packet p)

1: $e \leftarrow$ extract epoch # from p
2: $k \leftarrow$ extract feedback target from p
3: $CL \leftarrow$ extract the list of coefficients and feedback IDs from p
4: $CB \leftarrow$ extract coded feedback block from p
5: Insert CL and CB into the decoding buffer corresponding to the entry of k for epoch e, which triggers Gaussian elimination
6: **if** any original feedback is decoded **then**
7: insert it into the decoded buffer corresponding to the entry of k for epoch e
8: **end if**

3.5 Feedback Retrieval

TrustCode allows a peer to contact other nodes to retrieve feedbacks for a specific epoch and even a specific feedback target. The contacted nodes serve the retrieval request from their decoding and decoded buffers. Upon receiving the responses, the requesting node inserts them into the decoding buffer, which triggers Gaussian elimination and thus decodes original feedbacks. As shown in our simulations, feedback retrieval in TrustCode is very efficient, contacting only a small number of nodes. Moreover, when a new peer joins the network, it can immediately populate its caches by retrieving coded feedbacks from only a small number of nodes.

3.6 Using TrustCode

Due to space constraints, we here briefly present two potential applications of TrustCode. Please refer to our technical report for details.

The first application is *trust computation using social links or votes*. With TrustCode, a peer can exploits its social relationships such as friend lists to compute the trust vector. Due to the fact that TrustCode stores coded feedbacks massively among the peers and feedback retrieval is very efficient by contacting only a small number of nodes, a small set of friend peers can recover and aggregate all the feedbacks, cooperating in trust computation by playing the role of trust managers in the prior work [1–3]. Leveraging TrustCode and social links, we expect the trust computation is resilient to node failures and misbehavior.

We may also use FoF (friends-of-friends) to do trust computation, splitting the load over more peers. As an alternative, the system may recruit multiple groups of peers to do trust computation. The peers in each group act as trust managers to compute a trust vector (because of TrustCode, each group should be able to recover all the feedbacks). Each group reports its trust vector as a vote. Upon receiving all the votes, we choose a trust vector which is agreed on by the majority.

The second application is *trust monitoring and versioning*. Each epoch in TrustCode represents a version and the trust data in each epoch is a snapshot of system. A trust *collector* node can gather and version the feedbacks of each epoch by contacting only a small number of nodes. A primary purpose of trust versioning is for monitoring and diagnosis. For example, if a peer consistently has a low trust value, the system may evict the peer from the network. If a large percentage of nodes in the network consistently have low trust values, the system is very likely unhealthy and needs to be alarmed. With versioning, the system keeps track of each peer's reputation history, which is important to system monitoring.

4 Evaluation

4.1 Experimental Setup

A 500-node Chord was used as the underlying overlay network. Due to large memory requirement by network coding and packet caching, we did not simulate a larger network size. The Galois field of size is 8 for random network coding. Simulations were limited to one epoch where $5,000$ feedbacks were generated for 50 feedback targets each of which on average had 100 randomly chosen nodes as feedback sources. We ran our simulations in a controlled manner: We stop dissemination of coded feedbacks, if, for *each feedback target*, there are at least $x\%$ nodes each of which has the number of cached packets (including packets in the decoding buffer and decoded feedbacks as special packets in the decoded buffer) that is equal to or larger than a threshold m ($1 \leq m \leq 100$). x and m indicate degree of distribution of coded feedbacks among peers, with a default value of 20 and 30 respectively. The size of a feedback is 312 bytes while the TCP/IP header is counted as 40 bytes for each message during dissemination.

We also simulated random feedback dissemination w/o network coding (called *Random*) for comparison against TrustCode. During dissemination, each message contains up to 5 packets (each of which is encoded from up to 10 randomly chosen packets in the node's buffers) for different feedback targets in TrustCode, while in Random each message contains up to 5 feedbacks for different, randomly chosen targets.

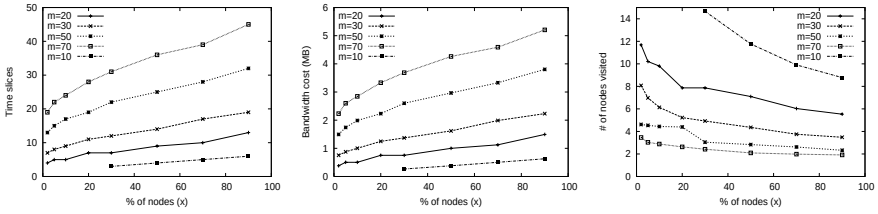
Three main metrics were used: (1) *# of time slices* required to reach the dissemination stop condition. It indicates how fast TrustCode spreads coded feedbacks among peers. (2) *Bandwidth cost per node*. It exposes the overhead of TrustCode in dissemination. (3) *# of nodes contacted* to recover all feedbacks.

It exhibits how efficiently a peer can collect all feedbacks. It also implies level of data availability: If the # of nodes contacted is small, TrustCode provides high data availability guarantee.

After dissemination of coded feedbacks, we scheduled a *collector* node which joins the network, retrieves the feedbacks for the current epoch by contacting a set of randomly chosen nodes till it recovers all the original feedbacks, and finally leaves. 1,000 collector nodes were scheduled in turn to perform the same operations. Then, we averaged the results in terms of *# of nodes contacted* which represents efficiency of feedback retrieval and also implies availability of trust data. If the set of randomly chosen nodes is small, then we believe feedback retrieval is efficient and trust data is highly available in TrustCode.

4.2 Results

We summarize our results before presenting the details: (1) TrustCode shows superior performance over Random which disseminates feedbacks without network coding, in terms of dissemination speed, bandwidth cost, data availability, and feedback retrieval efficiency. (2) The packet overhead in bandwidth due to the list of coefficients and feedback IDs in a packet is big (e.g., about $60 - 68.9\%$ for various values of x and m), rendering room for improvement, e.g., using compression to reduce the overhead. (3) TrustCode exhibits strong resilience to failures in dissemination and feedback retrieval. (4) TrustCode well balances coded feedback distribution among the peers (see Figure 3(c)). Due to space constraints, some data are omitted and please refer to our technical report for more details of the results.

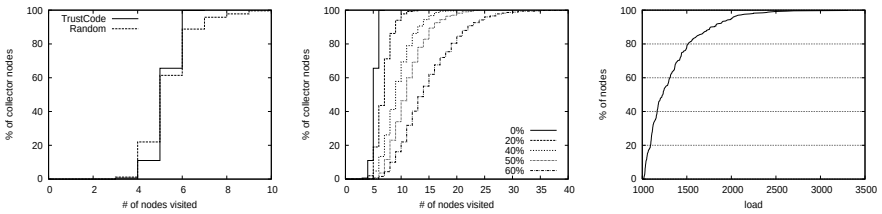


(a) (b) (c)

Fig. 2. Performance of TrustCode for various values of x and m. (a) # of time slices required to reach the dissemination stop condition. (b) Bandwidth cost per node. (c) # of nodes contacted to decode all the original feedbacks.

Performance with Different Configurations The first set of experiments investigate performance of TrustCode under different dissemination stop conditions. Figure 2 shows performance of TrustCode for various values of x and m. We can see that more massive distribution of coded feedbacks (bigger x and

m) makes feedback retrieval more efficient and data more available, but at the expense of more time slices and higher bandwidth consumption. Contacting a small number of randomly chosen nodes is able to recover or decode all original feedbacks. This is very encouraging, especially for the P2P settings where node churn and failures are the norm. When $m = 30$ and $x = 20$ (see Section 4.1 for specifications), a small number 5-6 of arbitrarily chosen nodes are able to recover all the original feedbacks, showing strong resilience to failures and high efficiency in feedback retrieval, while incurring low bandwidth cost in dissemination. The main reason for such high data availability and feedback retrieval efficiency is due to network coding which maximizes diversity of coded feedbacks among peers. Figure 3(a) shows CDF of $1,000$ collector nodes with respect to # of nodes contacted in order to retrieve all the original feedbacks. We can see that at most 7 nodes are needed in order to decode all original feedbacks. In the rest of the paper, we focus on TrustCode with $m = 30$ and $x = 20$ unless specified otherwise.



(a) (b) (c)

Fig. 3. $m = 30$, $x = 20$. (a) CDF of collector nodes. (b) CDF of collector nodes for various percentages of malicious nodes. (c) CDF of nodes with respect to load (defined as the number of packets in two buffers). The mean load of peers is $1,331$ and standard deviation is 318.

Comparison with Random We compared Random against TrustCode ($m = 30$ and $x = 20$) when they have similar level of data availability — that is, a same small number of nodes are able to recover all original feedbacks, thus providing same degree of data availability and retrieval efficiency. Table 1 presents the comparison when *# of nodes contacted* for TrustCode and Random is 5.23 and 5.33 respectively with $1,000$ collector nodes tested. Note that Random needs to replicate feedbacks of each feedback target with $m = 86$ and $x = 55$ in order to yield the similar degree of data availability and retrieval efficiency. This means that each peer in Random needs to devote more storage to cache feedbacks and more bandwidth to disseminate feedbacks. As shown in Table 1, TrustCode achieves one order of magnitude faster dissemination of feedbacks than Random and saves about 73% bandwidth over Random. Figure 3(a) plots CDF of collector nodes for TrustCode and Random. TrustCode has superior

performance over Random because it spreads coded feedbacks more diversely and thus are more resilient to failures.

Table 1. Performance comparison.

	# of nodes contacted	*# of time slices*	*Bandwidth (MB)*	m	x
TrustCode	5.23	11	1.25	30	20
Random	5.33	138	4.59	86	55

Impact of Malicious Nodes In this set of experiments, we explored the impact of malicious nodes on TrustCode. Malicious nodes receive coded feedbacks but do not disseminate any coded feedbacks during dissemination, and they do not respond to feedback retrieval requests (from collector nodes). The parameter x in the dissemination stop condition represents redundancy of coded feedbacks among peers: In the presence of malicious peers, we disregard coded feedbacks in malicious nodes, and x represents redundancy of coded feedbacks for each feedback target among only *benign* peers. Table 2 shows performance of TrustCode with respect to various fractions of malicious nodes. Figure 3(b) plots CDF of $1,000$ collector nodes with respect to different percentages of malicious nodes. TrustCode shows strong resilience to failures in dissemination with modest increase in bandwidth cost, and it is very efficient in feedback retrieval even when a large percentage of nodes are malicious. The driving reason for failure resilience is the diversity of coded feedbacks among peers by using network coding.

Table 2. Impact of malicious nodes ($m = 30$, $x = 20$).

	% of malicious nodes						
Metrics	0	10	20	30	40	50	60
# of time slices	11	12	13	15	17	21	25
bandwidth cost (MB)	1.25	1.35	1.44	1.65	1.85	2.26	2.63
# of nodes contacted	5.23	5.9	6.87	8.13	9.43	11.24	14.79

5 Conclusions

Exploiting network coding and storage capacity on peers, TrustCode provides a framework to manage trust data in P2P networks. Thanks to diversity of coded feedbacks cached among peers, TrustCode exhibits strong resilience to failures and high efficiency in feedback retrieval. Our simulations show that TrustCode is able to distribute coded feedbacks across the network in a fast, failure-resilient, and bandwidth-efficient manner. By contacting a small number of random nodes,

TrustCode is capable of recovering all the feedbacks, ensuring high data availability.

References

1. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the 12th international conference on World Wide Web (WWW)*, (New York, NY, USA), pp. 640–651, 2003.
2. L. Xiong and L. Liu, "PeerTrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, 2004.
3. R. Zhou and K. Hwang, "PowerTrust: A robust and scalable reputation system for trusted peer-to-peer computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 4, pp. 460–473, 2007.
4. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, (San Diego, CA), pp. 149–160, Aug. 2001.
5. A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed System Platforms (Middleware)*, (Heidelberg, Germany), pp. 329–350, Nov. 2001.
6. B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerance wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.
7. B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient replica maintenance for distributed storage systems," in *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2006.
8. A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, (Boston, Massachusetts), May 2005.
9. T. Ho, M. edard, J. Shi, M. Effros, and D. Karger, "On randomized network coding," in *Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing*, Oct 2003.
10. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 46, Jul 2000.
11. C. Fragouli, J.-Y. L. Boudec, and J. Widmer, "Network coding: an instant primer," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 63–68, 2006.
12. K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in *Proceedings of the tenth international conference on Information and knowledge management*, pp. 310–317, 2001.
13. F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "Choosing reputable servents in a P2P network," in *Proceedings of the 11th international conference on World Wide Web*, (New York, NY, USA), pp. 376–386, 2002.
14. Y. Zhu, "TrustCode: P2P reputation-based trust management using network coding," tech. rep., Department of CSSE, Seattle University, June 2007.