THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRA2/11

DCast: Sustaining Collaboration despite Rational Collusion

Haifeng Yu, Phillip B. Gibbons and Chenwei Shi

February 2011

Technical Report

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

DCast: Sustaining Collaboration despite Rational Collusion

Technical Report TRA2/11, School of Computing, National University of Singapore

Haifeng Yu
National Univ. of Singapore

Phillip B. Gibbons
Intel Labs Pittsburgh

Chenwei Shi
National Univ. of Singapore

February 7, 2011

Abstract

A key and well-known problem in large-scale collaborative distributed systems is how to incentivize the rational/selfish users so that they will properly collaborate. Within such context, this paper focuses on designing incentive mechanisms for overlay multicast systems. A key limitation shared by existing proposals on the problem is that they are no longer able to provide proper incentives and thus will collapse when rational users collude or launch sybil attacks.

This work explicitly aims to properly sustain collaboration despite collusion and sybil attacks by rational users. To this end, we propose a new decentralized *DCast* multicast protocol that uses a novel mechanism with *debt-links* and circulating debts. We formally prove that the protocol offers a novel concept of *safety-net guarantee*: A user running the protocol will always obtain a reasonably good utility despite the deviation of *any* number of rational users that potentially collude or launch sybil attacks. Our prototyping as well as simulation demonstrates the feasibility and safety-net guarantee of our design in practice.

1 Introduction

The past decade witnessed the emergence of many large-scale collaborative distributed systems, where the individual rational (selfish) peers are supposed to collaborate. How to incentivize these rational peers to sustain such collaboration is a key and well-known problem [15, 24, 25, 26]. The lack of a proper incentive mechanism can easily lead to a *tragedy of the commons*.

This paper focuses on one particular kind of collaborative distributed system, *overlay multicast*, for its practical importance. A peer of an overlay multicast system is supposed to help forwarding/relaying the multicast data to other peers. Overlay multicast has key applications such as video streaming of sporting events, TV programs, or academic conferences. One way to sidestep the incentive problem in overlay multicast, of course, is to deploy powerful servers with high outgoing bandwidth to directly send the data to each individual peer. Unfortunately, compared to other collaborative systems such as peer-to-peer data backup, multicast is much less amenable to such cloud computing style of solution, due to its excessive bandwidth requirements on the servers. As a result, large-scale commercial multicast systems (e.g., PPLive with millions of users [26]) often still rely on overlay multicast.

While there have been various interesting and practical proposals on incentivizing overlay multicast in particular [11, 17, 18, 22, 26] and various collaborative systems in general [29, 32], these prior solutions all share the following key limitation. Namely, they can no longer properly provide incentives if rational peers collude or launch *sybil attacks* [5].[1] This will again lead to the tragedy of the commons and system collapse in the presence of collusion. For example, collusion can render the punishment mechanism in these approaches completely ineffective, and thus the rational peers will no longer have incentives to collaborate. Such issue can arise even without a large group of colluding peers — as long as each rational peer colludes with a few other peers, these previous designs will fail to provide the desired incentives and collapse (see Section 2).

It is worth noting that collusion in the virtual world is far easier than one might expect at first thought. For example, if someone develops and shares a hacked version of the protocol with many other selfish peers, all these peers are by definition already coordinated and can readily collude. In particular, the colluding peers do not need to know each other beforehand, and the collusion can easily form dynamically *on-the-fly* as the protocol executes.

[1] In this paper, we focus on *rational sybil attacks*, in which it profits a user to create many identities.

Sybil attacks can also be easily launched, for example, by using different ports on the same IP address.[2] By definition, the sybil identities are already colluding.

Challenges in defending against collusion. While the need for dealing with collusion is obvious, collusion introduces the following two fundamental challenges, which helps explain the lack of a solution to date.

First, the key to incentivizing collaboration is always the implementation of a (potentially implicit) punishment mechanism, to punish or reward less those peers who fail to collaborate. The presence of collusion makes it challenging to punish. Evicting a peer is no longer an effective punishment — the evicted peer may obtain multicast data from its colluding peers. In fact, even the eviction process itself becomes tricky: We cannot rely on peers to help disseminate the eviction information, since the evicted peer may "bribe" other peers and form on-the-fly collusion with them, to stall further dissemination. This problem is further complicated by sybil attacks and *whitewashing attacks*, where a user abandons her/his identity to evade punishment and then rejoins with a new identity. While there are some existing sybil defense mechanisms (e.g., [31]), these mechanisms assume collaboration among the peers in the first place.

Second, in some cases the colluding peers may be able to obtain the multicast data from each other more efficiently. For example, suppose the multicast protocol is based on random gossiping, for better robustness against churn. If the colluding peers have low churn, then they can switch to more efficient tree-based multicast to disseminate data among themselves, and participate in gossiping only occasionally. Detection of such deviation is rather difficult, since occasional participation is indistinguishable from legitimate, but slow, participation.

Our results: Safety-net guarantee and the DCast protocol. This work aims to properly sustain collaboration in overlay multicast despite collusion and sybil attacks by rational users. Our first step shows that because performance overhead constitutes part of a peer's utility in overlay multicast, it is *impossible* in practice to prevent deviation by a colluding set of peers. On the other hand, we observe that it is not deviation that is harmful—rather, it is the deviation's negative impact on other (non-deviating) peers that is harmful. This leads to our novel concept of a *safety-net guarantee*, which serves as the formal goal of this work. Intuitively, a protocol offers a *safety-net guarantee* if a peer running the protocol (called a *non-deviator*) can always obtain a reasonably good utility (called the *safety-net utility*), despite deviations by *any* number of rational users who may collude or launch sybil attacks.

While the concept of a safety-net guarantee helps to circumvent the above impossibility, the two fundamental challenges discussed earlier still remain. This paper then proposes a novel *DCast* multicast protocol with the safety-net guarantee. To the best of our knowledge, DCast is the first practical overlay multicast protocol with such a safety-net guarantee. In comparison, in previous designs [11, 17, 18, 22, 26] without the safety-net guarantee, a non-deviator can fail to obtain any multicast data at all from other peers in the presence of collusion, due to the lack of incentives and the resulting tragedy of commons.

DCast also has a number of other salient features. The relative *social cost* (as compared to the multicast cost) incurred by DCast to sustain collaboration is rather small, and tends to zero over time in a large scale system. Second, DCast requires no crypto operations except for basic message authentication and encryption. Finally, DCast is also robust against malicious/byzantine peers. All possible malicious attacks on the utility of the peers have equivalent/similar effects as the attacker directly using its available bandwidth to DoS the peers or the multicast source.

Formally, we prove that DCast offers a safety-net guarantee with a good safety-net utility. We further implement a DCast prototype in Java, as well as a detailed simulator for DCast. Experimental results from running the prototype on Emulab[3] (with 180 peers) and from simulation (with 10,000 peers) confirm the feasibility and safety-net guarantee of our DCast design in practice.

Key techniques in DCast. DCast achieves the safety-net guarantee via the novel design of *debt-links* and *doins*. Debt-links can be viewed as novel *distributed entry fees*, which allow a peer to interact with some specific peers to a limited extent. Doins (shorthand for *debt coins*) are circulating debts and can be viewed as a variant of bankless virtual currency. A doin can be issued by any peer and may circulate only via debt-links. A debt-link from a peer A to a peer B is established by B sending some junk bits to A. A doin *occupies* a debt-link that it passes through, until the doin is paid. After doin payment the debt-link can be reused.

Occupied debt-links serve as an effective punishment, even in the presence of collusion, to a peer that fails to pay for a doin. The doin payment amount is explicitly designed to be strictly larger than the cost of issuing the doin. This serves to ensure that with proper debt-link reuse, the accumulated profit from doin payments will offset and further exceed the cost of accepting a debt-link establishment, making it profitable to accept debt-link establish-

[2]Requiring each user to have a distinct IP address would introduce problems for users behind NATs.

[3]`http://www.emulab.net`

ments and issue doins. Under proper parameters, such profit also conveniently incentivizes the colluding peers, even if they can enjoy a lower cost of disseminating data among themselves.

Summary of contributions. In summary, this paper aims to incentivize overlay multicast in the presence of rational collusion, where prior solutions can no longer properly provide incentives and will thus collapse. We make the following main contributions: i) we motivate and introduce the novel notion of a safety-net guarantee for collaborative distributed systems, ii) we present the novel DCast multicast protocol with debt-links and doins, iii) we formally prove that DCast offers a safety-net guarantee, and iv) we demonstrate via prototyping and simulation the feasibility and safety-net guarantee of the design in practice.

2 Related Work

Concepts related to our safety-net guarantee. Algorithmic mechanism design [6, 23] refers to the general problem of designing algorithms for sustaining collaboration among rational/selfish users. Almost all previous efforts in algorithmic mechanism design aim to develop algorithms that offer no profitable deviations by individuals or, in some cases, by a collusion. Eliminating profitable individual deviation [2, 11, 18] implies that the algorithm corresponds to a Nash equilibrium [23], while eliminating profitable deviation by a collusion corresponds to a collusion-resistant Nash equilibrium [23]. Achieving a collusion-resistant Nash is often quite challenging, and has been done only for a few problems such as secure multi-party computation [1, 14] and service differentiation [19] while assuming accurate global knowledge.

Different from all these approaches, this work explicitly does not focus on equilibrium, and we will explain why such equilibrium is not possible for multicast under rational collusion. Instead, we focus on protecting the utility of peers who do not deviate. Our goal is more closely related to the recent concept of *price of collusion* [10], which quantifies the negative impact of collusion on the overall social utility in a congestion game. Our safety-net notion, in contrast, bounds the negative impact of collusion on the utility of individual non-deviators in a multicast game.

Incentivizing overlay multicast. A number of interesting and practical techniques have been developed for building incentives into overlay multicast to prevent individual deviation [11, 17, 18, 22, 26]. When rational users collude, however, none of these approaches can continue to provide proper incentives. With the resulting tragedy of commons, they will not be able to provide any sort of guarantee (safety-net or otherwise) in those cases.

Specifically in the Contracts system [26], a peer issues *receipts* to those peers who send it data. These receipts serve to testify those peers' contribution. Contracts is rather vulnerable to even just *two* colluding users. For example, a user A wanting the multicast data can trivially obtain fake receipts from a buddy B who does not actually need the multicast data. (In return, A may give fake receipts to B in some other multicast sessions where A does not need the data.) The receipt validation mechanism in Contracts does not help at all here since B does not need the data. If we have another colluding peer C who further gives B fake receipts, then A's contribution will further have a good *effectiveness* (see [26] for definition), which means that A contributes to other contributing peers. BAR gossip [18] and FlightPath [17] rely on evicting peers as an effective punishment. Section 1 already explained that this will not be effective with collusion. The approaches in [11, 22] punish a deviator by not sending it data. Again, such punishment is ineffective since that peer can get the data from its colluding peers. Similarly, all these approaches are vulnerable to rational sybil attacks and whitewashing. Second, in scenarios where the colluding peers can obtain the data from each other more efficiently (e.g., via tree-based multicast), the protocols in [11, 18, 22, 26] can no longer guarantee non-deviators' utility. FlightPath [17] achieves stronger guarantee under the assumption that a peer will not deviate unless the deviation will bring at least ϵ (e.g., 10%) extra utility, which corresponds to an ϵ-Nash [20]. The value of ϵ usually is small since otherwise the ϵ-Nash assumption itself becomes questionable. However, the colluding peers may easily adopt optimizations specific to their own characteristics (e.g., low churn rate) and exceed such 10% threshold. In comparison, the safety-net guarantee of DCast continues to hold even if the utility gain of the colluding peers is much higher (e.g., 100% or 200%).

Finally and mostly for theoretical interest, in the grim trigger approach [16], if a peer does not receive enough multicast data, it conceptually signals the multicast root to shut down the entire multicast session. While this is indeed robust against rational collusion, the approach is far from practical — it is extremely vulnerable to even just a single malicious peer and to various performance instabilities in the system.

Incentivizing other collaborative systems. There have been a lot of efforts on incentivizing p2p backup and p2p file sharing systems. These efforts are largely heuristics and often do not even prevent individual deviation, let

alone dealing with collusion. We will focus on those efforts whose techniques are more related to ours.

Samsara [4] is a p2p backup system that aims to prevent free-riding. When a peer B wishes to back up data on peer A but A has nothing to back up on B, A will force B to store a *storage claim* of the same size as B's backup data. The storage claim may further circulate in the system. Storage claims are fundamentally different from doins in the sense that they cannot decouple the time of resource contribution and the time of resource consumption. Namely, when B consumes a resource, it needs to contribute an equal resource simultaneously. But this resource is of no value to A. The analogue in multicast would be forcing B to send junk bits back to A when getting a multicast block from A. In particular, because A's utility decreases here, A actually has incentive *not* to engage in such an interaction.

Many previous efforts [7, 13, 21] use *credit chains/paths* (including one-hop reputation [25] as length-2 credit chains) to achieve a similar functionality as circulating currency. For example, when a peer A provides service to a peer B, A will be given some *credit* that enables it to later buy service from B. If B further have some credits from C, then A can buy service from C as well (after credit swapping). In these designs, there is simply no mechanism to ensure that credits will be honored. DCast avoids such problem by allowing doins to circulate only on established debt-links, and further freeing the debt-links only after the doin is paid. For the same reason, these designs [7, 13, 21, 25] are usually susceptible to whitewashing. While [13] claims to address sybil attack and whitewashing, the design still requires peers to altruistically help and bootstrap new users, which makes sybil attacks and whitewashing profitable.

KARMA [29] and MARCH [32] use a standard virtual currency system to sustain collaboration, where a peer makes (spends) money when offering (obtaining) service. Their key feature is to assign each peer A some random set of other peers as A's bank to maintain A's balance. Such design is clearly vulnerable to on-the-fly collusion between A and its bank peers.

Virtual currency tied to real currency. In various micropayment schemes [8, 27, 28, 30], the virtual currency is usually tied to real money, and usually is not for incentivizing rational peers. In theory one could use such virtual currency to build incentives into overlay multicast, by requiring a monetary payment for each multicast block propagated. Doing so however not only would cost some users real money, but also would require a payment method from each user.

3 System Model

Basic model. A *user* of the multicast system has one or more identities (i.e., our model allows sybil attacks), and each identity is called a *peer*. Each peer has an IP address[4] and a locally generated public/private key pair. Peers are identified by their public keys, and their IP addresses serve only as a hint to find the peers. We expect that a peer's IP address does not change in the multicast session, so that it can be located. We do not bind the IP address to the public key and we allow IP address spoofing.

The multicast *root* is the source of the multicast data, and the multicast data are in the form of signed *multicast blocks*. The root has a publicly-known IP address and public key, and always follows our protocol. The peers and the root have loosely synchronized clocks with bounded clock errors. We assume that standard crypto primitives, such as encryption and Message Authentication Codes, cannot be broken. We assume that all messages are reliable (e.g., via re-transmission) with bounded delay d.[5] Later we will explain that our DCast design can actually tolerate a small number of message losses and delays beyond d. If needed, processing delay can be easily captured as part of message delay. Messages can be eavesdropped by every peer in the system.

Utility and rationality. A peer is either *rational* (i.e., selfish) or *malicious* (i.e., byzantine). A rational peer aims to maximize its *utility*, which is a function of i) the number of *multicast blocks* received, ii) the number of non-multicast bits received, and iii) the number of bits sent. To unify terminology, we call bits sent by a peer or non-multicast bits received by a peer as *cost bits*. We assume that receiving multicast blocks increases utility, while receiving/sending cost bits decreases utility. We assume that sending one cost bit and receiving one cost bit reduces the utility by the same amount. Generalizing to different sending and receiving costs is trivial in our design. We assume that there exists some constant $\sigma > 1$ such that for most rational peers, sending one multicast block and then receiving (on expectation) σ multicast blocks increase their utilities. As long as this property is satisfied, we allow the utility function to take any specific form. We allow a small fraction of rational peers whose utility functions do not satisfy this property, and we can simply pessimisti-

[4] If needed, in this paper "IP address" should be interpreted as IP address and port.

[5] Strictly speaking, efficient re-transmission may require the receiver to acknowledge the receipt of the message so that unnecessary re-transmission can be avoided. The receiver may or may not have the incentive to send the acknowledgment. But even in such case, we can simply assume that a message is re-transmitted a certain number of times so that the probability of it being received is close to 1.

cally treat them as malicious when needed.

A rational peer is a *non-deviator* if it follows our protocol, otherwise it is a *deviator*. We place no *a priori* limits on the number of deviators; after all, *every* rational peer seeks to maximize its utility and hence may potentially deviate.

Ruling out irrational deviations. We need to properly rule out irrational deviations when reasoning about the behavior of rational peers, since otherwise rational peers are no different from byzantine peers. For example, sending junk bits that are not part of our protocol to a non-deviator is not rational: Doing so hurts the utility of the sender itself, making such a deviation non-profitable and irrational. A deeper look reveals that requiring a profit by itself is still not sufficient to rule out irrational behavior. For example, imagine a profitable deviation which earns the deviator an extra utility of 10. One can modify this deviation so that the deviator spends a utility of 9 to send junk bits to some non-deviator. While doing so obviously is irrational, the modified deviation continues to make a profit of 1.

We thus use the notion of *domination* to rule out these irrational deviations. A *collusion strategy* defines the set of all the deviators, as well as the strategy adopted by each of them. Notice that here for convenience, the set of deviators is specified as part of the collusion strategy. Because the non-deviators by definition follow the protocol, a collusion strategy will uniquely determine the utility achieved by each peer in the system. Consider two collusion strategies α and β for the same set of deviators. We say that α is *dominated by* β if i) each deviator has the same or increased utility under β compared to under α, and ii) at least one deviator has increased utility under β. If there exists some collusion strategy that dominates α, we say that α is *dominated*. Otherwise α is *non-dominated*. One can easily verify that the previous two irrational collusion (deviation) strategies in our examples are dominated by some other strategy that avoids sending the junk bits.

We assume that if α is dominated by β, then the rational deviators will prefer and thus adopt β instead of α, since β offers better utility. Now to reason about the behavior of the rational peers, we only need to consider non-dominated collusion strategies.

4 Safety-net Guarantee

Safety-net guarantee: Motivation. Because the utility function in multicast involves performance overheads (i.e., the overhead of sending/receiving bits), it is impossible in practice to prevent colluding peers from deviating. Namely, unless a protocol can offer optimal performance for each possible subset of the peers (without knowing their specific properties such as low churn rate), some subset can *always* profit by deviating and switching to a more optimized protocol.

To illustrate, let us revisit the example in Section 1. Imagine that the multicast protocol is based on random gossiping for better robustness against churn. If the deviators have low churn rate, they can switch to more efficient tree-based multicast when disseminating data among themselves. Further assume that to avoid being detected, the deviators carefully mimic the original execution (where no one deviates) when interacting with the non-deviators. In such case, the deviators indeed have deviated and achieved a better utility. But such deviation is simply impossible to detect.

Continuing with this example, since the deviators can disseminate data among themselves more efficiently, they may prefer to obtain multicast blocks from each other rather than from the non-deviators. As a result, they may participate in the gossiping protocol less frequently. Detecting such deviation is rather difficult if not impossible, since it is indistinguishable from the deviators simply being slow.

Safety-net guarantee: Definition. Given the above observation, we do not intend to prevent deviation. Rather, we aim to protect the utility of the non-deviators, with the novel notion of *safety-net guarantee*. Safety-net guarantee requires that if a peer A chooses to stick to the protocol and if all peers are rational, then A should obtain a reasonably good utility (called the *safety-net utility*), despite *any* set of peers deviating from the protocol. For the deviators, the safety-net guarantee does not need to protect their utility — if a deviator's utility is below the safety-net utility, it can always switch back to being a non-deviator. Interestingly in the extreme, with safety-net guarantee it is possible for *all* peers to deviate if they can find a better protocol that increases everyone's utility.

The safety-net guarantee provides a lower bound on the utility of the non-deviators: When most peers do not deviate, the non-deviators' utility will likely be higher. One might argue that any loss of utility due to other peers deviating is unfair to the non-deviators. However, it would also be unfair to prevent colluding deviators from benefiting from advantageous factors such as low churn rate among themselves. In some sense, it is the non-deviators who prevent the system as a whole from using a more efficient protocol in such case.

The safety-net guarantee definition by itself does not guarantee the utility for a non-deviator if malicious peers

aim to bring down the utility of that non-deviator.[6] The need to do so is simple: Malicious peers can always send junk bits to a specific non-deviator, and drive down that non-deviator's utility arbitrarily. Dealing with this kind of targeted DoS attack is clearly beyond our scope, and thus the safety-net guarantee definition does not intend to capture that.

Finally, the safety-net guarantee does not explicitly capture the utility of the root. Clearly, a design should not achieve safety-net guarantee by requiring the root to directly send blocks to each peer. In DCast, the number of multicast blocks directly sent from the root will be no more than in a standard gossip-based multicast system where all peers cooperate.

5 Design Overview and Intuition

5.1 DCast Design Overview

Basic multicast design. Similar to several recent efforts [11, 17, 18] on incentivizing overlay multicast, we use gossiping to disseminate multicast blocks for its simplicity and its robustness against churn. Specifically, we use pull-based gossiping where in each *round* (see later), each peer pulls data from another random peer. DCast allows new peers to dynamically join and leave the multicast session. A peer contacts the root to join the multicast session, and the root maintains a list of the IP addresses of all peers. This list is obviously subject to sybil attack and we will address that later. The root allows peers to periodically (e.g., every 10 minutes) request a random *view*, which is a small random set of IP addresses in the root's list. A peer uses such information to find and gossip with other peers. Notice that a rational peer does not have incentive to repeatedly request a view since it consumes the peer's bandwidth as well.

The multicast session is divided into *intervals*, where each interval has a fixed number of synchronous gossiping *rounds* (e.g., 30 rounds of 2 seconds long each). In each round, the root sends signed erasure-coded blocks to as many randomly selected peers in the list as the root's (limited) bandwidth can support. Before the root sends a block to a peer, the root will request that peer to send $\mathcal{D}_{\text{root}}$ (e.g., 3) *junk blocks* to the root, where each junk block is of the same size as a multicast block. To avoid delay, the sending of the junk blocks can be done before the round starts.

[6] Note however that DCast offers additional properties beyond the safety-net guarantee, and DCast is robust against malicious users (see discussion in Section 5.3).

After obtaining a sufficient number of blocks for a given round, a peer can decode the video frames for that round and then render the video. If a peer fails to obtain enough blocks before a certain deadline (e.g., 20 rounds after the multicast blocks were sent by the root), it will consider the frames as lost.

Debt-links and Doins. In DCast, except during doin payment (discussed later), the propagation of a multicast block from one peer A to another peer B is always coupled with the propagation of a *doin* on a free *debt-link* from A to B. A *debt-link* from A to B is established by B sending $\mathcal{D}_{\text{link}}$ (e.g., 2.5) junk blocks to A. Notice that establishing the debt-link hurts the utility of both A and B. B may establish multiple debt-links from A if needed, and it is also possible to simultaneously have debt-links in the reverse direction from B to A. A debt-link is *free* when first established. After propagating a doin via that debt-link, the debt-link becomes *occupied* until the corresponding doin is *paid* which will free the debt-link again.

A doin, which is a shorthand for *debt coin*, is a debt and can be *issued* by any peer. The current holder of a doin conceptually "owes" the issuer of the doin. Doins may circulate in the system and thus can be viewed as a special kind of bankless virtual currency. Namely, when B pulls a multicast block from A, A may either *relay* a doin that A currently holds or A may issue a new doin. Each doin corresponds to one multicast block in our design, though it is trivial to make it more general.

All doins issued within one interval *expire* at the beginning of the next interval, and new doins will be issued in the next interval. Peers holding these expired doins will then pay for these doins. Expiring all these doins at the same time is important, because otherwise peers may prefer to accept, for example, doins with larger residual lifetime. To pay for a doin, a peer needs to send the doin issuer $\mathcal{D}_{\text{pay}}$ (e.g., 2) multicast blocks. After payment, all debt-links that the doin has traversed and thus occupies will be freed. Notice that since doins allow a peer to pay back its debt in the next interval, bootstrapping of new peers is trivial.

5.2 Intuition behind DCast

We will focus on intuition in this section and leave the formal claims and proofs to Section 7. For explaining the intuition, we will assume that there is no control message overhead (i.e., a bit sent/received is either a bit in some multicast block or a bit in some junk block). The parameters in DCast should satisfy $1 < \mathcal{D}_{\text{pay}} < \mathcal{D}_{\text{link}} < \mathcal{D}_{\text{root}}$.

Roughly speaking, the safety-net utility offered by DCast is such that i) a non-deviator will receive all multi-

cast blocks needed,[7] and ii) a non-deviator needs to send up to $\Psi > 1$ (e.g., $\Psi = 2$) cost bits for each multicast bit received. Because peers in overlay multicast usually care much more about receiving all the data than the overhead of sending data, we expect such utility to be sufficiently good as a safety-net utility. Recall that in comparison, in previous designs [11, 17, 18, 22, 26] without the safety-net guarantee, a non-deviator can fail to obtain any multicast data at all from other peers in the presence of collusion, due to the lack of incentives and the resulting tragedy of commons. The following intuitively explains how this safety-net guarantee is achieved.

Incentivize doin payment via debt-link establishment cost. The first key mechanism in DCast is an effective punishment (despite collusion) against peers who do not pay for doins. As explained in Section 1, evicting a peer by itself is not effective in the presence of collusion. In DCast, the debt-link establishment cost serves as an effective punishment. For the following discussion, we will assume that there are *infinite* number of intervals, and we will explain how this assumption can be weakened later.

Doins in DCast may only circulate on debt-links and a debt-link cannot be reused until the corresponding doin is paid. A peer B establishes debt-links from other peers purely for the purpose of pulling blocks from them. Because establishing a debt-link incurs overhead on B and B has a choice over whether or not to establish a debt-link, a rational B will establish a debt-link only if the debt-link's existence is necessary to maintain its utility. If B does not pay for a doin, then the corresponding debt-link will not be freed, and B may need to establish another debt-link to compensate. Establishing one more debt-link is less desirable than paying for the doin since $\mathcal{D}_{\text{link}} > \mathcal{D}_{\text{pay}}$.

Such argument holds for colluding peers as well. A colluding peer can obtain multicast blocks from other colluding peers. But if the colluding peer does establish an incoming debt-link from a non-deviator, it indicates that the peer is not able to rely on other colluding peers only, and has to pull blocks from some non-deviators. Similarly, consider a whitewashing attack where a peer tries to evade doin payment by abandoning its old identity and then rejoining with a new identity. Such attack is not profitable because the new identity will need to establish new debt-links again.

Incentivize doin issuance via debt-link reuse. Establishing a debt link from A to B involves B sending junk blocks to A. Here using junk blocks instead of multicast blocks is important since i) otherwise A may simply attract debt-link establishments without later issuing/propagating doins, and ii) a new peer B may not have any multicast blocks to offer.

[7] Strictly speaking, even with fully cooperative peers, random gossiping may still fail to achieve such a property, with some vanishingly small probability. But to focus on the incentive mechanism and for clarity, our discussion will ignore this vanishingly small probability. It is trivial to fully qualify our discussion by adding a "with high probability" condition.

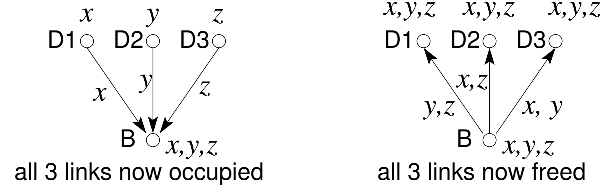

Figure 1: A non-deviator B pulls three blocks (x, y, and z), together with three doins, from three deviators D_1, D_2, and D_3. B then uses these blocks to pay off the exact three doins accompanying the three blocks ($\mathcal{D}_{\text{pay}}$=2). The arrows in the figure are messages. The debt-links from D_1, D_2, and D_3 to B are not shown in the figure.

However, using junk blocks also makes it unclear why A has incentive to accept such link establishment and issue a doin. Namely since $\mathcal{D}_{\text{link}} > \mathcal{D}_{\text{pay}}$, A can still suffer a loss even after a (single) doin payment. The key to incentivizing A here is debt-link reuse. Namely, reusing the debt-link enables the net profit from doin payments (since $\mathcal{D}_{\text{pay}} > 1$) to eventually offset and exceed the initial one-time setup cost of the debt-link. Further notice that B has the incentive to reuse the debt-link as well, instead of establishing other new debt-links. Thus maximizing debt-link reuse is in the common interest of A and B.

Conveniently, similar incentives for doin issuance apply to colluding peers as well. Although in some cases they may be able to obtain multicast blocks from each other with lower cost, as long as this cost is not zero, we can always set $\mathcal{D}_{\text{pay}}$ high enough so that the profit of issuing doins is attractive. In practice, we expect that setting $\mathcal{D}_{\text{pay}}$ to be 2 or 3 is sufficient to incentivize a large fraction of the deviators. We need not incentivize every deviator—random gossiping naturally takes care of a small number of peers that do not collaborate.

DCast does not allow or provide any mechanism for peers to negotiate $\mathcal{D}_{\text{pay}}$, for two reasons. First, we do not believe that it is practical for users to negotiate the payment price. Second, setting a fixed price precludes the possibility of monopoly. Namely, because the non-deviators will only make payments by the system-set price, the deviators cannot use their possible monopoly to raise the price.

Finally, having $\mathcal{D}_{\text{pay}} > 1$ also incentivizes peers to relay doins: it is less costly to relay a doin than to be stuck paying for it.

Ability of paying the debts. So far we have intuitively

explained that a peer has the incentive to pay for a doin, assuming infinite number of intervals. For the payment to actually occur, the peer needs to have i) enough multicast blocks to offer to the doin issuer, and ii) enough bandwidth to send those blocks.

It may not be immediately clear why peers will have enough blocks to pay off their high-interest debts. If we view each block as a dollar, then every dollar must be repaid with multiple dollars. The "central bank" (i.e., the root) only injects a small number of dollars into the system. It then seems that the whole "economy" cannot possibly sustain. Fortunately, our situation is fundamentally different from a real economy in the sense that a peer B can use the same block to offset its debts from *multiple* peers, by sending that block to each of them. This, in turn, makes it possible for B to purely rely on blocks pulled from deviators to pay off the debts to those deviators, despite that pulling the blocks will cause B to take more debts. For example in Figure 1, B starts without any blocks, and then pulls 3 blocks (x, y, and z) from 3 deviators (D_1, D_2, and D_3), respectively. Afterwards, B can use these 3 blocks to fully pay off the debts to the three deviators.

The example also helps to illustrate what will happen if B does not have enough (upload) bandwidth to pay off the debts. Conveniently, if B does not have bandwidth to pay, then B will not have bandwidth to establish new debt-links either. In fact, once B has available bandwidth, it will prefer paying off the old debts instead of establishing new debt-links. This avoids the undesirable situation where a peer without enough bandwidth just keeps borrowing new debts without paying off the old debts.

Rational sybil attacks. All our reasoning above applies to both colluding non-sybil peers and colluding sybil peers, except for the discussion regarding the peer list maintained by the root. Launching a sybil attack will enable one rational user to occupy a large fraction of the peer list, which has two effects. First, the sybil peers will be selected more often for receiving blocks directly from the root. But since $\mathcal{D}_{\text{root}} > \mathcal{D}_{\text{link}}$, directly receiving the blocks from the root even exceeds the cost of pulling them from other peers. Thus the user has no incentive to create sybil identities to attract more blocks from the root.

The second effect is that the sybil peers will more likely be in the view of other peers. If a peer A is in the view of a peer B, it just means that B may establish debt-links from A. By our design, debt-link establishment by itself actually hurts A. Rather, A makes a profit only when A propagates doins to B and when the doins are paid. Thus a rational user only has the incentive to create as many sybil peers as they have enough bandwidth to issue/relay doins and propagate multicast blocks. As long as the sybil peers do this, they will not have any negative impact other than increasing the system size, which may in turn just slightly increase the logarithmic number of rounds needed for gossiping to finish.

Small social cost. A nice property of our DCast design is its rather small *social cost*. In mechanism design, *social cost* refers to the extra overhead incurred in order to incentivize the rational peers, as compared to the scenario where all peers are cooperative. DCast incurs social cost only when peers send junk blocks to establish debt-links and when peers send junk blocks to the root. As our later experiments will show, the number of debt-links that a peer establishes converges rather quickly over time. This means that the average number of debt-links established per round approaches zero over time. Second, since the root only directly sends multicast blocks to a rather small number of peers, the total number of junk blocks involved in this process is rather small as well, compared to the total number of multicast blocks propagated among all the peers.

5.3 Additional Issues

End-game effect due to finite number of intervals. So far we have been assuming infinite number of intervals, which serves to avoid the well-known *end-game effect*. For example, if the peers know that there are exactly 10 intervals, then no peer will issue doins in the 10th interval since the doins will not be paid back. In turn, no peer will have incentive to pay off the doins issued in the 9th interval and free the debt-links. Backward induction will cause such argument to cascade back to all the intervals. This end-game effect is quite fundamental and applies to all previous proposals [11, 16, 17, 18, 22, 26] on incentivizing overlay multicast, as well as to all kinds of repeated games such as the iterated prisoner's dilemma.

On the other hand, this effect is widely considered [20] to be an artifact of modeling instead of a good prediction of how rational peers will behave. There are many well-known ways [20] to avoid this effect, based on what behavior one would expect from rational peers. For example, it also suffices (as in [11]) to just assume that at each interval, the peers *expect* (which may or may not correspond to what actually happens) that with probability at least p_0, there will be at least one more interval. Alternatively, one can invoke the ϵ-Nash concept and assume that the extra small utility obtained by not issuing doins in the very last interval does not give sufficient incentive for the peers to deviate [20]. Since this end-game effect and the ways to overcome it are largely orthogonal to DCast, we

will not discuss this further in this paper. Instead, to focus on DCast, we will continue assuming infinite number of rounds.

Unexpected events such as message losses. For clarity, our model assumed away various unexpected events such as message losses, excessive message propagation delays, peer failures, etc. As long as these events happen with small enough probability so that the gaps among 1, $\mathcal{D}_{\text{pay}}$, $\mathcal{D}_{\text{link}}$, and $\mathcal{D}_{\text{root}}$ are preserved, the incentives in DCast continue to hold. For example, if a peer fails while holding a doin, then the doin will never be paid. However, as long as such probability p satisfies $1 < (1-p)\mathcal{D}_{\text{pay}}$, issuing and relaying doins will still be profitable. Similarly, imagine a scenario where a peer makes a doin payment but due to message loss, the payment is not actually received by the doin issuer. In this case, the payment is "wasted" and the debt-link will not be freed. Furthermore, the doin issuer and the doin holder now have an inconsistent view regarding whether the payment has been made. However, as long as the message loss probability p satisfies $\mathcal{D}_{\text{pay}} < (1-p)\mathcal{D}_{\text{link}}$, a rational peer will still prefer making a payment (and hope that the message will not be lost) over establishing a new debt-link.

Malicious peers. As explained in Section 4, the safety-net guarantee intentionally does not aim to prevent malicious peers from bringing down the utility of a non-deviator. Nevertheless, similar to the concept of the BAR model [18][8] and t-immune equilibrium [1], we still want to ensure that at least our DCast design does not introduce any extra vulnerability. For example, we should avoid grim trigger designs [16] where the whole system can "melt down" due to a single malicious user sending a single control message. Due to space limitations, we defer a detailed discussion on malicious peers to Appendix D. As a concise summary, all possible malicious attacks on the utility of the non-deviators in DCast have equivalent/similar effects as the malicious users directly using their bandwidth to DoS the peers or the root. Furthermore, the attacker cannot significantly amplify its DoS capacity by exploiting our DCast design, as compared to a direct DoS attack on the peers or the root.

5.4 Alternative Designs and Discussion

Before ending this section, we explore some alternative designs, and intuitively explain why designs that can deal with collusion and sybil/whitewashing attacks will tend to share the key features of our DCast design. As discussed in Section 1, the key in all these designs is to implement an effective punishment mechanism despite sybil/whitewashing attacks and collusion. (If the colluders have more efficient ways to disseminate data among themselves, this raises an additional complication designs must handle—this will be addressed at the end of the section.)

Entry fee as effective punishment. Fundamentally, there are only a few effective punishments in the presence of sybil/whitewashing attacks, and perhaps the most natural one is to impose an entry fee on each new peer. With proper entry fee, evicting a peer (if we can) will constitute an effective punishment despite collusion.

The key question, however, is how to design this entry fee and how to evict a peer when needed. Because overlay multicast needs peers to contribute bandwidth, the entry fee must be in the form of bandwidth consumption. For example, if we instead use computational puzzles as entry fees, then certain users with ample computational resources may still prefer whitewashing over contributing bandwidth.

Using bandwidth consumption as the entry fee turns out to be tricky. Directly paying this entry fee to the root would overwhelm the root and thus is infeasible. Paying this entry fee to individual peers is possible, but colluding peers may simply accept fake entry fees from each other and vouch for each other. Furthermore, this entry fee must be in the form of sending junk data because a new peer has no useful data to send. This means that the individual peers have no incentive to accept this entry fee. Finally, evicting a peer is also tricky under collusion due to the difficulty of disseminating the eviction information to all peers, because the colluding peers may interfere and stall such dissemination.

Distributed entry fees. Fundamentally, the debt-links established by a peer in DCast are *distributed entry fees* paid by that peer. A traditional non-distributed entry fee is system-wide in the sense that it entitles the peer to interact with any peer to any extent, which introduces all the problems described earlier. Distributed entry fees only entitle a peer to interact with some specific peers (i.e., those peers from which the debt-links are established) and in a limited form (i.e., cannot borrow more blocks than the number of free debt-links). These distributed entry fees enable individual peers to *unilaterally* "accept" or "evict" a peer, overcoming some of the earlier difficulties. Such a design seems to be necessary if one wants to use bandwidth consumption as an entry fee. The use of distributed entry fees in turn, further makes it necessary for doin payment to be strictly larger than the cost of issuing doins. Otherwise peers will simply not have the incentive to accept the entry fee (i.e., to establish debt-links).

[8] Note that because safety-net guarantee does not aim for an equilibrium, it escapes the impossibility result from [3].

Further incentivizing colluding peers if needed. Finally, in some cases the colluding peers may be able to obtain multicast blocks from each other more efficiently. Conveniently, having large doin payment also naturally motivates even those peers to issue doins, addressing this complication.

6 DCast Protocol

This section elaborates on DCast's protocol of debt-link establishment, doin propagation, doin payment, and debt-link freeing. Similar to traditional virtual currency systems, while the concept of debt-links and doins is not hard to understand, implementing such a concept in the presence of deviators is far from trivial. In particular, a single conceptual procedure (e.g., paying for doins) involves multiple protocol steps, and we need to carefully build incentive into each step.

We will focus on intuition in this section and leave the formal claims and proofs to Section 7. For space limitations, we defer to Appendix A the *simultaneous exchange* optimization, which allows two willing peers to exchange an equal number of multicast blocks without eventually occupying any debt-links. We will also assume zero clock error here due to space limitations. Bounded clock errors can be easily addressed by accounting for possible clock drift, as described in Appendix B.

6.1 Message Encryption and Authentication

In DCast, every message from a peer A to another peer B is encrypted and authenticated via a MAC using a symmetric session key between A and B. Except for this, DCast does not need any other crypto operations. Notice that this is a sharp contrast to typical designs involving virtual currency (e.g., [30]). Because each peer in DCast has a locally generated public/private key pair, it is trivial to set up the symmetric session keys. A peer B will initiate communication with another peer A for i) paying for doins issued by A, ii) establishing or freeing debt-links from A, or iii) pulling multicast blocks from A. For paying for doins, B can obtain information about A's public key in the doin's id (explained later). For establishing debt-links, B will contact some IP addresses in its (random) view to obtain those peers' public keys. Notice that B is not concerned about obtaining the public key of some specific peers—rather, all it needs is to obtain the public key of some random peers. Thus it does not matter if the IP address is spoofed or the message is intercepted—that would just be equivalent to polluting B's view via a sybil attack, which we already discussed in Section 5.2. Finally, for freeing debt-links or pulling multicast blocks from A, B must have previously established debt-links from A and hence must already have a session key with A.

6.2 Establishing Debt-links

To establish a debt-link from another peer A, a peer B simply sends A $\mathcal{D}_{\text{link}}$ junk blocks. No reply from A is needed. The ith debt-link from A to B has a *debt-link id* of i, known by both A and B. How many debt-links to establish from which peers is largely a performance issue, and DCast uses the following heuristic. During each of the first few rounds (e.g., 10), each peer B selects a distinct peer in its view and establishes a certain number (e.g., 20 to 80, depending on the bandwidth available in that round) of debt-links from that peer. Afterwards, B monitors the number of multicast blocks that it has received in recent rounds, and establishes additional debt-links if that number is low.

6.3 Pulling Blocks and Propagating Doins

Algorithm 1 presents concise pseudo-code for pulling blocks, sending blocks with doins, paying for doins and freeing debt-links. In each round, a peer B selects some random peer A that has debt-links to B, and pulls multicast blocks from A. To do so, B first sends to A a *summary* describing the multicast blocks that B already has (Step 2). A then sends back those blocks that A has but B does not have, up to the number of free debt-links from A to B (Step 9). For each block sent, A also includes the id (explained below) of some doin and the id of some free debt-link. During circulation, a doin may propagate from A to B multiple times, in different interactions. The debt-link id in A's reply thus helps to distinguish different interactions between A and B.

A doin's *id* includes the IP address and the one-way hash of the public key of the doin issuer, a distinct *sequence number* assigned by the doin issuer, as well as the *issuing interval* indicating the interval during which the doin was issued. The total size of the doin id is less than 40 bytes. The one-way hash function is publicly-known, and the hash enables the authentication of the doin issuer. Alternatively, one could directly include the public key itself, but a hash is shorter.

Salient protocol features enabled by debt-links and doins. It is worth noting that using debt-links and circulating debts makes possible several salient features in the above protocol, as compared to traditional virtual cur-

Algorithm 1 DCast routines for pulling blocks, sending blocks with doins, paying for doins, and freeing debt-links. All messages are encrypted and authenticated (not shown). Meaning of acronyms: DSN = doin sequence number (part of doin id); DII = doin issuing interval (part of doin id); RLI = release local id.

1: **/* Pulling multicast blocks */**
2: Send ⟨"data-request", summary⟩ to a random peer that has at least one free debt-link to me;
3: Wait for ⟨"data-reply", (block, doin id, debt-link id) tuples⟩;
4: **/* Sending multicast blocks and doins */**
5: Upon receiving ⟨"data-request", summary⟩ from B:
6: S = set of blocks that I have and are not in summary;
7: $k = \max(|S|$, number of free debt-links to B);
8: **if** I hold less than k doins, **then** issue new doins;
9: Send back ⟨"data-reply", k tuples of (block, doin id, debt-link id)⟩;
10: **/* Paying for a doin */**
11: **for** each expired doin that I hold **do**
12: Send ⟨"pay-request", DSN, DII, summary⟩ to doin issuer;
13: Wait for ⟨"bill", DSN, DII, blocks needed⟩;
14: Send back ⟨"payment", DSN, DII, $\mathcal{D}_{\text{pay}}$ blocks⟩;
15: Wait for time d;
16: Send ⟨"release", debt-link id, RLI⟩ to predecessor;
17: **end for**
18: **/* Accepting a doin payment */**
19: Upon receiving ⟨"pay-request", DSN, DII, summary⟩:
20: **if** I can find in the summary at least $\mathcal{D}_{\text{pay}}$ blocks that I need **then**
21: Send back ⟨"bill", DSN, DII, blocks needed⟩;
22: Wait for ⟨"payment", DSN, DII, $\mathcal{D}_{\text{pay}}$ blocks⟩;
23: Mark the doin (DSN,DII) as paid;
24: **end if**
25: Upon receiving ⟨"release", debt-link id, RLI⟩ and if I am the issuer of the corresponding doin:
26: **if** the doin has been paid, **then** free debt-link;
27: **else** send back ⟨"denial", debt-link id, RLI⟩;
28: **/* For a peer who previously relayed the doin */**
29: Upon receiving ⟨"release", debt-link id, RLI⟩:
30: Relay release to predecessor (with proper debt-link id and RLI);
31: Set a timer of $2d \cdot$ (doin's r-stamp) for this release;
32: **if** no denial within the timeout, **then** free debt-link;
33: Upon receiving ⟨"denial", debt-link id, RLI⟩:
34: **if** the corresponding release has not timed out **then**
35: Invalidate the release (i.e., will not free debt-link);
36: Relay denial to successor (with proper debt-link id and RLI);
37: **end if**

rency systems. First, the only difference between the above protocol and plain pull-based gossiping is the piggybacking of debt-link ids and doin ids in A's reply. This 2-message interaction is substantially simpler and more efficient than typical interactions involving virtual currency. In particular, there is no need for A to obtain any signature from B to acknowledge the acceptance of the debt. Once A sends the message (and since messages are reliable[9]), A knows that those doins have been propagated to B. It does not matter if B denies the receipt of the debt — if B does not pay for the doins later, then A simply will not free the corresponding debt-links. In fact, even if A obtained a signature from B, such signature would be of no use: There is no arbitrator in the system that can arbitrate the interaction and punish B.

Second, the double-spending problem with virtual currency is completely avoided — a rational peer will prefer to issue a new debt instead of relaying the same debt to two different peers. Third, there is no need to protect the doin id from manipulation and no need to verify the doin id. Modifying any field in the doin id will be equivalent to "issuing" a new doin and not propagating the current doin. Of course, this also means that a deviator may "issue" doins on other peer's behalf by putting that peer as the doin issuer. Our proof later will show that a deviator has no incentive to do so, again because a doin is a debt.

6.4 Making Payments / Freeing Debt-links: One-hop Doins

If a doin only circulated for one-hop before being paid, then the payment process would be simple. Namely, the current holder B of the doin will send to the doin issuer A the doin sequence number and issuing interval, as well as a summary describing the multicast blocks that B has (Step 12). If A can find $\mathcal{D}_{\text{pay}}$ multicast blocks that A needs, A sends back a message to B describing those blocks (Step 21). (If A cannot find $\mathcal{D}_{\text{pay}}$ multicast blocks that it needs, B will need to try again later.) B then sends the blocks and completes the payment (Step 14). After receiving the blocks, A frees the debt-link and potentially later uses that debt-link to propagate another doin.

Notice that B does not need to obtain a signature from A to confirm the receipt of the payment. In fact, A has incentive not to send such a receipt, since it costs A's bandwidth. On the other hand, a rational A has no incentive to avoid freeing the debt-link after the payment. After all, even if it does not free the debt-link, B will not make a second payment (since B knows that B has made a pay-

[9]Recall from Section 5.3 that unexpected message losses can be readily dealt with as well, as long as the probability is not excessive.

ment and since messages are reliable).[10] Furthermore, not freeing the debt-link prevents A from making further profits on the debt-link.

6.5 Making Payments / Freeing Debt-links: Multi-hop Doins

The situation becomes more complex when a doin traverses through a chain of peers. The central difficulty is the lack of incentive for the doin issuer to notify other peers on the chain to free the debt-links. For example, imagine that a doin traverses three peers ACB and then B pays A for the doin. C needs to be notified of the successful payment so that it can free the debt-link to B. A has no incentive to notify C, since sending notification costs bandwidth. B does have the incentive to notify C, but B needs to present C a receipt from A to convince C. Unfortunately, A again has no incentive to send B such a receipt. The standard way of solving this problem is to use fair exchange protocols [12] so that A will only get the payment if it simultaneously gives B a receipt. Unfortunately, those heavy-weight protocols will involve many rounds of communication between A and B, and thus incur by far too much overhead for our purpose.

It turns out that by leveraging the rationality of the peers, a fair exchange is not necessary. Our key observation here is that while A has no incentive to confirm the payment, it does have the incentive to rebut a false payment claim. Namely, if B does not pay and falsely claims to C that B has paid, then A has the incentive to tell C that B has lied, because otherwise A will no longer get the payment. This observation leads to the following design where A does not send out a receipt — rather, it sends out a denial if necessary.

After making the payment, B directly propagates a simple Release message backward along the chain to request the debt-links to be freed (Steps 16 and 30). Each peer on the chain has clear incentive to relay the Release to its predecessor, since doing so is the only way to free its incoming debt-link. In each hop of its propagation, the Release message indicates the debt-link to be freed at the current hop, as well as a Release *local id*, which enables the two peers involved in the current hop to uniquely identify this Release. This local id serves to differentiate potential concurrent Releases for the same debt-link.

When the Release reaches A, if A did not actually receive the payment, A will propagate a Denial message along the chain to prevent the debt-links from being freed (Steps 27 and 36). A has clear incentive to do so, because otherwise A will no longer get a payment. Similarly, all the peers on the chain have the incentive to relay the Denial to make sure that the message reaches B (so that a rational B will make the proper payment), because otherwise their incoming debt-links will never be freed.

Obviously, if A is a deviator, A may still propagate a Denial even though payment has been made. A closer examination shows that A actually has no incentive to do so. The reason is exactly the same as in our earlier scenario with one-hop doins: A will simply not obtain a second payment even if it propagates a Denial.

6.6 Properly Timing Out

In our design above, after receiving a Release, a peer needs to wait for a potential Denial within a certain timeout, before freeing the debt-link. Properly timing out turns out to be non-trivial: If every peer uses the same timeout, then a deviator may delay the Denial and cause the Denial to reach some peers but not others before the timeout.

Conceptually to avoid this problem, peers farther away from the doin issuer need to have larger timeouts. Unfortunately, with potential manipulation from deviators, a non-deviator cannot easily use a potential hop count on the doin to determine its position on the chain. To overcome this difficulty, DCast uses the receiving time of a doin as a secure hop count that is guaranteed to be monotonic but not necessarily consecutive. Specifically, we divide each interval into a number (e.g., 10) of equal-length *subintervals*. A peer records the subinterval during which a doin is received as the doin's *r-stamp*. For a doin received in subinterval i, a non-deviator may relay the doin in subinterval j if $j \geq i + 1$. In other words, a doin traverses at most one non-deviator in one subinterval. Notice that r-stamp can never be decreased by a deviator.

With r-stamp, a peer B now simply uses a timeout duration of $2d\times$r-stamp after it forwards a Release to its predecessor C. If B and C are both non-deviators, then the r-stamp on B will be at least 1 larger than the r-stamp on C, and B's timer's duration is at least $2d$ longer than C's. Thus if C receives the Denial in time, B is guaranteed to receive the Denial in time as well. The following simple lemma, whose proof is trivial and thus omitted, formalizes the correctness of this design:

Lemma 1 Consider any consecutive sequence of non-deviators $A_1A_2...A_k$ that a doin traverses. With our above

[10]Formally, it is possible to model this interaction as a game with two Nash equilibriums, and the completion of the payment (i.e., B's sending the payment message to A) *publicly* signals to the two peers the transition from one equilibrium to the other. Properly switching between two equilibriums is a well-known technique [20] in mechanism design. Here we omit the tedious formalism for space constraints. Also recall from Section 5.3 that unexpected message losses can be readily dealt with as well, as long as the probability is not excessive.

design, if A_1 receives (or generates, if A_1 is the doin issuer) a Denial before its local timer expires, then the Denial will reach every peer A_i ($1 \leq i \leq k$) before A_i's timer expires.

7 Formally Achieving Safety-net Guarantee

This section formalizes and proves that DCast offers a safety-net guarantee, so that a non-deviator at least obtains a safety-net utility. Such formalization is non-trivial, and we start from the notion of a *reference execution*.

Using reference execution to capture safety-net utility. As mentioned in Section 3, we avoid defining any specific utility function, since it is impossible for any single function to capture all the rational peers. Without a concrete utility function, we are not able to assign a numerical value to the safety-net utility. Rather, we will use a *reference execution* as the reference point, where all peers follow a certain simple multicast protocol. Because there are no deviating peers in this reference execution, the utility achieved by each peer here is easy to understand, and one can readily plug in various utility functions to obtain instantiated utility values. We will then show that DCast's safety-net utility for a specific peer is the same as that peer's utility in the reference execution.

For our formal arguments next, we will assume a static setting where all peers join the multicast session before the session starts, and no peers join or leave on the fly. Our reference execution is the execution of a simple multicast protocol using pull-based gossip. There are m users in the system, and user i has $x_i \geq 1$ identities (peers). Let $n = \sum_{i=1}^{m} x_i$, which is the total number of peers. The multicast proceeds in rounds, where in each round the root sends erasure-coded multicast blocks to some set of peers chosen *adversarially*. Also in each round, a peer selects a uniformly random peer out of the n peers from whom to pull multicast blocks. Because all peers are cooperative, there are no concepts such as debt-links or doins. The execution has a single parameter Ψ. For each multicast block received, a peer sends to some special virtual sink $b \cdot \Psi$ cost bits, where b is the size of a multicast block.

The key property of the reference execution is that all peers follow the specified protocol, so that the utility achieved is trivial to understand. The fact that which peers receive multicast blocks directly from the root is chosen adversarially does not make the situation complex. The properties of gossiping (e.g., reaching all peers within $O(\log n)$ rounds with high probability) continue to hold in such a case. On the other hand, allowing the peers to be chosen adversarially will simplify our proof later. We intentionally allow users to launch sybil attacks in the reference execution. Because all peers (including the sybil peers) follow the specified protocol, the *only* effect of sybil attacks in the reference execution is to increase the logarithmic number of rounds needed for gossiping to finish. Section 5.2 explained that in DCast, users will have incentives to create only as many identities as they have enough bandwidth to issue/relay doins. So we expect the logarithmic increase to be rather limited.

Relative control cost. We next formalize the *relative control cost* in DCast. The cost bits sent/received by a peer can be either i) control messages (e.g., the message containing the summary) and control bits inside a message (e.g., MACs), or ii) junk blocks and multicast blocks. We refer to the first category as *control cost* and the second category as *inherent cost*. The *relative control cost* (denoted ρ) is the ratio between the two. Given our efficient DCast design, under normal parameters, ρ will be relatively small (e.g., 10% or 20%).

The main theorem. Now we can present our main theorem, and then explain how to prove it. Let λ be the probability that a debt-link is used in an interval, if it is free at the beginning of that interval.

Theorem 2 Assume that:

- When pulling multicast blocks from another peer, a non-deviator establishes enough new debt-links, if needed, to pull all the blocks that it needs from that peer.
- The condition at Step 20 of Algorithm 1 is always met.
- At any point of time, a deviator gets at least those multicast blocks that it would get if it did not deviate.

If we set the parameters in DCast such that:

$$(1+\rho)^2 \mathcal{D}_{\text{pay}} < (1+\rho)\mathcal{D}_{\text{link}} < \mathcal{D}_{\text{root}} \quad (1)$$

and

$$\sigma \leq \frac{1 - 2/\lambda}{(1+\rho)(1+2\mathcal{D}_{\text{link}}/\lambda)} \cdot \mathcal{D}_{\text{pay}} \quad (2)$$

where σ is defined as in Section 3, then DCast provides a safety-net guarantee where a non-deviating peer will always obtain at least the safety-net utility, despite the rational deviation of other peers. Here the safety-net utility is that peer's utility in the reference execution with the same set of peers and with a parameter $\Psi = (1+\rho)\mathcal{D}_{\text{root}}$.

The theorem is pessimistic and the assumptions in the theorem can mostly be replaced with weaker but less concise assumptions. Let us examine these assumptions one

by one. First, a non-deviator clearly needs to establish enough debt-links in order to achieve a reasonable utility. Second, the condition at Step 20 of Algorithm 1 is about whether the doin holder has enough data to pay for a doin. In our later experiments, we observe that over 99.95% of the doins can be properly paid. Finally, the last assumption is needed because if the deviators receive fewer multicast blocks after deviating, then the non-deviators will not be able to pull enough multicast blocks even if the deviators entirely cooperate with the pull. But since a rational deviation needs to be profitable, this necessarily means that the deviators send/receive far fewer cost bits after deviating, so that the reduction in cost outweighs the decrease in benefit (due to receiving fewer multicast blocks). This however, is rather unlikely since in practice receiving enough multicast blocks tends to be far more important for a peer than reducing costs.

Properly setting the various parameters in DCast to satisfy Equation 1 and 2 is not difficult. Notice that ρ tends to be a small value in DCast. The term of $2/\lambda$ is just the inverse of the expected number of times that a debt-link is used. Our experiments later show that this inverse is easily below 0.05. All these mean that $\mathcal{D}_{\text{pay}}$ only needs to be slightly larger than σ. The value of σ depends on the extra efficiency that the deviators can sometimes enjoy, by obtaining multicast blocks from each other. We expect that a relatively small σ (e.g., 2) is sufficient to capture most rational peers.

Proving Theorem 2. For lack of space, we defer the full proof to Appendix C. We briefly explain here how the proof is obtained. As discussed in Section 3, we need to consider only non-dominated collusion strategies. At a high level, our proof first shows that a non-dominated collusion strategy must be *non-damaging*. Intuitively, under a *non-damaging* collusion strategy, the *non-deviators'* utility is the same as or better than when the deviators exactly followed the protocol during their interactions with the non-deviators. See Appendix C for the formal definition. Intuitively, a non-dominated collusion strategy must be non-damaging because if a deviator takes some "damaging" action to the non-deviators, in DCast it will hurt its own utility as well.

Next based on Equation 2 in the theorem, we show that issuing doins is profitable under proper debt-link reuse, and thus the deviators will be willing to issue doins and propagate multicast blocks to the non-deviators. Furthermore because the collusion strategy is non-damaging, the utility of non-deviators during doin propagation and doin payment will be properly protected. This enables us to reason about the total cost bits that a non-deviator needs to send/receive for obtaining the multicast blocks. Similar reasoning applies when a non-deviator has the opportunity to issue doins. Combining all the above leads to the proof of the safety-net guarantee.

8 Implementation and Evaluation

We have implemented both a prototype and a simulator for DCast. Our prototype serves to validate the feasibility of DCast in practice, and also to validate the accuracy of our simulator. The simulator on the other hand, enables us to perform experiments of larger scale. The DCast prototype is implemented in Java 1.6.0 (about 4,000 lines of code) using regular TCP for communication. To strike a balance between performance and ease of implementation, we use an event-driven architecture with non-blocking I/O in bottleneck components, and a simpler multi-threading architecture in other parts.

Unless otherwise mentioned, all our experiments use the following setting. There are a total of 10,000 peers in the multicast session. The session is one hour long, with a streaming rate of 200Kbps. The session has 1800 rounds, divided into 60 intervals of 30 rounds each, and each subinterval has 3 rounds. The size of each multicast block is 1KB. We set $\mathcal{D}_{\text{link}}=\mathcal{D}_{\text{pay}}+0.5$, $\mathcal{D}_{\text{root}}=\mathcal{D}_{\text{pay}}+1$, and $\mathcal{D}_{\text{pay}}$ ranges from 2 to 4. We do not expect that $\mathcal{D}_{\text{pay}}=4$ is needed in practical scenarios, and thus this setting mainly serves as a stress test. In each round, the root sends 100 erasure-coded multicast blocks to 100 randomly chosen peers.[11] Any 50 out of these 100 blocks are sufficient to decode the video frames for a given round. The deadline of the multicast blocks is 20 rounds after they are sent from the root.

In the next, we will first elaborate our large-scale simulation results, and then present the validation results of the simulator using our prototype. The end-to-end metric for video streaming is usually delivery_rate, which is the fraction of rounds that an average peer receives the corresponding multicast blocks by the deadline and thus can render the video frames for those rounds. In *all* our simulation experiments next (even with colluding deviators), we observe a delivery rate for the non-deviators of at least 99.95% and thus we will not discuss delivery_rate further. The following presents results that help us to gain further insight into the behavior and the safety-net guarantee of DCast.

[11] Note that we intentionally keep the load on the root to be rather light. This not only reserves enough bandwidth for the root to receive junk blocks, but also stress tests our design. The total load, including the load incurred by the junk blocks, on our root is less than half of the load in similar experiments with gossip-based multicast in prior work [17].

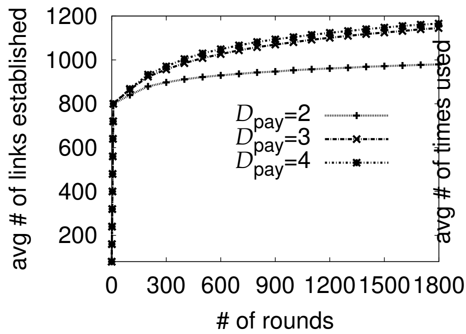

Figure 2: # of debt-links created by an avg peer.

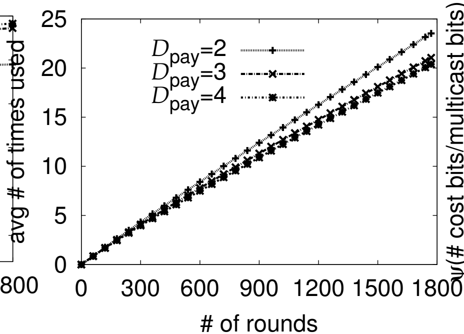

Figure 3: # of times that an avg debt-link is used.

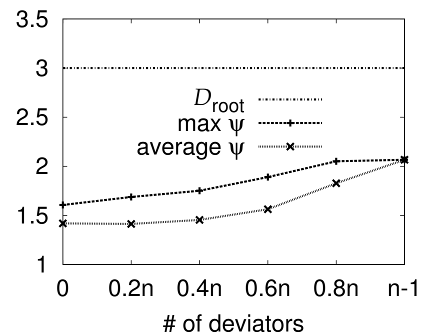

Figure 4: Example of safety-net guarantee.

No deviators: Social cost of DCast. We first consider scenarios where no peer deviates. Figure 2 plots the average number of incoming debt-links established by each peer, denoted as links_established. Debt-link establishment purely involves junk blocks, and hence is a social cost in DCast.[12] The figure shows that for the first few rounds, there is a sharp jump because all peers are establishing new debt-links. After that, the curve flattens out or grows rather slowly, and remains below 1,200 even in the end.[13] On average, each peer only establishes 0.67 debt-links in each round. This number is rather small since establishing a debt-link just involves no more than 4.5 junk blocks, while a peer already needs to receive and send roughly 50 multicast blocks in each round.

No deviators: Number of times a debt-link is used. Next we examine the number of times that a debt-link is used (denoted as times_used) and payment success rate (defined later) in DCast. These two quantities capture two key assumptions in Theorem 2. times_used exactly corresponds to the $\lambda/2$ term in the theorem. An overly small times_used may make the profit of issuing doins unable to offset the debt-link establishment cost. Figure 3 shows that an average debt-link is used over 0.33 times within each interval (i.e., one minute), and over 20 times during the one-hour multicast session. This value is far larger than what is needed to offset the initial debt-link establishment cost: Establishing a debt-link from A to B involves $\mathcal{D}_{\text{pay}}+0.5$ junk blocks, while every time the debt-link is used, A sends one multicast block and in return either receives $\mathcal{D}_{\text{pay}}$ multicast blocks (if A issues a doin) or eliminates $\mathcal{D}_{\text{pay}}$ of debt (if A relays a doin).

No deviators: Payment success rate. *Payment success rate*, denoted as pay_succ_rate, is the fraction of expired doins that are paid by the time that the multicast session ends.[14] Even though we have proved that a rational peer always has the incentive to pay for an expired doin, pay_succ_rate may still be below 100% if the peer has no multicast blocks to offer to the doin issuer. If pay_succ_rate is too low (e.g., below 50%), then the assumption in Theorem 2 may no longer hold and the peers may not have incentive to issue doins anymore. Our experiments show a pay_succ_rate of above 99.95% under all $\mathcal{D}_{\text{pay}}$ values that we try (i.e., 2 to 4). We believe that the corresponding payment failure rate is rather negligible—even peer failure rate can be higher than 0.05%.

Effect of deviators. So far our results on links_established, times_used and pay_succ_rate are only for scenarios where no peer deviates. For times_used and pay_succ_rate, notice that larger times_used and pay_succ_rate benefit all peers. This means that the deviators actually have the incentive to further increase these two quantities, if possible. Thus at the very least, we do not expect that the deviators will behave in such a way to significantly decrease them, as compared to what we observe in our experiments where no peer deviates to intentionally increase them. On the other hand when peers deviate, links_established and the corresponding social cost may increase substantially, as we show in the experiments next on the safety-net guarantee. Notice that there the debt-link establishment social cost will be already captured in (i.e., deducted from) the safety-net utility.

Having 0 through $n-1$ deviators: Safety-net utility under an example collusion strategy. We next aim to

[12] The social cost resulted from sending junk blocks to the root (for receiving multicast blocks from the root) is by far dwarfed by the debt-link establishment cost.

[13] Notice that because of the possibility of simultaneous exchange (see Section 6) as well as getting multicast blocks as doin payments, the number of debt-links needed by a peer can be smaller than the number of multicast blocks received during an interval (which is $50 \times 30 = 1500$).

[14] Doins issued in the last interval are excluded because they have not expired yet when the session ends.

illustrate the safety-net guarantee offered by DCast under an example collusion strategy, with $\mathcal{D}_{\text{pay}} = 2$. We do not intend to be exhaustive here—experimental methods by definition cannot cover all collusion strategies. It is not meaningful to consider "representative" collusion strategies either, because the human attacker is intelligent and may devise novel collusion strategies that we do not expect today. Ultimately, the safety-net guarantee has to be proved, as we did in Theorem 2, instead of being experimentally confirmed. Thus our experiments here purely serve as an example.

In this example collusion strategy, a deviator never pulls data from a non-deviator, to avoid receiving doins and paying for doins. This also means that a deviator never establishes debt-links from a non-deviator. The deviators collude by pulling data from each other, without the constraints of debt-links and doins (i.e., they trust each other). Doing so enables them to disseminate the multicast blocks faster. Because issuing doins is profitable, a deviator will still issue doins to a non-deviator. Globally, under this collusion strategy, a doin will only flow from a deviator to a non-deviator, or from a non-deviator to another non-deviator.

Our experiments vary the number of deviators from 0 to $n-1$, where n is the total number of peers. For each non-deviator, we log the total number of cost bits sent and received, as well as the total number of multicast bits received. Note that since the delivery_rate we observe is close to 100%, the non-deviators almost always receive all the multicast blocks needed. The ratio between the two numbers is essentially the Ψ parameter in the safety-net utility. Figure 4 plots both the average and the maximum Ψ of the non-deviators, which is always below $\mathcal{D}_{\text{root}}$ and thus is consistent with Theorem 2. Even with $n-1$ deviators, by running the DCast protocol, the single non-deviator still enjoys a delivery_rate of above 99.95%, while paying about $\Psi = 2$ cost bits for each multicast bit received. Without DCast, the non-deviator would fail to obtain any multicast blocks from other peers at all under this collusion strategy.

As expected, in Figure 4 the average and maximum Ψ increase with the number of deviators. Such increase is the combined result of paying for more doins, issuing fewer doins, and establishing more debt-links. Specifically, the average number of debt-links established by a peer in each round increases from roughly 0.6 to 2.2 when the number of deviators increases from 0 to $n-1$. This is because a deviator does not pull multicast blocks/doins from a non-deviator, and thus it does not perform simultaneous exchange (see Section 6) with a non-deviator either. In turn, all the multicast blocks propagated from a deviator to a non-deviator come with doins that will occupy the debt-links at least for the current interval.

Table 1: Validating the simulator using the prototype.

metric	prototype	simulation
delivery_rate	99.93%	99.996%
links_established	1114	890
times_used	13.5	12.7
pay_succ_rate	99.97%	1.0

Finally, when there is no deviator, Ψ is still around 1.5. Here debt-link establishment and sending junk blocks to the root only contribute about 0.056 and 0.0012 to Ψ, respectively. The remaining part is for sending multicast blocks to other peers. This part is larger than 1.0 because with simultaneous exchange, sometime multiple peers may send redundant blocks to the same target peer. Our simulator is particularly pessimistic in this aspect—the multicast blocks received by the target peer will only show up in the peer's summary in the next round (i.e., 2 seconds later). This causes a relatively large number of redundant blocks. In fact, further simulation shows that if the multicast blocks received are immediately visible in the peer's summary, then the Ψ observed in our experiment will drop from around 1.5 to 1.077, which is rather close to 1.

Validate the simulator using the prototype. Finally, to validate our simulator as well as the simulation results we have discussed so far, we run our DCast prototype on the Emulab testbed. Due to resource constraints, the scale of the experiments is limited to 180 peers. Because our protocol is an *overlay* multicast protocol, we are not able to emulate a network topology, which would require us to emulate all the routers between each pair of the peers. To capture communication delay, we artificially add wide-area communication delays for the messages. To use realistic delay values, we map each peer to a random node in the King Internet latency dataset [9], and then use the latency there as the delay value between the corresponding peers.

Table 1 compares the results from a half-hour multicast session using our prototype versus the results from our simulator under the same setting. The value of times_used is rather similar in the two cases. While delivery_rate and pay_succ_rate on Emulab are slight lower than in the simulation, their absolute values remain rather high. The difference is mainly caused by unexpected delays on some Emulab nodes, which our simulator is unable to capture. Finally, the Emulab experiment establishes about 25% more debt-links than the simulation experiment. This is due to the processing delay on some Emulab nodes, which

causes peers to establish more debt-links to ensure that they can get the multicast blocks in time. Because debt-link establishment contributes to a rather small component of Ψ, we do not expect such difference to significantly affect the safety-net utility as observed in the simulation.

9 Conclusion

This paper aims to sustain collaboration in overlay multicast systems despite the collusion and sybil attacks by rational users. We first introduced the novel notion of a *safety-net guarantee*, that focuses on protecting the utility of the non-deviators. We then presented the decentralized *DCast* multicast protocol that uses a novel mechanism with debts circulating on pre-established debt-links. This novel mechanism enabled us to overcome two fundamental challenges introduced by rational collusion. We formally proved that the protocol offers a safety-net guarantee, and further demonstrated via prototyping and simulation the feasibility and safety-net guarantee of our design in practice.

10 Acknowledgments

This work is partly supported by National University of Singapore FRC grant R-252-000-406-112.

References

[1] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *PODC*, 2006.

[2] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in p2p systems. In *P2P*, 2003.

[3] A. Clement, J. Napper, H. Li, J.-P. Martin, L. Alvisi, and M. Dahlin. Brief announcement: Theory of BAR games. In *PODC*, 2007.

[4] L. P. Cox and B. D. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *SOSP*, 2003.

[5] J. Douceur. The Sybil attack. In *IPTPS*, 2002.

[6] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *DIALM*, 2002.

[7] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An architecture for secure resource peering. In *SOSP*, 2003.

[8] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The millicent protocol for inexpensive electronic commerce. In *Proceedings of the International World Wide Web Conference*, 1995.

[9] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM*, 2002.

[10] A. Hayrapetyan, E. Tardos, and T. Wexler. The effect of collusion in congestion games. In *STOC*, 2006.

[11] I. Keidar, R. Melamed, and A. Orda. EquiCast: Scalable multicast with selfish users. In *PODC*, 2006.

[12] S. Kremer, O. Markowitch, and J. Zhou. An Intensive Survey of Fair Non-Repudiation Protocols. *Computer Comm.*, 25(17), 2002.

[13] R. Landa, D. Griffin, R. Clegg, E. Mykoniati, and M. Rio. A sybil-proof indirect reciprocity mechanism for peer-to-peer networks. In *INFOCOM*, 2009.

[14] M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely fair SFE and coalition-safe cheap talk. In *PODC*, 2008.

[15] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives. In *SIGCOMM*, 2008.

[16] D. Levin, R. Sherwood, and B. Bhattacharjee. Fair File Swarming with FOX. In *IPTPS*, 2006.

[17] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin. Flightpath: Obedience vs. choice in cooperative services. In *OSDI*, 2008.

[18] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR Gossip. In *OSDI*, 2006.

[19] R. T. B. Ma, S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau. Incentive and Service Differentiation in P2P Networks: A Game Theoretic Approach. *IEEE/ACM Transactions on Networking*, 14(5), 2006.

[20] G. J. Mailath and L. Samuelson. *Repeated Games and Reputations*. Oxford University Press, 2006.

[21] A. Nandi, T.-W. J. Ngan, A. Singh, P. Druschel, and D. S. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Middleware*, 2005.

[22] T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Incentives-compatible peer-to-peer multicast. In *P2P Econ*, 2004.

[23] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[24] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent. In *NSDI*, 2007.

[25] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One hop reputations for peer to peer file sharing workloads. In *NSDI*, 2008.

[26] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, and D. Zhang. Contracts: Practical Contribution Incentives for P2P Live Streaming. In *NSDI*, 2010.

[27] T. Poutanen, H. Hinton, and M. Stumm. NetCents: A lightweight protocol for secure micropayments. In *Proceedings of the USENIX Workshop on Electronic Commerce*, 1998.

[28] R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In *Proceedings of the International Workshop on Security Protocols*, 1997.

[29] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. Karma: A secure economic framework for peer-to-peer resource sharing. In *P2P Econ*, 2003.

[30] B. Yang and H. Garcia-Molina. PPay: Micropayments for peer-to-peer systems. In *CCS*, 2003.

[31] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against sybil attacks. In *IEEE Symp. on Security and Privacy*, 2008.

[32] Z. Zhang, S. Chen, and M. Yoon. MARCH: A Distributed Incentive Scheme for Peer-to-Peer Networks. In *INFOCOM*, 2007.

A Simultaneous Exchange

Our DCast design in Section 6 only allows pull-based gossiping, and the multicast blocks pulled always accompany some doins. Imagine two peers A and B, each with a multicast block that the other party needs. These two blocks can be propagated using two doins, which will occupy a debt-link from A to B and a debt-link from B to A.

As an optimization, DCast permits A and B to exchange these two multicast blocks without eventually occupying any debt-links, if they both would like to do so. Specifically, when B pulls the multicast block from A, A will propagate the block together with a doin as usual. If A is interested in performing simultaneous exchange with B, A will set a special flag on the doin, and also indicate which multicast block that A wants (based on B's summary). The flag tells B that if B returns the requested block by the next round, A will consider the doin as never having been propagated to B, and will free the debt-link from A to B immediately. Notice that such simultaneous exchange does not require any free debt-link from B to A, and it only succeeds if A and B both want to do so (i.e., if A sets the flag and if B returns the block as requested). If B does not return the block as expected, this interaction will just be considered as a normal pull of B from A, and B will hold the doin. This protocol can be viewed as an elegant and highly-efficient fair exchange protocol [12] enabled by our design of doins and debt-links in our specific context.

B Dealing with Clock Drifts

In Section 6, we made the simplifying assumption that clock error is zero among the peer. While it is reasonable to assume loosely synchronized clocks, clock drifts are never zero. Non-zero clock drifts will introduce two issues in the design in Section 6. First, when a peer A propagates a doin (issued during interval i) to another peer B, A may consider the current time to be in interval i, while B may consider the current time to be in interval $i+1$ and thus ignore the doin. This problem is trivial to address—A will simply avoid propagating doins issued during interval i (and start issuing doins for interval $i+1$) when it is rather close to the end of interval i. A peer should accept doins whose issuing interval is either the current interval or some future interval.

A second and similar issue arises for subintervals: When A relays a doin with an r-stamp of x in the $(x+1)$th subinterval (according to A's local clock) to B, B may believe that the current time is still in the xth subinterval. This is easily addressed by having A avoid relaying doins with an r-stamp of x during the first few seconds of subinterval $x+1$.

Finally, clock drifts will not affect the proper timing out for a Release in Section 6.6, because there we rely on the timer *durations* instead of the clock readings at individual peers.

C Proof for Theorem 2

We first need to formally define the notion of *non-damaging* collusion strategies. A collusion strategy is *non-damaging* if for every non-deviator (all steps below refer to steps in Algorithm 1):

1. If it issues a new doin, then for that doin it successfully executes Step 19 at least once.
2. If it executes Step 19 for a doin (which may or may not correspond to a doin that it has issued), then for that doin it successfully executes either Step 20-25 exactly once, or Step 20-23 exactly once.
3. It never executes Step 27.
4. If it holds a doin that has expired, then for that doin it successfully executes Step 12-16 exactly once, and the corresponding incoming debt-link will be freed.
5. If it relays a doin, then for that doin it successfully executes Step 29-32 exactly once, and the corresponding incoming debt-link will be freed.
6. It never executes Step 33-37.
7. It never receives non-protocol messages (i.e., messages not specified by the DCast protocol).

We next prove Theorem 3, which will later be used to prove Theorem 2.

Theorem 3 *Under the assumptions made in Theorem 2 and the DCast parameters set in Theorem 2, a non-dominated collusion strategy must be non-damaging.*

Proof sketch: We need to introduce some notions for the formal arguments next. An *N-sequence* is a sequence of non-deviators that a doin traverses. An N-sequence is *maximal* if it is not part of another N-sequence. A *segment* is an N-sequence prepended by the deviator (if any) that sends the doin to the first peer in the N-sequence, and appended by the deviator (if any) that receives the doin form the last peer in the N-sequence. The first peer (which can be either a deviator or a non-deviator) of a segment is called the *head* of the segment, and the last peer is called

the *tail*. Notice that by definition of a segment, all non-deviators on a segment see the same doin id, and that doin id is called the doin id of the given segment. It is possible for multiple segments to have the same doin id.

Now consider any given non-dominated collusion strategy α, and we will show that it satisfy all the properties needed to be non-damaging.

Property 7. Non-protocol messages will always be ignored by a non-deviator, and thus their only effect is to reduce the utility of the sender and the receiver of the messages. We claim that in α no deviators will send non-protocol messages to non-deviators, because otherwise α will be dominated by some other collusion strategy (which avoids sending these messages).

Property 1. Consider any given non-deviator A that issues some given new doin. Consider the segment starting from A with that doin id. (Notice that there can be multiple segments with that doin id, but exactly one of those will start from A.) If the tail of the segment is a non-deviator, then obviously it will initiate a payment and cause A to execute Step 19. Now consider the case where the tail is a deviator D. Let D's predecessor on the segment be B, which must be a non-deviator (B can be A itself).

We show via contradiction that under α, A will execute Step 19 at least once for that doin. Assume that A never executes Step 19. Then A will never mark the doin as "paid". In turn, we claim that the debt-link from B to D will never be freed. The reason is that to free the debt-link, B must receive a Release message, and then must not receive a Denial message within the timeout. But because the Release message will reach A, and A will generate a Denial message immediately, Lemma 1 tells us that B will receive the Denial message before timing out. Because the debt-link will not be freed, D will need to establish another new debt-link. On the other hand, if we consider a second collusion strategy β where D simply makes the payment properly to A, D's utility will be better under β than under α. This contradicts with the fact the α is non-dominated.

Property 2. If a non-deviator A executes Step 19 for some doin, then we claim that the sender B (regardless of whether it is a non-deviator or deviator) of the "payrequest" message must enable A to complete Step 20-23. The reason is that the only effect of Step 19 is to trigger A to send the message at Step 21, which is of no use to B unless Step 23 is completed. If B does not enable A to complete Step 22 (implying that B is a deviator), then not sending the "payrequest" message would actually improve B's utility, rendering α dominated.

Property 3. If a non-deviator A executes Step 27, then the corresponding segment must end with some deviator D. This Denial message must have been triggered by some previous Release message. Notice that the only effect of a Release message is to free debt-links. But because A executes Step 27, by Lemma 1 none of the debt-links will not be freed. We thus claim that D will never send such a Release message (which triggers the Denial message), because otherwise not sending the Release message would improve D's utility, rendering α dominated.

Property 4. Consider any given non-deviator A that holds an expired doin. Notice that the doin issuer (as shown in the doin's id) may be different from the head of the corresponding segment. If the doin issuer is a non-deviator, then A can always complete the payment and then propagate a Release message to its predecessor, causing the debt-link to be freed. Notice that even if the head of the segment is a deviator D, D will not send a Denial message. The reason is that the only effect of sending a Denial message is to cause some debt-links to be permanently occupied, as well as incurring some overhead to every peer on the segment (including D itself). In particular, A will not make a second payment if it receives the Denial message. Causing the debt-links to be occupied will only force the non-deviators to establish new debt-links. Because by our design, when establishing new debt-links, they will always first establish debt-links to compensate for occupied debt-links. Thus the net effect is only to incur some extra debt-link establishment overhead for the non-deviators on the segment. This means that D's sending the Denial message will simply decrease the utility of some peers, including itself. Because the collusion strategy α is non-dominated, D will clearly not do so under α.

If the doin issuer is a deviator D, we claim under α, D will always accept the payment because it always improves D's utility if it does so. For the same reason as above, no Denial message will be propagated on the segment, and thus A's incoming debt-link will be freed.

Finally, it is possible for the doin issuer to be *non-existent*, when the head of the segment is a deviator D. A doin has a *non-existent* issuer if either the IP address in the doin's id does not correspond to any peer, or the peer at that IP address does not have the corresponding private key for the public key in the doin's id. Obviously, a doin with a non-existent issuer cannot be paid. We claim that because the collusion strategy α is non-dominated, the doin issuer will never be non-existent. The reason is that if it were non-existent, then D should just replace the doin issuer with itself, and accept payment later. This must improve D's utility, making α dominated.

Property 5 and 6. Consider any given non-deviator A that relays a doin, and consider the corresponding segment. If the tail of the segment is a non-deviator, then the tail holds an expired doin and will try to make the payment. As shown above for Property 4, the tail will successfully make the payment and propagate a Release message backward. We have also shown above that there will not be a Denial message. Similarly, if the head of the segment is a non-deviator, then we have shown earlier (for Property 1, 2, and 3) that a Release message will be propagated to the head (passing A), and there will be no Denial messages.

The only remaining case is when the head and the tail are two deviators D_1 and D_2. We first show that there will not be a Denial message. Assume by contradiction that D_1 sends a Denial message to its successor B before B's timer expires. Because all peers on the segment, except D_1 and D_2, are non-deviators, Lemma 1 tells us that all these peers will receive the Denial message before their timers expire. Thus none of the debt-links will be freed, and the only effect of this Denial message is to cancel out some earlier Release message. We can now construct a second collusion strategy β by modifying α so that D_2 does not send the earlier Release message. Doing so clearly reduces the cost bits sent/received by D_1 and D_2, which would imply that β dominates α.

Next we prove that there is a Release message from D_2 to its predecessor C on the segment, via a contradiction. If there is no Release message, it means that D_2's incoming debt-link cannot be freed, and D_2 needs to establish a new debt-link. Consider a second collusion strategy β where D_2 makes a payment to D_1 and then propagates a Release. Compared to α, β improves the utility of both D_1 and D_2, which would make α dominated. □

Now we are ready to prove Theorem 2.

Proof sketch for Theorem 2: By the definition of safety-net guarantee, we only need to consider non-dominated collusion strategies. Consider any given non-dominated collusion strategy α. We need to show that a non-deviator will achieve at least as good utility under α in the DCast execution as it would in the reference execution. We set the reference execution such that in each round, the root sends multicast blocks to exactly the same set of peers as in the DCast execution. We can do this because the reference execution allows this set of peers to be adversarially chosen.

We will first prove that at any point of time, each peer (non-deviator or deviator) in the DCast execution gets at least those multicast blocks that it gets in the reference execution. We prove via an induction on the number of deviators. The induction base for zero deviator obviously holds. Now assume that the statement holds when the number of deviators is k. If we have $k+1$ deviators, let us consider any given deviator. By the last assumption in Theorem 2, at any point of time, that deviator must get at least those multicast blocks that it would get if it did not deviate. On the other hand, if it did not deviate, then we would have only k deviators. By inductive hypothesis, every peer in the DCast execution with k deviators gets at least those multicast blocks that it gets in the reference execution. Thus the given deviator must get at least those multicast blocks that it gets in the reference execution.

Now consider any given non-deviator. Compared to the reference execution, the only possible scenario where it may get less multicast blocks is when it pulls from a deviator. We already proved above that the deviator must have all the multicast blocks that it has in the reference execution. Second, the non-deviator will always try to establish enough debt-links for pulling the multicast blocks. A deviator may choose to propagate the multicast blocks or not propagate the multicast blocks using those debt-links. Consider a debt-link that is free in the current interval. One can show that if the deviator issues doins using that debt-link whenever possible, then this debt-link will be used on expectation for x times where $x \geq \lambda/2$. The total expected cost to the deviator will be the same as sending/receiving $(1+\rho)(x+\mathcal{D}_{\text{link}})$ multicast blocks, while the total expected reward will be receiving $(x-1)\mathcal{D}_{\text{pay}}$ multicast blocks. Because the deviator's utility function is such that it will benefit from sending one multicast block and then receiving σ multicast blocks, the expected reward always exceeds the expected cost. Thus propagating multicast blocks (and issuing doins) using that debt-link will increase that deviator's utility. Because the collusion strategy α is non-dominated, that deviator must issue doins on that debt-link (because otherwise α will be dominated by some "better" collusion strategy).

We have now proved that at any point of time, each peer in the DCast execution gets at least those multicast blocks as it would get in the reference execution. We next would like to reason about the cost bits that a non-deviator needs to send and receive in the DCast execution. Because α is non-dominated, Theorem 3 tells us that it must be non-damaging. By definition of non-damaging collusion strategies, we know that the total number of cost bits that a non-deviator sends and receives in the DCast execution is at most $b(1+\rho)\mathcal{D}_{\text{root}}$ per multicast block received. This is true regardless of whether it directly receives the multicast blocks from the root or it receives them from another peer and then relays/pays for the doin.

Finally, we need to consider the case where a non-deviator issues doins. Under a non-damaging collusion

strategy, every doin is paid. Thus if a non-deviator manages to issue new doins, it can only increase the non-deviator's utility above the safety-net utility. □

D Malicious Peers

As explained in Section 4, the safety-net guarantee intentionally assumes away malicious peers. The need for doing so is simple: Malicious peers can always send junk bits to a non-deviator, and drive down that non-deviator's utility arbitrarily. Nevertheless, we still want to ensure that at least our DCast design does not introduce any extra vulnerability. For example, we should avoid grim trigger designs [16] where the whole system can "melt down" due to a single malicious user sending a single message.

Because all the multicast blocks in DCast are signed by the root, the only kind of possible malicious attacks is DoS attacks. Regardless of the design, overlay multicast by definition has a single root and is rather vulnerable to DoS attacks to the root. Our goal is thus to ensure that the attacker cannot significantly amplify its attack capacity (i.e., its budget on attack bandwidth) by exploiting our design, as compared to directly sending junk bits in order to DoS the root and/or the peers. We next consider the possible DoS attacks in DCast.

First, the malicious peers may attract the root to send multicast blocks to them and then discard those blocks. However, for each multicast block from the root, the malicious peers need to send $\mathcal{D}_{\text{root}}$ junk blocks. If the malicious peers can send enough junk blocks to attract all the multicast blocks (notice that the root usually sends out as many multicast blocks as its bandwidth allows), then they can likely already directly DoS the root with such bandwidth. If the malicious peers simply join the multicast session and refuse to send junk blocks to the root, then they will quickly be flagged as non-accepting by the root. The root may further control such overhead by rate-limiting new joins.

Second, malicious peers may DoS the other peers in the system. For example, they may remain silent when other peers pull from them. To do so, however, the malicious peers need to receive the junk blocks for debt-link establishment first, which consumes attack bandwidth. Further because a peer monitors the multicast blocks it receives so far, the peer will simply establish new debt-links and pull from other peers when under such attack. Similarly, a malicious peer may participate in doin issuance and circulation. But the *worst* situation that can happen there is for all the debt-links on the propagation chain to be permanently occupied. With our design in Section 6.6 with subintervals, the total number of non-deviators that a doin can traverse is bounded by the total number of subintervals (e.g., 10) in an interval. This in turn, limits the damage that a malicious peer can cause in such a case. Finally, the malicious peers will need to send/receive multicast blocks during this attack, again making the damage constrained by their attack bandwidth.