Distrib. Comput. (2003) 16: 49–68
Digital Object Identifier (DOI) 10.1007/s00446-002-0082-4

DISTRIBUTED COMPUTING

Tolerating malicious gossip

Yaron M. Minsky, Fred B. Schneider

Department of Computer Science, Cornell University, Ithaca, New York 14853, USA (e-mail: {yminsky,fbs}@cs.cornell.edu)

Received: October 2001 / Accepted: July 2002

Abstract. A new class of gossip protocols to diffuse updates securely is presented. The protocols rely on annotating updates with the path along which they travel. To avoid a combinatorial explosion in the number of annotated updates, rules are employed to choose which updates to keep. Different sets of rules lead to different protocols. Results of simulated executions for a collection of such protocols are described – the protocols would appear to be practical, even in large networks.

Keywords: Gossip protocols – Byzantine fault tolerance – replicated data

1 Introduction

The essential characteristic of a *gossip protocol* is that information exchanges, or *gossips*, occur between hosts and randomly chosen *gossip partners*. Gossip protocols have proven effective in building systems resilient to benign host failures [1,13,5,16,15,12]. In this paper, we discuss techniques for constructing gossip protocols that are resilient to malicious host failures.

An archetypal problem for gossip protocols, introduced in [9], is the *diffusion problem*, where some update u known to a few hosts must be disseminated and *accepted* by all correctly operating hosts. When failures are known to be benign, the diffusion problem can be solved by a simple gossip protocol: a host H accepts an update as soon as it gossips with any host that already has (or claims to have) accepted that update. But, this protocol does not tolerate malicious failures, since a malicious host could cause other hosts to accept arbitrary updates.

In designing a diffusion protocol that can tolerate up to t malicious failures, the central question is what evidence would allow a host to accept an update? One obvious answer is to accept an update u only after gossiping with $t+1$ hosts that claim to have accepted u. We refer to such gossip protocols as *direct verification* protocols. Protocols using direct verification were first studied in [9], and the approach was found to yield protocols that are necessarily slow to complete or impose a high load on at least some hosts.

In this paper, we consider a more efficient class of gossip protocols called *path verification* protocols. Path verification protocols allow an update to be forwarded even when the forwarding host has not accepted that update. In these protocols, hosts exchange *proposals*, each consisting of an update u paired with a *gossip path* p. The gossip path records a sequence of hosts through which the proposal is alleged to have traveled. And a host accepts an update u only after receiving $t+1$ proposals with the same update and disjoint gossip paths.

The central problem in designing a path verification protocol is deciding which proposals a host will store. Because of the large number of paths along which a proposal can be transmitted, storing all possible proposals is not practical. The decision of which proposals to keep can be made using two sub-protocols: a *selection protocol* that chooses a single distinguished proposal at every host, and a *sampling protocol* that, for each host, gathers from a set of randomly selected hosts proposals chosen by the selection protocol at that host. This paper is thus about the design of these sub-protocols and about the performance and security implications of various design choices.

The paper is organized as follows. Section 2 presents our model and assumptions; it also discusses performance metrics. Section 3 defines the class of path verification protocols and presents some protocols within that class. Section 4 shows how the protocols of Sect. 3 can be improved by introducing a more efficient sampling protocol. Section 5 then analyzes the performance of our path verification protocols. Related work and some open problems are reviewed in Sect. 6.

Supported in part by ARPA/RADC grant F30602-96-1-0317, AFOSR grant F49620-00-1-0198, Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Material Command USAF under agreement number F30602-99-1-0533, National Science Foundation Grant 9703470, and a grant from Intel Corporation. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government.

2 Model and definitions

Assume a synchronous system (i.e., host execution speeds and message delivery delays are bounded), so that protocols can be described in terms of rounds. We also assume that a host can determine the sender of each message it receives and that messages are not modified in transit.[1] And we assume that every site can communicate with every other site. Note that a cryptographic infrastructure with facilities for constructing digital signatures is not being assumed.[2] These assumptions are the same as those first made in [9].

Hosts that are *uncorrupted* follow their specified protocols; due to failures and/or attacks, *corrupted* hosts can exhibit arbitrary and malicious behavior, perhaps even colluding with other corrupted hosts. If a host is uncorrupted, then it is presumed to be uncorrupted throughout execution of the protocol.

Under this model, we investigate protocols that solve:

(2.1) Diffusion Problem: Of n hosts that make up a system, at most t are corrupted. Let there exist k *source* hosts, each uncorrupted and each of which knows that it is a source host and has accepted *initial update* u. Assume $1 \leq t < k$ holds. Once an uncorrupted host accepts an update, that host is considered *updated*.

A solution to the diffusion problem satisfies:

Accept Safety: No uncorrupted host accepts an update other than u.

Accept Liveness: With probability 1, every uncorrupted host eventually accepts some update A source host presumably receives initial update u from outside the diffusion protocol. One scheme might have source hosts receive u from some distinguished host D, which is assumed to be uncorrupted. If D sends u to $2t + 1$ or more hosts, then at least $t + 1$ recipients are uncorrupted and satisfy the definition of a source host. Further, the assumption that D is uncorrupted can be dropped by disseminating u using an agreement protocol with D serving as the "transmitter".[3]

A second scheme for distributing initial update u to source hosts is possible when u is part of a shared environment that k uncorrupted hosts can each directly sense. An uncorrupted host reading u from that shared environment becomes a source host. For example, under some assumptions about the timing characteristics of the network and the nature of host failures, a host H can detect whether another host J has crashed by whether J responds to messages. If H detects that J has crashed, then H can act as a source host with its initial update being a datum indicating that J has crashed.

In analyzing solutions to Diffusion Problem (2.1), three performance metrics are likely to be of concern:

Diffusion time: Number of rounds until all uncorrupted hosts accept u.

Host load: Maximum number of messages in a given round that a single uncorrupted host H sends (to any host) plus the number of messages that H receives from uncorrupted hosts.[4]

Message size: The worst-case size of messages sent by uncorrupted hosts.

Computation time: The worst-case computational complexity of running the protocol on any host.

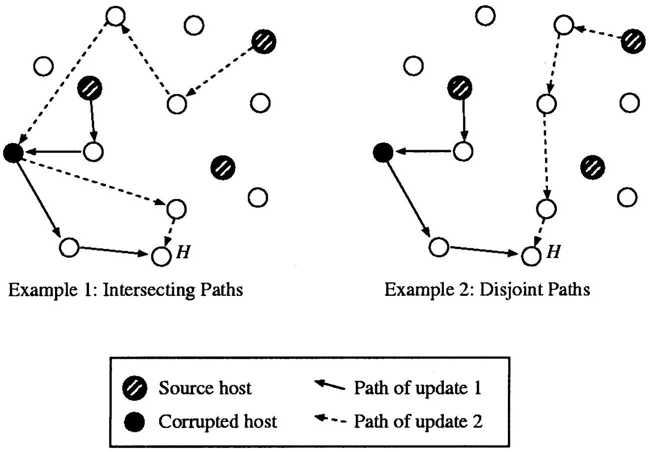

Fig. 1. Examples of gossip propagation

It is these metrics that drive the design of our protocols.[5]

The Diffusion Problem is only concerned with the distribution of a single update and there is no requirement that a diffusion protocol ever terminate. In a real system a non-terminating protocol capable of distributing only a single update would not be particularly useful – a means for distributing multiple updates would therefore be required. Further discussion of termination and the distribution of multiple updates can be found in Sects. 5 and 6.

3 Path verification

One of the reasons gossip protocols based on direct verification are slow (as shown in [9]) is that direct verification does not allow a host to further disseminate an update until that host has itself accepted the update. To improve upon the performance of direct verification, path verification protocols have hosts forward updates not yet accepted. Forwarded updates might be modified in transit, but this problem can be solved by taking into account the paths along which updates are transmitted.

Figure 1 depicts updates transmitted along two different paths. Nodes correspond to hosts; a directed edge from a host I to a host J signifies that I forwarded update u to J.

The corrupted host in example 1 of Fig. 1 is part of both paths to H, and therefore it can affect the update transmitted along each; but the corrupted host in example 2 can affect the

[1] In fact, it suffices to assume that corrupted hosts cannot impersonate uncorrupted hosts.

[2] Having digital signatures would simplify the diffusion problem considerably, because malicious hosts are then prevented from altering updates that they relay.

[3] The value that uncorrupted source nodes obtain from execution of an agreement protocol, although equal, might be meaningless were transmitter D corrupted.

[4] Messages from corrupted hosts are excluded from host load because a corrupted host could send an unbounded number of messages. If host load included messages from corrupted hosts then host load would be unbounded.

[5] Diffusion time is called *delay* in [9]. Host load is replaced there by *fan in*, defined as the expected maximum number of messages received by any replica in any round from correct replicas. Our host load seems the more useful metric, since it encompasses the cost of both in-bound and out-bound traffic that a host must handle.

update transmitted along only one of the two paths depicted. Generalizing, i distinct corrupted hosts are necessary in order to corrupt updates transmitted along i disjoint paths.

Accept Safety of Diffusion Problem (2.1) holds if and only if a host only accepts an update if that update was accepted previously by some uncorrupted host. The examples of Fig. 1 suggest that a host H can conclude that some uncorrupted host has previously accepted u if H receives u along $t+1$ disjoint paths originating at hosts that claim to have accepted u.

This disjoint path test requires knowing the paths through which updates are transmitted. Our gossip protocols therefore communicate not just updates but proposals, which contain an update and a list of hosts through which that proposal has been forwarded. We write (u, p), for the proposal containing update u that has traversed gossip path p; we write $\emptyset$ for the empty gossip path and use square brackets to write a non-empty gossip path (*e.g.* $[H, J, K]$); we write π::H to denote proposal π with host H appended to its gossip path. Two proposals are said to be disjoint iff, independent of the updates they contain, their gossip paths are disjoint.

The class of Path verification protocols is characterized as follows.

(3.1) Path Verification: For each host H, let $\mathcal{P}_H$ be the set of proposals that contains a given proposal π iff

(i) $\pi = \pi' :: J$ and H received proposal π' from host J, or
(ii) $\pi = (u, \emptyset)$ and host H has accepted update u.

A protocol is a *path verification protocol* if and only if, for every uncorrupted host H, the following conditions hold.

(1) When H sends proposal π, $\pi \in \mathcal{P}_H$ holds.
(2) When H accepts update u, $\mathcal{P}_H$ contains a *satisfying set for update* u – a set of $t+1$ mutually disjoint proposals with update u.

Here and throughout, variables are subscripted by the host to which they are associated, and variables are presumed to be instantiated once per instance of the protocol. In particular there is a different instance of P_H for each instance of the diffusion protocol.

Note that Path Verification allows corrupted hosts to forge proposals they transmit to other hosts and, in particular, to forge a proposal's gossip path. However, since the receiver is required to add the sender's identity to the path, a corrupted host cannot remove its own identity from the gossip path of a proposal it sends to another host. Because of this, the requirement that hosts accept only those updates for which they have $t+1$ mutually disjoint proposals ensures that Accept Safety holds. This is shown in the following theorem.

Theorem 3.1. *Path verification protocols satisfy Accept Safety.*

Proof. A proposal π in $\mathcal{P}_H$ can be traced back from receiver to sender, repeatedly, terminating either when it first reaches a corrupted host, or else when it reaches the host originating the proposal, which itself must have accepted u. Call the resulting sequence of hosts the *true path* of proposal π. A true path is defined to be *uncorrupted* if it contains all uncorrupted hosts, and *corrupted* otherwise. A proposal is said to be (un)corrupted iff its true path is (un)corrupted. Note that if proposal π is uncorrupted, then its gossip path is equal to its true path.

To prove that Accept Safety is preserved, it suffices to show that if a non-source host H accepts an update u then some uncorrupted host has already accepted u. Condition (2) of Path Verification (3.1) requires that if H accepts update u, then $\mathcal{P}_H$ contains $t+1$ disjoint proposals with update u. Given the assumption of at most t corrupted hosts, we conclude that at least one of those proposals – call it π – must have no corrupted hosts on its gossip path. Therefore, the true path of π equals the proposal's gossip path and is also uncorrupted.

Thus, it suffices to show that whenever a proposal's true path is uncorrupted, then some uncorrupted host appearing on that path has accepted the update. Either the first host on the true path is a source host which, by definition, has accepted the update. Or, the first host on the true path is an uncorrupted host H' for which $(u, \emptyset)$ is in $\mathcal{P}_{H'}$ and, therefore, according to clause (ii) of the definition of Path Verification (3.1), host H' has accepted update u.

We have thus proved that some uncorrupted host has accepted u, and therefore Accept Safety holds.

3.1 Direct Diffusion

A naive Path Verification protocol would have each host H store every proposal it receives; that entire set of proposals would then be forwarded to any host that chooses H as a gossip partner. In such a protocol, H would (approximately) double the set of proposals it stores and transmits each round, since in a given round H keeps all of the proposals it is already storing and adds all of the proposals being stored by its gossip partner. Message size and storage for proposals grow exponentially, rendering the protocol impractical.

In order to avoid these exponential costs, hosts must limit the number of proposals they store and transmit. One approach to discarding proposals comes from direct verification. Direct verification can be understood as a special case of Path Verification where all proposals with gossip paths of length greater than 1 are discarded. A host H accepts an update once it has gossiped with $t+1$ or more hosts J for which $\mathcal{P}_J$ contains $(u, \emptyset)$. By definition, if $\mathcal{P}_J$ contains $(u, \emptyset)$, then either J is corrupted or J has accepted u. Thus, H only accepts an update after gossiping with $t+1$ hosts that claim to have accepted u, satisfying the definition of direct verification.

Direct Diffusion, shown below, is a simple direct verification protocol recast as a Path Verification protocol. Variable d_H is initially set to *null proposal* $\perp$, unless H is a source host, in which case d_H is initially set to $(u, \emptyset)$.

(3.2) Direct Diffusion:[6]
Let J be H's gossip partner.

(1) If $d_J = (u, \emptyset)$ for some update u, then $\mathcal{D}_H := \mathcal{D}_H \cup \{(u, [J])\}$
(2) If $\mathcal{D}_H$ contains a satisfying set for some update u, then accept update u and set d_H to $(u, \emptyset)$

[6] Direct Diffusion can be divided into a selection protocol that sets the value of d_H and a sampling protocol that collects values in $\mathcal{D}_H$. Such a division, however, is not instructive for analyzing this protocol. Further discussion of selection and sampling protocols is deferred to Sect. 3.2.

In the above, whenever the value of a variable v belonging to a remote host J is referenced, it is presumed to be the value of v from the previous round; if J is corrupted, then the value is arbitrary. Each host writes only to its own local variables.

Direct Diffusion was described by Malkhi *et al.* in [9]. The following lower bound on the expected diffusion time of Direct Diffusion is implied by their analysis for $2 < t < n/4$:

$$\Omega\left(t\left(\frac{n}{k}\right)^{(1-1/2t)}\right).$$

This bound is polynomial in n, and it becomes nearly linear in n as t gets larger (so $1 - 1/2t$ approaches 1). In contrast, as shown in [2,16,7], diffusion protocols that don't tolerate malicious failures (*e.g.*, Direct Diffusion with $t = 0$) have diffusion times logarithmic in the number of hosts.

Malkhi *et al.* [9] noted that the long diffusion time of Direct Diffusion can be attributed to the early behavior of the protocol. To better understand this insight, let n_u be the number of updated hosts in a given round. In [9] it is shown that once n_u/n is greater than $1/2$, the number of rounds remaining until $n_u = n$ holds is $O(log(n))$. Since the overall diffusion time of Direct Diffusion for $t > 0$ is polynomial in n, the bulk of the diffusion time must be spent in the early phase, when n_u/n is less than $1/2$.

The ℓ-tree-random protocol of [9] is a direct verification protocol that improves upon the diffusion time of Direct Diffusion by changing how hosts choose gossip partners. As shown in [9], however, for direct verification protocols there is a tradeoff between diffusion time and host load. Improvements to diffusion time are thus accompanied by a deterioration in host load. Moreover, as discussed in Sect. 6, our experiments suggest that the best diffusion times achievable by ℓ-Tree-Random are significantly longer than those of the fastest algorithms shown in this paper.

Another problem with the ℓ-Tree-Random protocol is that benign fault tolerance is sacrificed. Processes are organized into blocks of ℓ processes, arranged in a binary tree. If the hosts in one or more of these blocks fail or are partitioned, then this tree becomes disconnected and distribution of updates becomes impossible. This is in contrast to Direct Diffusion, where no matter how many benign failures there are, a collection of $t + 1$ source hosts can always disseminate an update throughout the system.

3.2 Youngest Diffusion

In order to improve upon the performance of Direct Diffusion, we seek a better method for discarding proposals. Discarding the wrong proposals could make it unlikely that a host ever stores the $t + 1$ disjoint proposals needed to accept an update. Indeed, corrupted hosts might be able to generate proposals that are unlikely to be discarded, thereby crowding-out other proposals and preventing or delaying system-wide dissemination of the update.

Consider a selection protocol that each round selects a proposal π_H from among the proposals in $\mathcal{P}_H$ (as defined in Path Verification (3.1)). The choice of π_H might change from round to round; unless otherwise specified, the initial value of π_H is $\bot$. Consider a sampling protocol that collects for each host values of π_H from a random subset of at least $t + 1$ hosts H. A host looks for a satisfying set from among the non-$\bot$ proposals obtained using the sampling protocol.

Refining the Selection Protocol. Since the sampling protocol obtains values of π_H from randomly chosen hosts H, the sampling protocol retrieves a significant number of non-$\bot$ proposals only when a significant fraction of hosts have non-$\bot$ values for π_H. Moreover, for there to be a high probability that proposals gathered by the sampling protocol contain a satisfying set (as required for termination), the selection protocol must ensure that non-$\bot$ proposals held by hosts chosen at random are likely to be disjoint.

Accordingly, a selection protocol should ensure that each host H chooses non-$\bot$ values for π_H early in the protocol's execution and that those values are likely to be disjoint. To this end, we propose selection protocol *Youngest Selection*, where π_H stores the youngest, *i.e.* minimum-age, proposal seen by H. The *age* of a proposal is a natural number equal to the number of rounds since the proposal originated.[7]

Youngest Selection is described formally below. Variable age_H holds the age that host H records for proposal π_H. $\bot :: J$ is defined to be $\bot$, and if $\pi_J = \bot$ then age_J is defined to be ∞. Accordingly, π_H is initialized to $\bot$ and age_H is initialized to ∞ for all hosts except source hosts. For source hosts, π_H is initially set to $(u, \emptyset)$ and age_H is set to 0.

The following defines the behavior of Youngest Selection for a host H in a given round.

(3.3) Youngest Selection:
Let J be H's gossip partner.
If H is not a source host, then

(1) $\pi_H := \begin{cases} \pi_H & \text{if } age_H < age_J, \\ \pi_J :: J & \text{otherwise.} \end{cases}$

(2) $age_H := \min(age_H, age_J) + 1$

Youngest Selection has a number of advantages as a selection protocol. First, Youngest Selection causes a host H to select non-$\bot$ values for π_H within a small (*viz* logarithmic in the number n of hosts) number of rounds. This is because H chooses a non-$\bot$ value for π_H once it gossips with a single host J that has already chosen a non-$\bot$ value for π_J. As shown in [4,14], a gossip protocol that spreads an update in this way requires only $O(\log n)$ rounds until the update is disseminated throughout the system.

A second advantage of Youngest Selection is that low-age proposals tend to have short gossip paths. This increases the probability that youngest proposals held by different hosts are disjoint. Given the utility of short paths, it might seem more natural for each host to select the shortest path it has seen, rather than selecting the youngest. The problem with selecting the shortest path is that once a host has selected a particularly short path for π_H, that path is unlikely to change quickly. Thus, an unlucky distribution of proposals that persists for a long time is possible, making it difficult to find a satisfying

[7] As noted in the proof of Theorem 3.1, a proposal can be traced back from receiver to sender over multiple rounds, eventually reaching the host at which the proposal originated. It is from this round that the age of the proposal is measured.

set. Under Youngest Selection, however, proposals perpetually get older and less attractive, so no distribution of proposals persists for long. Note that line (1) of Youngest Selection (3.3) specifies that π_H is kept only if $age_H < age_J$, rather than if $age_H \leq age_J$, for similar reasons – that is, to encourage proposals to change quickly.

The most important feature of Youngest Selection is that corrupted hosts have only a limited ability to forge the age of a proposal. The worst a corrupted host might do is perpetuate its own proposals by setting the age of proposals it distributes to zero. The presence of a corrupted host is thus, at worst equivalent to an extra source host distributing the wrong update. (This last point is discussed in more detail in Sect. 5.2.)

A seemingly natural improvement to Youngest Selection would be to have any host that has accepted update u thereupon start to act as if it were a source host for u, setting π_H to $(u, \emptyset)$ and age_H to 0. The resulting protocol, which we call Promiscuous Youngest Diffusion, is in fact a Path Verification protocol, but it turns out not to have the well-defined and limited worst case behavior just described. However, in Sect. 3.3 we will develop a protocol, called Hybrid Diffusion, whose performance is very close to that of Promiscuous Youngest Diffusion in the normal case while maintaining a well-defined worst case behavior. The explanation of why the worst-case behavior for Promiscuous Youngest Diffusion is problematic is given in Appendix C.

Refining the Sampling Protocol. A straightforward sampling protocol has each host H obtain π_J directly from each gossip partner J. In order to bound the storage required, each host stores at most S proposals, where S is some constant. The resulting protocol, called Simple Sampling, is shown below. Queue $\mathcal{Y}_H$ contains the S most recent proposals collected by H, and it is initially empty.

(3.4) Simple Sampling:

```
Let $J$ be $H$'s gossip partner.
If $H$ is not a source host:
(1) If $\pi_J \neq \perp$, then:
(2)     $\mathcal{Y}_H.\mathtt{enqueue}(\pi_J :: J)$
(3)     if $|\mathcal{Y}_H| > \mathsf{S}$ then $\mathcal{Y}_H.\mathtt{dequeue}()$
```

More sophisticated sampling protocols are discussed in Sect. 4.

Youngest Selection and Simple Sampling can be combined into a single protocol, called Youngest Diffusion. The following defines the behavior of Youngest Diffusion in any given round.

(3.5) Youngest Diffusion:

```
(1) Execute Youngest Selection (3.3)
(2) Execute Simple Sampling (3.4)
(3) If $\mathcal{Y}_H$ contains a satisfying subset for update $u$, then accept $u$
```

Youngest Diffusion satisfies Accept Safety and Accept Liveness, and therefore satisfies Diffusion Problem (2.1). This is shown below.

Theorem 3.2. *Youngest Diffusion satisfies Accept Safety of Diffusion Problem (2.1)*

Proof. Given Theorem 3.1, it suffices to prove that Youngest Diffusion (3.5) is an instance of Path Verification (3.1). So, we show that actions taken by Youngest Diffusion (3.5) are consistent with conditions (1) and (2) of Path Verification (3.1):

- Condition (1) of Path Verification (3.1) is satisfied because $\pi_H \in \mathcal{P}_H$ holds throughout execution, and π_H is the only proposal transmitted by H.
- Condition (2) of Path Verification (3.1) is satisfied because $\mathcal{Y}_H \subseteq \mathcal{P}_H$ holds, so the decision to accept an update u in line (3.2) of the protocol implies that $\mathcal{P}_H$ contains $t+1$ mutually disjoint proposals with update u.

Theorem 3.3. *If $\mathsf{S} > t$ then Youngest Diffusion (3.5) satisfies Accept Liveness of Diffusion Problem (2.1).*

Proof: In Youngest Diffusion, an uncorrupted host will accept an update if it chooses as its gossip partners $t+1$ different source hosts, one after the other, and if $\mathsf{S} > t$. There is a nonzero probability of this occurring in any sequence of $t+1$ rounds. Therefore, the probability that this never occurs converges to zero as the number of rounds approaches infinity. Thus, the probability that an uncorrupted host eventually accepts an update is 1. And therefore, the probability that all uncorrupted hosts eventually accept an update is also 1.

3.3 Hybrid Diffusion

Section 3.1 notes that, although Direct Diffusion has a very long diffusion time, the performance of Direct Diffusion improves when n_u/n is large. Indeed, once n_u/n gets sufficiently large, Direct Diffusion becomes more efficient than Youngest Diffusion: once a certain value of n_u/n is reached, the last few non-updated hosts will become updated sooner under Direct Diffusion than under Youngest Diffusion. This is because, as shown in Sect. 4, finding $t+1$ disjoint proposals through sampling can take a significant number of rounds, even under optimistic assumptions about the likelihood that π_H at different hosts H are disjoint. For Direct Diffusion, however, once n_u/n is close to 1, almost every gossip partner chosen by any given host will have accepted an update, so the remaining hosts will accept updates more quickly.

By composing Youngest Diffusion and Direct Diffusion, we can construct a diffusion protocol that does better than either – it performs at least as well as Youngest Diffusion when n_u is small, and at least as well as Direct Diffusion when n_u is large. Our Hybrid Diffusion protocol is such a composition. It involves running Direct Diffusion and Youngest Diffusion side by side, as follows.

(3.6) Hybrid Diffusion:

```
(1) Execute Youngest Diffusion (3.5)
(2) Execute Direct Diffusion (3.2)
(3) if $\mathcal{Y}_H \cup \mathcal{D}_H$ contains a satisfying set for update $u$, then accept $u$
```

Note that the tests in line (3.2) of Youngest Diffusion (3.5) and line (3.1) of Direct Diffusion (3.2) are redundant in light of line (3.3) of Hybrid Diffusion, so these tests can be removed for Hybrid Diffusion. Also note that lines (3.3) and (3.3) are executed using the same gossip partner.

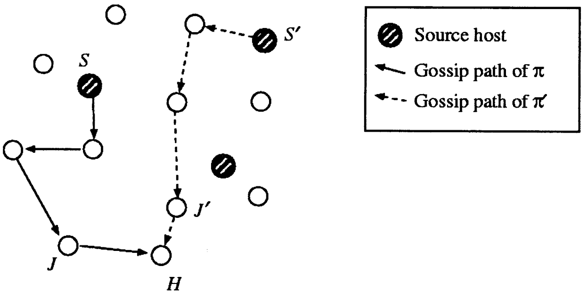

Fig. 2. Transmission of disjoint proposals

There is a sense in which Hybrid Diffusion is more efficient than either Youngest Diffusion or Direct Diffusion, since if a host would accept a given update at a given round under Direct Diffusion or under Youngest Diffusion, then that host will have accepted the update by that round under Hybrid Diffusion as well.[8] As such, the fact that Youngest Diffusion and Direct Diffusion satisfy Accept Liveness implies that Hybrid Diffusion satisfies Accept Liveness.[9] Since Youngest Diffusion and Direct Diffusion are Path Verification protocols, Hybrid Diffusion is a Path Verification protocol, and, therefore, it must satisfy Accept Safety. Thus, we have:

Theorem 3.4. *Hybrid Diffusion satisfies Diffusion Problem (2.1).*

4 Efficient Sampling

The diffusion time of Youngest Diffusion (3.5) is based, in part, on the number of rounds it takes under Youngest Selection (3.3) for hosts to select non-$\perp$ values for π_H, and, in part, on the number of rounds required under Simple Sampling (3.4) for a host to collect a satisfying set once a significant subset of hosts have selected non-$\perp$ values for π_H. So, it is instructive to estimate the performance of Simple Sampling. This estimate gives insight that leads to a new sampling protocol with improved performance (at the expense of increased message size).

4.1 Estimating the performance of Simple Sampling

Define the *first host* and *last host* of a proposal to be the first and last hosts respectively on that proposal's gossip path. Figure 2 depicts a host H that has received two disjoint proposals, π and π', with first hosts S and S' and last hosts J and J' respectively. That π and π' are disjoint trivially implies that $S \neq S'$ and $J \neq J'$ hold, *i.e.*, that π and π' have different first and last hosts. We refer to a pair of proposals whose gossip paths have different first and last hosts as *end-disjoint*.

A set of disjoint proposals are necessarily end-disjoint. Because youngest proposals tend to have short gossip paths, a pair of youngest proposals that are end-disjoint are more likely (though not guaranteed) to be disjoint. Thus the number of rounds that it takes for a host to obtain a set of end-disjoint youngest proposals is correlated with the number of rounds required to find a set of disjoint proposals.

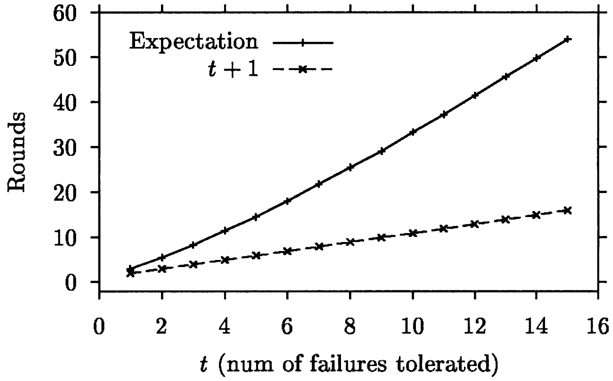

Fig. 3. Expected number of rounds required for sampling in Youngest Diffusion. The minimum of $t + 1$ required rounds is also plotted

To provide a lower bound on the number of rounds required to collect $t + 1$ end-disjoint proposals in the absence of corrupted hosts, we make the following optimistic assumptions:

1. For all hosts H, proposal π_H is not $\perp$.
2. S is unbounded.
3. n is sufficiently large that the probability of a host choosing the same gossip partner twice in close succession is negligible.
4. The first host of the proposal stored in π_H for a randomly chosen host H is an independent random variable distributed uniformly over the set of source hosts.

By Simple Sampling (3.4) and assumption 1, a non-$\perp$ proposal $\pi_J :: J$ is added to $\mathcal{Y}_H$ every round, where J is H's gossip partner in the given round. By assumption 2, none of these proposals are discarded. Assumption 3 implies that H's gossip partners are (with high probability) distinct. Since the last host of $\pi_J :: J$ is J, the proposals in $\mathcal{Y}_H$ must (with high probability) have distinct last hosts.

Assuming that the proposals in $\mathcal{Y}_H$ have distinct last hosts, $\mathcal{Y}_H$ contains a set of $t + 1$ end-disjoint proposals if and only if $\mathcal{Y}_H$ contains a set of proposals with distinct first hosts. First hosts are chosen from among the k source hosts, and as such are much less likely to be distinct than the last hosts, which can be any host in the system.

By assumption 4, the first hosts of the proposals in $\mathcal{Y}_H$ are chosen uniformly and at random. Thus, the problem of finding $t + 1$ proposals with distinct first hosts is equivalent to the problem of randomly choosing $t + 1$ different values from a set of k possibilities. This problem is known as the $(t + 1, k)$ coupon collector's problem [3, p. 11]. Therefore, under assumptions 1 through 4, the number of rounds required for a host to obtain $t + 1$ end-disjoint proposals when there are k source hosts can be approximated by the number of iterations required by the $(t + 1, k)$ coupon collector's problem.

Figure 3 graphs the expected number of iterations required to solve the $(t+1, k)$ coupon collector's problem, for $k = t+1$. This value of k is the minimum allowed, and it is also the case with the largest expected number of rounds, since it becomes

[8] However, message size with Hybrid Diffusion is larger than message size with either Youngest or Direct Diffusion. So, in that sense, Hybrid Diffusion is less efficient.

[9] Actually, it suffices that only one of Youngest Diffusion and Direct Diffusion satisfy Accept Liveness.

easier to collect $t + 1$ distinct values the more possibilities there are to sample from. Note also that the minimum number of rounds required for solving the $(t+1, k)$ coupon collector's problem is $t+1$. This minimum is also graphed in Fig. 3. Since our lower bound is based on the coupon collector's problem, the graph suggests that Simple Sampling requires far more than the $t + 1$ round minimum in order to obtain $t + 1$ disjoint proposals. For $t = 15$, for example, the $(16, 16)$ coupon collector problem is expected to take more than 50 rounds. Thus, the expected number of rounds required to find $t + 1$ disjoint proposals is more than 3 times the minimum (16).

Clearly, there is significant room to improve the performance of Simple Sampling, and consequently the performance of Youngest Diffusion. Hybrid Diffusion would benefit from improving the sampling protocol as well, since when n_u is small, Hybrid Diffusion behaves similarly to Youngest Diffusion.

4.2 Bundle Sampling

The performance of Simple Sampling (3.4) is limited by the fact that a given host can obtain at most one sample per round. Better performance can be achieved by a distributed sampling protocol in which hosts share their samples, thereby allowing an individual host to get multiple samples per round. Thus, a host H gossiping with a host J could obtain not only proposal π_J, but also some values collected by J from yet other hosts.

In designing such a distributed sampling protocol, a decision must be made about which samples to keep and which to discard. Ideally, samples would be gathered quickly and discarded quickly so that if a host is unlucky and the samples it gathers do not contain a satisfying set, then those samples will be replaced as soon as possible. This argues against using path length, for example, as a basis for discarding samples because the samples that remained would then become less and less likely to change over time. Furthermore, we want to limit the influence that corrupted hosts have on the set of samples gathered, which argues against using any property of the sampled proposals themselves as a basis for discarding them – the proposal's age, for example, could be manipulated by a corrupted host to make those proposals it contributes more desirable, thereby displacing legitimate proposals.

So we consider protocols that decide which proposals to discard according to *sample age*, where the sample age of a proposal is the number of rounds since that proposal was obtained by the sampling protocol. Consider a sampling protocol that discards proposals with sample age larger than some bound SA. Initially, proposals are gathered as quickly as possible, since in the first SA rounds, no proposals are discarded. In subsequent rounds, proposals are discarded in order of their sample age, so no proposal is stored for more than SA rounds. Discarding proposals based on sample age also limits the influence of corrupted hosts, as shown in Sect. 4.3.

A *sample* will be a pair (π, sAge), where π is a proposal and *sAge* is the sample age. A sample is initially created by the *originating host* H as a pair $(\pi_H, 0)$ and can be transmitted from one host to another, with the gossip path of π extended accordingly. Whether or not the proposal is forwarded, *sAge* is incremented each round.

Bundle Accumulation is a sampling protocol based on the above ideas. Each host H maintains a set of samples bundle_H, called a *sample bundle*. When H gossips with another host J, H adds bundle_J to bundle_H, along with a sample holding the current value of π_H. H then discards from bundle_H any samples with sample age greater than SA.

Bundle Accumulation uses the following function to update the age of each sample and to discard old samples.

(4.1) $\mathsf{UpdateBundle}(\mathit{bundle}) =$
$\{(\pi, \mathit{sAge} + 1) \mid (\pi, \mathit{sAge}) \in \mathit{bundle} \ \wedge \ \mathit{sAge} < \mathsf{SA}\}$

When a host H receives a bundle from another host J, it appends J to every sample in host J's bundle. Accordingly, define $\mathit{bundle} :: J$ to be $\{(\pi :: J, \mathit{sAge}) \mid (\pi, \mathit{sAge}) \in \mathit{bundle}\}$. The following summarizes the behavior of Bundle Accumulation for a host H in a given round. bundle_H is assumed to be initially empty

(4.2) Bundle Accumulation:
Let J be H's gossip partner.

```
(1) $\mathit{bundle}_H :=$
        $\mathsf{UpdateBundle}(\mathit{bundle}_H \cup \mathit{bundle}_J :: J)$
          $\cup \ \{(\pi_H, 0)\}$
```

As noted above, in the first SA rounds of the protocol, no proposals are older than SA, so no proposals are discarded by UpdateBundle. As such, the size of each host's bundle grows exponentially, approximately doubling every round. After SA rounds, the size of each host's bundle stabilizes. Thus, the size of each sample bundle is exponential in SA.

Samples collected by Bundle Accumulation (4.2) have randomly chosen originating hosts. However, the paths taken by the samples since leaving the originating host are not particularly diverse – samples in a bundle are collected directly from no more than SA different hosts (*i.e.*, one host per round), and thus the proposals contained therein can have no more than SA different last hosts. Thus, in order to obtain a set of $t + 1$ disjoint proposals, SA must be larger than t. This, however, is an inefficient way of obtaining proposals with diverse last hosts, since it would require $O(2^t)$ bundles per sample bundle, which can become unmanageable for even moderate values of t.

We need a way of collecting proposals that have both diverse last hosts and diverse originating hosts, without requiring too many samples. Simple Sampling (3.4) ensures that proposals are gathered with multiple different last hosts by keeping a queue of proposals chosen from successive gossip partners. Since gossip partners are chosen at random, the last hosts of the proposals gathered are likely to be different. Following a similar approach, we can require each host to store a bounded-length queue of sample bundles taken from successive gossip partners. This ensures diversity of last hosts among the proposals from different bundles in the queue. Moreover, if the length of the queue is on the order of t, then the number of proposals stored is $O(t \cdot 2^{\mathsf{SA}})$, which can be kept small by keeping SA small.

The resulting protocol is called Bundle Sampling. Let $\mathcal{B}_H$ be the queue containing the most recently collected bundles, and $\mathsf{S} > t$ the maximum number of bundles stored in the queue at an individual host. The following defines Bundle Sampling for a host H in a given round. $\mathcal{B}_H$ is assumed to be initially empty.

(4.3) Bundle Sampling:
Let J be H's gossip partner.

(1) $\mathcal{B}_H := \mathcal{B}_H.\texttt{enqueue}(\mathit{bundle}_J :: J)$
(2) If $|\mathcal{B}_H| > \mathsf{S}$ then $\mathcal{B}_H.\texttt{dequeue}()$

Note that the above protocol requires Bundle Accumulation to choose values for *bundle*$_H$. Bundle Sampling can be used in place of Simple Sampling in Youngest Diffusion (3.5), as follows.

(4.4) Youngest Diffusion with Bundle Sampling:

(1) Execute Youngest Selection (3.3)
(2) Execute Bundle Accumulation (4.2)
(3) Execute Bundle Sampling (4.3)
(4) If $\mathcal{B}_H$ contains a satisfying subset for update u, then accept u

In the above protocol, Youngest Selection is used to determine the value of π_J for each host J. These values are then used by Bundle Accumulation to compute *bundle*$_J$ for each host J, and those values are in turn used by Bundle Sampling to compute $\mathcal{B}_J$ for each host J. Note that, in a given round, a host H requests both π_J and *bundle*$_J$ from its gossip partner J.

Bundle Sampling can also be applied to Direct Diffusion, except that instead of samples containing values of π_H, samples would contain values of d_H. It follows that Bundle Sampling can be applied to Hybrid Diffusion as well.

Theorem 4.1. *Youngest Diffusion, Direct Diffusion, and Hybrid Diffusion with Bundle Sampling all satisfy Path Verification.*

The proof of this theorem is found in Appendix A.1.

4.3 Fault tolerance of Bundle Sampling

Because we are interested in using sampling as part of Byzantine-fault tolerant protocols, it is necessary to assess the behavior of our sampling protocols in the presence of corrupted hosts. Define the *sample path* of a sample to be the sequence of hosts along which the sample was transmitted from its originating host to some host at which the sample resides. Note that the sample path is the tail end of the gossip path from the proposal held by that sample. This is because a sample path only records the hosts through which a proposal was transmitted since it was included in a sample. The gossip path of a proposal, however, also records the hosts through which the proposal was transmitted before that point.

Define a sample held by host H to be *uncorrupted* iff its sample path does not contain any corrupted hosts. A sample can be uncorrupted even if the proposal it contains is corrupted, since the gossip path of some proposal π may contain a corrupted host even though the sample path of a sample containing π may not. This corresponds to the situation where a corrupted proposal held by an uncorrupted host is obtained by the sampling protocol and then transmitted through a sequence of uncorrupted hosts. A measure of the influence of corrupted hosts is the expected number of corrupted samples contained in a given uncorrupted host's bundle. The following theorem bounds this expectation.

Theorem 4.2. *Let f be the actual number of corrupted hosts,*[10] *and let H be an uncorrupted host. If $0 \leq a \leq \mathsf{SA}$ and $a \leq r$, then the expected number of uncorrupted samples with sample age a in bundle$_H$ at round r is $(2 - f/n)^a$, and the actual number of uncorrupted samples of sample age a never exceeds 2^a.*

The proof of Theorem 4.2 is found in Appendix A.2.

Note that Theorem 4.2 does not bound the number of corrupted samples in *bundle*$_H$. Indeed, as described, Bundle Sampling allows a corrupted host J to store arbitrarily many proposals in *bundle*$_J$. If an uncorrupted host chooses J as a gossip partner and therefore integrates *bundle*$_J$ into *bundle*$_H$, an unbounded number of corrupted proposals will be integrated into *bundle*$_H$, even though in the absence of failures, no host's bundle can contain more than 2^a samples of sample age a. To avoid this problem, uncorrupted hosts must refuse to accept sample bundles with more than 2^a samples of any given sample age a. This ensures that, even in the presence of corrupted hosts, an uncorrupted host's sample bundle will contain no more than 2^a samples of age a.

If the above rule for discarding sample bundles is used, then the expected ratio of uncorrupted samples to total number of samples of a given age a is bounded below by

$$(2 - f/n)^a/2^a = (1 - f/2n)^a.$$

Here the numerator is the bound on the number of uncorrupted samples from Theorem 4.2, and the denominator is the bound on the total number of samples. This ratio approaches zero as a increases. However, for f small relative to n and moderate values of a, this ratio is favorable. For example, for $n = 100$ $f = 10$ and $a = 4$, each sample bundle will contain 16 samples of age 4, 81% of which are expected to be uncorrupted. This suggests that Bundle Sampling is reasonably efficient in terms of the number of uncorrupted proposals obtained.

4.4 Choosing constants

Bundle Sampling is parameterized by SA, the maximum sample age of the proposals stored in *bundle*$_H$, and S, the maximum number of bundles stored in $\mathcal{B}_H$. In the following, we investigate values for these constants by considering how different values affect the number of rounds of sampling required to obtain a satisfying set.

As in Sect. 4.1, we use the number of rounds required to find a set of mutually end-disjoint proposals as a guide to the number of rounds required to find a satisfying set. We make simplifying assumptions that each bundle contains b proposals and that the source host of each proposal in a bundle can be modeled as an independent random variable distributed evenly over the k source hosts. Finally, we assume that each bundle in $\mathcal{B}_H$ is obtained from a different host. Our analysis then

[10] In contrast to t, which is the number corrupted hosts tolerated by a given protocol.

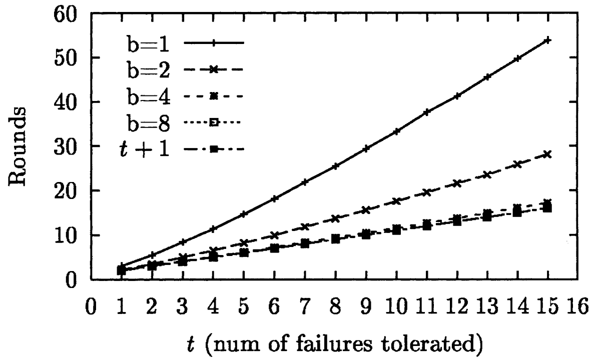

Fig. 4. Average number of rounds required to complete b-ary coupon collector's problem. Results are shown for $b = 1, 2, 4$ and 8. Each data point was generated from 10,000 iterations

looks at the case where the number k of source hosts is $t + 1$, the minimum number of source hosts allowed by Diffusion Problem (2.1).

Under the above assumptions, the number of rounds required for a host to find a set of end-disjoint proposals using Bundle Sampling is the same as the number of rounds required for the termination of the following process, which we call the b-ary coupon collector's problem: Each round, a collection of b source hosts are selected at random (with replacement) from the set of k source hosts. These source hosts represent the first hosts of the different proposals stored in a given bundle. The process terminates as soon as any set of $t+1$ different source hosts can be selected, one from each collection.

Unable to derive a closed form solution, we ran a Monte-Carlo simulation in order to estimate the expected number of rounds required for our b-ary coupon collector's problem to terminate. Figure 4 shows the average number of rounds over 10,000 iterations. Results are shown for b equal to 1, 2, 4 and 8. For $b = 1$, the process is exactly the coupon collector's problem, and the results are the same as those shown in Fig. 3. As in that case, the process cannot terminate in fewer than $t+1$ rounds. The fact that the line for $b = 4$ is within a few rounds of the minimal $t + 1$ rounds, suggests that, for the values of t shown, $b = 4$ is near optimal, and that further increasing b will do little to improve expected diffusion time.

We now consider how to choose SA in order to ensure that each bundle contains $b = 4$ uncorrupted proposals. In the absence of failures, Theorem 4.2 implies that if SA $=$ 2, then each bundle contains $2^{\mathsf{SA}} = 4$ uncorrupted samples of sample age SA.[11] Corrupted hosts can reduce the number of uncorrupted proposals in a sample bundle in two ways. First, as shown in Theorem 4.2, the number of uncorrupted samples collected decreases as t/n increases. However, for small values of SA and t/n, this effect is small.

The second way in which corrupted hosts reduce the number of uncorrupted proposals in a sample bundle is by corrupting the proposals π_H being sampled. If a given proportion ρ of the values of π_H are corrupted, then, for any randomly selected subset of those proposals, the expected fraction of corrupted proposals will be ρ as well. Thus, in order to estimate the number of uncorrupted proposals obtained by sampling, we must estimate the proportion of underlying proposals that are uncorrupted.

As argued in Sect. 5.2, the worst behavior by corrupted hosts is to act like source hosts, *i.e*, to distribute (corrupted) proposals with age 0. Accordingly, consider the case where there are $2t+1$ hosts acting as source hosts, $t+1$ of which are uncorrupted and t of which are corrupted. Thus, almost half of the hosts acting as source hosts are corrupted, suggesting that for approximately half of the hosts H, proposal π_H is corrupted. As argued above, this implies that roughly half of the proposals in any given bundle will be corrupted. Thus, by setting SA to 3, we ensure that there are $2^{\mathsf{SA}} = 8$ maximum-age proposals, and therefore approximately $8/2 = 4$ uncorrupted samples. This is equal to the desired value of b computed above, *i.e.*, the desired number of samples per sample bundle.

It remains to select a value for S, the number of bundles stored by each host. S should be chosen such that most hosts do not require more than S rounds before they can obtain a satisfying set of proposals – otherwise, the dropping of sample bundles due to S might slow down the protocol.

In order to choose a sufficiently large value for S, we must consider the probability that a host will require more than S rounds. We therefore consider the standard deviation of the number of rounds required for the b-ary coupon collector's problem. For $b = 4$, our Monte-Carlo simulations found that $2(t + 1)$ is within 5 standards of deviation of the expected number of rounds required to complete the b-ary coupon collector's problem. By Chebyshev's Inequality [3, p. 233], the probability that more than $2(t + 1)$ rounds will be required is no more than 4%. Insofar as the b-ary coupon collector's problem is a good model for the number of rounds of sampling required to obtain a set of end-disjoint proposals, this suggests that $2(t + 1)$ is a reasonable setting for S.

The above analysis presumes that a given host chooses S distinct gossip partners in a row. In fact, it is sufficient that almost all of the S gossip partners chosen by a given host are distinct. It is easy to show that the expected number of distinct hosts among S randomly selected gossip partners is bounded by $\mathsf{S} - \frac{\mathsf{S}^2-\mathsf{S}}{2n-2}$. Thus, when S, and therefore t, is on the order of $\sqrt{n}$, the expected number of distinct hosts is only a small constant away from S, and the above analysis applies.

5 Estimating performance

5.1 Host load and message size

The gossip protocols described so far share two features. First, they are all *pull* protocols [2], meaning that the host initiating a gossip requests information from its gossip partner. Second, gossip partners are selected uniformly and at random. We refer to the class of protocols sharing these two properties as *uniform-pull* gossip protocols. In any uniform-pull gossip protocol, the expected number of requests each host receives is one. Thus, in the absence of failures, the expected host load for uniform-pull gossip protocols is $O(1)$.

Even though messages from corrupted hosts are not counted in host load, corrupted hosts can affect host load by causing uncorrupted hosts to respond to spurious requests. The

[11] Since a bundle contains samples of all ages between 0 and SA, there are in fact $2^{\mathsf{SA}+1} - 1$ different samples. However, we only count the 2^{SA} maximum-age samples, because samples of different ages are not necessarily chosen independently.

worst case has every corrupted host sending a request to every uncorrupted host in each round. Accordingly, each host responds to each of the t corrupted hosts, leading to an expected host load of $O(t)$. This analysis assumes that hosts refuse to respond to more than one request per round from the same host.[12]

Message size. In order to bound message size, we must bound both the number of proposals per message and the length of each proposal's gossip path. Note that the length of the gossip path of an uncorrupted proposal π_H is bounded by age_H. It thus suffices to consider how age_H can be bounded.

Recall, age_H records the age of the youngest path received by H. Accordingly, no matter how corrupted hosts behave, if there is a sequence of gossips leading from some source host to host H involving only uncorrupted hosts in the last m rounds, then age_H must be no greater than m. This is shown more formally in Appendix B.1.

Define *simple diffusion* to be the special case of Direct Diffusion where $t = 0$, so no Byzantine failures are tolerated. Under simple diffusion, if H has accepted an update by round r, then there is necessarily a sequence of gossips leading from some source host to H. This means that age_H must be bounded by m. If within m rounds all hosts would accept an update by round r, then for all H, the value of age_H is bounded by m. Thus, a bound on the diffusion time of simple diffusion serves as a bound on age_H for Youngest Path Diffusion and Hybrid Diffusion.

The above analysis bounds the length of π_H's gossip path only if H is uncorrupted. If H is corrupted, then π_H's gossip path could be arbitrarily long. Since our protocols may retransmit proposals held by other, uncorrupted hosts, this means that the message size of our protocols is unbounded.

In order to limit message size, our protocols must reject proposals with excessively long gossip paths. Whether a proposal is excessively long can be determined from bounds on the diffusion time of simple diffusion, as discussed above. Asymptotic bounds of $O(\log n)$ for the diffusion time of simple diffusion are well known (see, for example [7]). Tight explicit bounds on the diffusion time of simple diffusion can be computed using the techniques found in [16]. In particular, the results of [16] allow the computation of a bound r_ϵ for any probability ϵ, such that the probability that there exists a host that has not been reached within r_ϵ rounds is ϵ.

In the case of Bundle Sampling, proposals should be discarded whose gossip paths exceed r_ϵ by SA or more. This is because proposals collected by Bundle Sampling may be transmitted from host to host for up to SA rounds, so these proposals may have gossip paths as much as SA longer than the gossip paths of the proposals π_H being sampled.

If proposals are discarded according to the above rules, then the size of an individual proposal is bounded by $\mathsf{SA} + r_\epsilon$, where r_ϵ is $O(\log n)$. It remains to bound the number of proposals sent per message. The only protocol where a host sends more than a constant number of proposals per round is Bundle Sampling. By Theorem 4.2, the number of age $a \leq \mathsf{SA}$ proposals per bundle is at most 2^a, so the total number of proposals per bundle is $2^{\mathsf{SA}+1} - 1$. For $\mathsf{SA} = 3$, the value proposed in Sect. 4.4, this leads to 15 proposals per bundle. In Hybrid Diffusion, where sampling is used to sample values of both π_H and d_H, this leads to 30 proposals per bundle.

In order to bound overall message size, bounds on the size of host names and the allowable size of updates must be obtained as well. Such bounds depend on specifics of the environment.

5.2 Diffusion time

Diffusion time has proven to be difficult to calculate. Our approach, therefore, is to estimate diffusion time by using simulation. Since our protocols are meant to tolerate malicious failures, it is important that these simulations capture the worst-case behavior of the protocols in the presence of corrupted hosts.

The protocols in Sects. 3 and 4 were designed so that the worst case behavior by corrupted hosts is well-defined and carefully circumscribed. In this worst-case behavior, corrupted hosts act as if they were source hosts that have accepted the wrong update $\hat{u}$ instead of the initial update u. In addition, if the sampling protocols of Sect. 4 are in use, then corrupted hosts maintain empty sample bundles. A proof that this behavior is in fact the worst case is found in Appendix B.2; the following gives the results of simulations of this worst-case behavior.

The simulator executes as a series of synchronous rounds. Each data point we report is an average of 10 experiments; standard deviation is shown with error bars. Figure 5 shows the performance of Direct Diffusion, Youngest Diffusion, and Hybrid Diffusion without Bundle Sampling, for networks of size $n = 100$ and 1000. In the $n = 100$ case, we see that Direct Diffusion and Youngest Diffusion perform similarly, while the number of rounds for completion of Hybrid Diffusion is approximately one third. In the $n = 1000$ case, the diffusion time of Direct Diffusion increases by nearly a factor of 10, while the diffusion time of Youngest Diffusion and Hybrid Diffusion increase by only a few rounds. The poor scaling of Direct Diffusion with n is consistent with the analysis of Sect. 3.1.

Figure 6 shows the diffusion time of Youngest Diffusion and Hybrid Diffusion, both with and without Bundle Sampling. Here, the maximum sample age SA of a sample bundle is set to 3 and S is set to $2t + 1$, in accordance with the discussion of Sect. 4.4. The improvement due to Bundle Sampling is quite large, particularly in the case of Youngest Diffusion. Indeed, Youngest Diffusion with Bundle Sampling has a shorter diffusion time than Hybrid Diffusion without Bundle Sampling. The diffusion time of Youngest Diffusion is decreased by almost a factor of 4, while the diffusion time of Hybrid Diffusion is decreased by a factor of just under 2.5.

Figure 7 shows the diffusion time of Hybrid Diffusion for $n = 100$, 1000 and 10000. Notably, the diffusion time deteriorates more quickly when $n = 100$ than it does for $n = 1000$ and 10000. Indeed, for $t = 10$, the diffusion time for $n = 100$ rises to nearly match the diffusion time for $n = 10000$. The reason that diffusion time increases more quickly

[12] More aggressive techniques for reducing expected host load are possible. For example, hosts could refuse to respond to multiple requests from the same host in close succession. Since it is unlikely that an uncorrupted host would send multiple requests to the same host, this should impact diffusion time minimally.

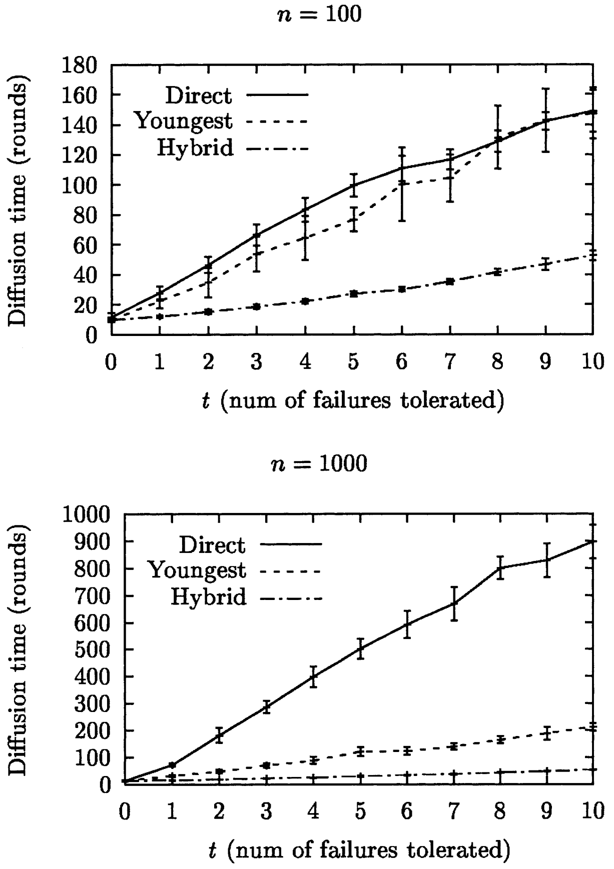

Fig. 5. Diffusion time of Direct Diffusion, Youngest Diffusion and Hybrid Diffusion for $n = 100$ and $n = 1000$ hosts

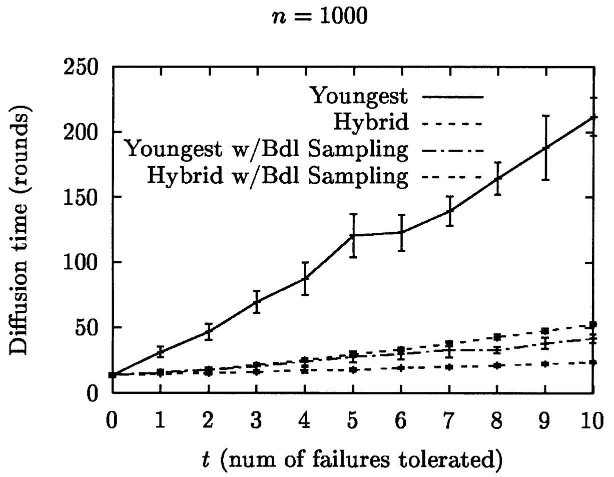

Fig. 6. Diffusion time of Youngest Diffusion and Hybrid Diffusion for $n = 1000$ hosts, with and without Bundle Sampling

when $n = 100$ then when $n = 1000$ or $n = 10000$ is that finding $t + 1$ mutually disjoint proposals is harder in smaller networks. Let L be the average length of a proposal's gossip path and let n be the number of hosts in the network. Then the smaller $L(t + 1)/n$ is, the more likely it becomes that a randomly chosen set of $t+1$ proposals will be disjoint. L only grows logarithmically with n, so $L(t+1)/n$ becomes smaller as n gets larger.

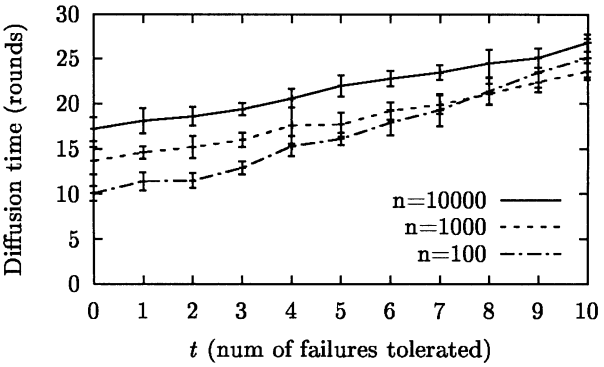

Fig. 7. Diffusion time of Hybrid Diffusion with Bundle Sampling for $n = 100$, $n = 1000$ and $n = 10000$

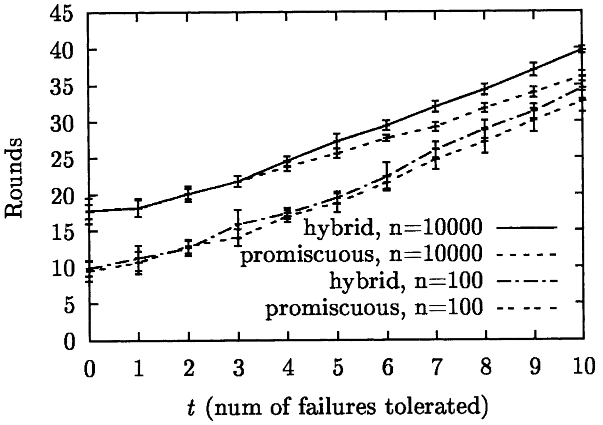

Fig. 8. Diffusion time of Hybrid Diffusion and Promiscuous Youngest Diffusion, both with Bundle Sampling. No faulty hosts are simulated

Figure 8 compares the performance of Hybrid Diffusion with Bundle Sampling and Promiscuous Youngest Diffusion with Bundle Sampling. The results shown reflect the behavior of the protocols in the absence of malicious hosts, since as noted in Sect. 3.2, we do not have a well-defined worst-case behavior for Promiscuous Youngest Diffusion. The performance of the two protocols is close, although as expected, Promiscuous Youngest Diffusion is clearly better, particularly for larger values of t. Note also that message size and storage requirements for Hybrid Diffusion are worse than for Promiscuous Youngest Diffusion, since Hybrid Diffusion needs to keep track of twice as many proposals as Promiscuous Youngest Diffusion. These experiments suggest that we pay a small but significant penalty for the well-understood worst-case behavior of Hybrid Diffusion.

Comparison to optimal diffusion time. Define an uncorrupted host H to be *touched* if either it is a source host or it has gossiped with at least one touched host. Note that for any Path Verification protocol, a host H can only have an uncorrupted non-$\perp$ proposal in $\mathcal{P}_H$ if H is touched. A satisfying set of proposals must have proposals with $t + 1$ distinct last hosts. Therefore, in order for a host to obtain a satisfying set of proposals, it must obtain a proposal from at least $t + 1$ different hosts. A host can therefore only become updated once it gossips with $t + 1$ touched hosts. As a result, the diffusion time

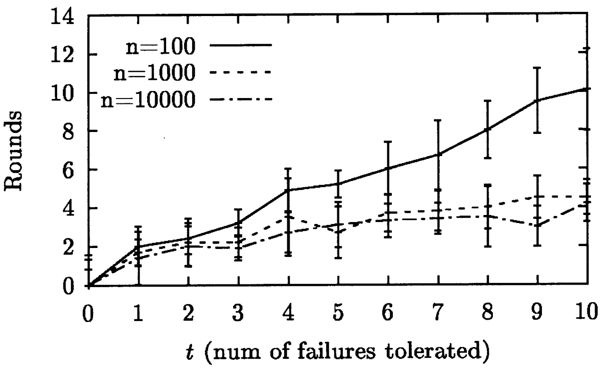

Fig. 9. Difference between diffusion time of Hybrid Diffusion and optimal diffusion time for $n = 100$, $n = 1000$ and $n = 10000$

of any Path Verification protocol can be no less than the time it takes for all hosts to become touched plus t more rounds for the last touched host to gossip with t other touched hosts.

Figure 9 shows the difference between the diffusion time of Hybrid Diffusion and the optimal performance described above: namely t plus the round at which the last host becomes touched.[13] The difference is zero for $t = 0$, since when no malicious failures are tolerated, a host becomes updated as soon as it becomes touched. For $n = 1000$ and 10000, the difference stays within 5 rounds, and the growth in the difference gets smaller as t gets larger. As discussed above, the diffusion time is markedly worse for $n = 100$.

These results show that for sufficiently large n and moderate values of t, the diffusion time of Hybrid Diffusion is close to the best possible diffusion time for the class of Path Verification protocols.

5.3 Computation time

The only significant computational step in our protocols is that of finding a satisfying set from among the proposals held by a given host. As we show below, this problem is NP-complete.

Recall, a set of proposals Π contains a satisfying set iff it contains a set of proposals with the same update and disjoint gossip paths. Define $\Pi|_u$ to be the set of proposals contained in Π that have update u. We therefore consider the problem of finding a set of $t + 1$ mutually disjoint proposals in $\Pi|_u$, for some update u.

We can reduce the $t + 1$ independent set problem to the disjoint proposal problem described above. The $t + 1$ independent set problem is the problem of determining whether a given undirected graph $G = (V, E)$ has a subset of $t + 1$ vertices V' such that for every pair of nodes $v, v' \in V'$, the edge (v, v') is not in E.

Given a graph G, we construct a corresponding set of proposals as follows: For every vertex $v \in V$, associate a proposal π_v. Edges serve as hostnames, and the gossip path of the proposal π_v will contain every edge that is incident on vertex v. Accordingly, a pair of proposals are disjoint iff they do not share an edge. A set of mutually disjoint proposals, therefore, corresponds to an independent set.

Although finding independent sets is NP-complete, actual computational load is quite manageable for small values of t and for the moderate number of proposals stored by each host. We were able to simulate 10,000 nodes on a 700MHz Pentium III machine, which required running 10,000 instances of the independent-set problem for each round of the simulation. This computational problem does, however, limit the usefulness of this approach for large values of t.

5.4 Distribution of multiple updates

The performance analysis so far applies only to distributing a single update. Multiple updates can be distributed by running multiple instances of a given diffusion protocol. Section 6 discusses the problem of how to control the number of protocol instances. Below, we consider how the performance of our diffusion protocols scales with the number of concurrent instances.

Diffusion time for one protocol instance is unaffected by other instances running concurrently, so diffusion time is unaffected by the number of instances. The main computational load comes from the search for a satisfying set. Each protocol instance has a different set of proposals to search, so this cost cannot be shared. Accordingly, computation time scales linearly with the number of instances. The scaling of host load and message size with respect to the number of protocol instances depends on the implementation. Multiple instances can each initiate their own separate gossip messages, in which case message size will remain unchanged, but host load will increase linearly. Alternately, multiple instances can share the same gossip messages, leaving host load unchanged but message size increased.

6 Related work and open problems

Few of the gossip protocols for disseminating updates or messages [1,13,5,16,15,12] tolerate malicious failures. Those that do allow for malicious failures, such as [15], generally assume the availability of unforgeable signatures.

The table in Fig. 11 summarizes the performance of those gossip-based protocols that have been developed to address the Diffusion Problem without the use of unforgeable signatures. The first such protocols were described by Malkhi, Mansour and Reiter in [9]. Their analysis was restricted to the class of direct verification protocols, and they presented lower bounds on the performance of that class of protocols. These lower bounds showed that, although it is possible to trade off between host load and diffusion time, the product of these two performance metrics is necessarily high. They also presented a protocol called ℓ-Tree-Random Diffusion that achieves different tradeoffs between host load and diffusion time for different values of ℓ.

Our Path Verification protocol class is not governed by the analysis of [9]. The first two columns of Fig. 11 compare the performance of two of the direct verification protocols discussed in [9]: Direct Diffusion, where gossip partners are chosen uniformly and at random, and Tree-Random, which is just ℓ-Tree-Random where ℓ has been chosen equal to $4t$, so as to maximally optimize diffusion time at the expense of

[13] Note, this means that optimal performance in Fig. 9 is being defined relative to pull-uniform Path Verification protocols.

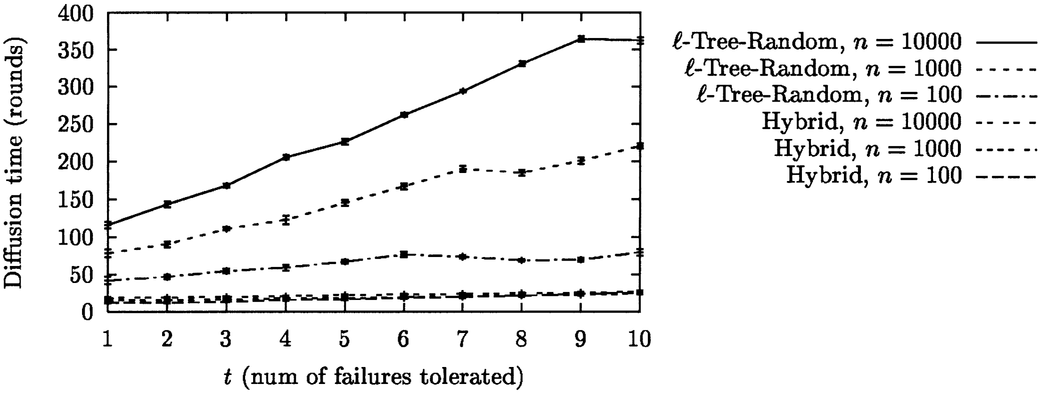

Fig. 10. Graph of diffusion time for Hybrid Diffusion with Bundle Sampling and for the ℓ-Tree-Random protocol with $\ell = 4t$

host load. The graph in Fig. 10 compares the diffusion time of the Tree Random protocol with that of Hybrid Diffusion. As the graph shows, the diffusion time of Tree-Random is significantly worse than that of Hybrid Diffusion, and the gap in performance grows with n.

Concurrently with and independent from our research, Malkhi, Pavlov and Sella [10] also developed a Path Verification protocol. That protocol keeps all proposals with length less than $\log(n/(t+1))$; longer proposals are discarded. We refer to this protocol as Short-Path Diffusion, and its performance is summarized in the third column of Fig. 11. A strength of the work is an asymptotically optimal $O(\log n{+}t{+}1)$ bound on diffusion time.[14] However, Short-Path Diffusion requires a large number of proposals to be stored and transmitted by each host – in particular, the size of messages carrying proposals is bounded from above by

$$\left(\frac{n}{t+1}\right)^{O(\log(t+1+\log n))} . \tag{6.1}$$

Thus, the number of proposals stored per host grows faster than any polynomial in the number n of hosts. Explicit bounds on message size are not derived in [10], but if we assume a constant of 1 to transform the asymptotic complexity in (6.1) to a concrete complexity, then for $n = 1000$ and $t = 1$, an individual host might have to store as many as 600,000 proposals. Increase n by a factor of 10, to $n = 10000$, and the number of proposals rises by a factor of nearly 1000. (By way of contrast, the number of proposals stored in our protocol is constant in n.) Given that finding a satisfying set is NP-complete, these protocols do not appear to be practical.

Malkhi, Reiter, Rodeh and Sella [11] also developed a class of direct verification diffusion protocols that are not strictly speaking gossip protocols because communication is done deterministically. The protocols are based on the division of the system into a logical tree-structure, where each node in the tree represents a set of $\ell \geq 2t+1$ hosts. Updates are propagated along this tree through deterministic communication between hosts in neighboring nodes. A similar tree structure is used in the Tree-Random protocol, and the performance of the resulting protocols is largely the same as that of Tree-Random, except in terms of host load. The performance of this protocol is therefore summarized in column 3 of Fig. 11, which is also used to summarize the performance of Tree-Random.

The key issue in these protocols is how to ensure that some single node ends up with at least $2t+1$ hosts that know the update in question. Two approaches are proposed. The first is to require that the sets of hosts that might act as source hosts for a given update be known *a priori*, and that there be few such sets. (Malkhi *et al.* argue that this assumption is appropriate to Byzantine quorum systems.) The nodes of the tree structure are chosen to ensure that every possible set of source hosts overlaps with some node of the tree in at least $2t+1$ hosts. The second approach is to use the content of the update in question along with a pseudorandom function to select a node of the tree for the source hosts to target. This latter approach performs well when the rate at which updates are introduced is low.

The use of an explicit tree structure in the above algorithm sacrifices some of the benign fault-tolerance of gossip. In particular, the crash-failure or partition of any single node of this tree would disconnect the system, even if the source hosts themselves have not crashed or become partitioned. Furthermore, simulations [11] show diffusion times somewhat inferior to the diffusion times of ℓ-Tree-Random (although with much lower host load). As shown in Fig. 10, our protocols significantly outperform the ℓ-Tree-Random protocols in terms of diffusion time.

Controlling the number of protocol instances

As noted in Sect. 5, the distribution of multiple updates requires execution of multiple concurrent instances of the given diffusion protocol. How are such instances brought into existence, and how are they terminated? If the number of protocol instances is not controlled then a malicious host could launch a denial of service attack by creating new instances or preventing existing instances from terminating. Moreover, neither the related work in Fig. 11 nor the protocols presented in this paper have explicit mechanisms for initiation or termination. While a full analysis of these issues is beyond the scope of this paper, we nonetheless present some preliminary ideas below.

One simple approach to controlling the number of protocol instances would be to schedule the creation and termination of protocol instances in advance. Each protocol instance would

[14] The techniques used to obtain these bounds unfortunately do not appear to apply to our protocols, and obtaining analytic bounds for the diffusion time of our protocols remains an open question.

	Direct Diffusion	Tree Random[15]	Short-Path Diffusion	Hybrid w/Bundle Sampling
Diffusion time	$\Omega\left(t\left(\frac{n}{k}\right)^{(1-1/2t)}\right)$	$\Omega(t\log(n/t))$	$O(\log n + t)$	$\sim O(\log n) + t + c$[16]
Host Load	$O(1)$	$O(n/t)$, $O(1)$[17]	$O(1)$	$O(1)$
Message Size	$O(1)$	$O(1)$	$\xi(n,t)$[18]	$30t \cdot O(\log n)$
Storage	$O(t)$	$O(t)$	$\xi(n,t)$	$30t \cdot O(\log n)$
Comp. Time	$O(\log t)$	$O(\log t)$	$\Omega\left(\left(\frac{\xi(n,t)}{\log(n/b)}\right)^t\right)$	$O(t^t + t\log n)$

Fig. 11. Performance comparison of different diffusion protocols. Each measure is per host per round in the worst case. All metrics are relative to the distribution of a single update. In all of these protocols, diffusion time and host load do not change with the number of updates being distributed concurrently, whereas the other metrics all increase linearly with the number of updates

[15] Includes both the ℓ-Tree Random protocol for $\ell = 4t$ as well as the tree-based deterministic algorithms of [11].

[16] The $\sim$ indicates an experimental and not analytic result. Our experiments show diffusion time within a small number c of rounds of the lower bound t plus the diffusion time in the absence of malicious faults, which is $O(\log n)$.

[17] Results shown for protocols in [9] and [11] respectively.

[18] $\xi(n,t)$ is equal to $(n/(t+1))^{O(\log(t+1+\log n))}$. This term is not exponential but does grow faster than any polynomial in n.

begin at a predefined time and run for a bounded period. The initiation time of the various instances would have to be staggered in order to bound the total number of concurrent executions; the amount of time alloted to each instance determines the probability that that instance successfully distributes its update to all hosts. Note, this scheme relies on synchronized clocks.

The synchronized clocks used in the above scheme are not strictly necessary. Consider a diffusion protocol used to broadcast messages from a given host H. In this approach, two protocol instances execute concurrently. In order to initiate a broadcast, H would recruit a set of at least $2t+1$ hosts to act as source hosts for the first instance of the protocol. Once enough time had elapsed so that the proposal is likely to have been distributed to all hosts, H would use the second instance of the diffusion protocol to terminate the first instance and, at the same time, to deliver a new message. After a further timeout, H could initiate a third instance to terminate the second, and so on. Higher throughput can be achieved with a larger collection of simultaneous protocol instances.

Another approach to controlling the number of protocol instances is to avoid termination altogether. This makes sense in a context where, rather than distributing a single update to all hosts, the goal of the diffusion protocol is to distribute recent values of some changing source of information. A variety of such gossip protocols have been considered in the context of benign failures [8,5,6,16,15]. Perhaps the simplest example is the failure detector of [16]. There, each host has a *heartbeat counter*, which is an integer counter that is incremented every round. A gossip protocol is used to distribute each host's heartbeat counter, with hosts storing only the largest received value of the other hosts' heartbeat counters. H marks J as faulty if H's knowledge of J's heartbeat counter is not updated within a given timeout. Thus, every value of a given host J's counter need not be delivered to every other host – it suffices that each host periodically receives recent versions of every other host's heartbeat counter. Since newer versions of a host's heartbeat counter displace older versions, no explicit termination mechanism is required.

Unfortunately, the Diffusion Problem as defined in Sect. 2 does not allow source hosts to change the updates they are distributing. As such, the Diffusion Problem is not amenable to the kind of termination-free protocol just described. But the diffusion protocols presented in this paper can be extended to handle changing updates by employing a means for combining proposals that contain different updates. Although we do not develop this approach further here, we believe some of the most promising applications of the algorithms described in this paper involve termination-free gossip protocols.

When the number of concurrent protocol instances is bounded, must a cost be incurred for all instances allowed by that bound? Our protocols as well as Malkhi *et al.*'s Path Verification protocol [10] have the same worst case cost independent of whether uncorrupted hosts use every protocol instance. This is because corrupted hosts can always forward proposals to uncorrupted hosts. These proposals will then be forwarded on, consuming resources as they go. In the case of Direct Verification, however, a host will not transmit an update until it has accepted that update. As such, t or fewer corrupted hosts cannot effectively increase message size. Storage and computation time, however, do increase with the bound on the number of protocol instances.

	Direct Diffusion	Tree Random	Path Verification
Message Size[**]	α	α	β
Storage	β	$\geq \alpha \wedge\ \leq \beta$[*]	β
Comp. Time[**]	α	$\geq \alpha \wedge\ \leq \beta$[*]	β

Fig. 12. Cost multiples associated with distributing multiple updates. β corresponds to the bound on the number of concurrent updates, and α corresponds to the actual number of updates being distributed by t or more hosts

[*] The amount of storage and computation is multiplied by β for all hosts in blocks adjacent to blocks containing corrupted hosts, and α for all other hosts. The number of hosts in blocks adjacent to blocks with corrupted hosts can be as large as $3t\ell$, where ℓ is at least $4t$.

[**] The values for message size and computational time apply in rounds when a host's gossip partner is uncorrupted. When a host's gossip partner is corrupted, the values are all β.

Figure 12 shows how the performance metrics of Fig. 11 are multiplied for a given bound β on the number of protocol instances and a given number α of protocol instances that are actually used by uncorrupted hosts to distribute an update. Host load and diffusion time are not included in Fig. 12 because these are unaffected by the number of protocol instances. The column for Path Verification protocols includes both our Hybrid Diffusion with Bundle Sampling and Malkhi *et al.*'s Shortest Path Diffusion, since they have the same behavior in this respect.

Acknowledgements. We gratefully acknowledge suggestions on an earlier draft of this paper from Robbert van Renesse, Dahlia Malkhi, and Alan Demers. The reviewers' comments were also very helpful.

References

1. Adams R (1987) RFC1036: Standard for interchange of USENET messages, December 1987.
2. Demers A, Greene D, Hauser C, Irish W, Larson J, Shenker S, Sturgis H, Swinehart D, Terry D (1987) Epidemic algorithms for replicated database maintenance. In: *Proc. of the Sixth ACM Symp. on Principles of Distributed Computing*, pp 1–12, Vancouver, British Columbia, August
3. Feller W (1968) *An Introduction to Probability Theory and its Applications*. Wiley Series in Probability and Mathematica Statistics. New York, Wiley
4. Frieze AM, Girmmett GR (1985) The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics* 10:57–77
5. Golding RA (1992) *Weak-Consistency Group Communication and Membership*. PhD thesis, UC Santa Cruz, December. Published as technical report UCSC-CRL-92-52
6. Guo K, Hayden M, Renesse R van, Vogels W, Birman KP (1997) GSGC: an efficient gossip-style garbage collection scheme for scalable reliable multicast. Technical report, Cornell University, December
7. Karp R, Shenker S, Schindelhauer C, Vocking B (2000) Randomized rumor spreading. In: *41st Symposium on Foundations of Computer Science*, pp 565–574, Portland, Oregon, July
8. Liskov B, Ladin R (1986) Highly-available distributed services and fault-tolerant distributed garbage collection. In: *Proc. of the Fifth ACM Symp. on Principles of Distributed Computing*, pp 29–39, Calgary, Alberta, August
9. Malkhi D, Mansour Y, Reiter M (1999) On diffusing updates in a byzantine environment. In: *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, pp 134–143, Lausanne, Switzerland, October. IEEE
10. Malkhi D, Pavlov E, Sella Y (2001) Optimal unconditional information diffusion. In: *15th International Symposium on Distributed Computing*, pp 63–77, Lisbon, Portugal, October
11. Malkhi D, Reiter M, Rodeh O, Sella Y (2001) Efficient update diffusion in byzantine environments. In *20th Symposium on Reliable Distributed Systems*, pp 90–98, New Orleans, USA, October 2001. IEEE Computer Society
12. Ozkasap O, Renesse R van, Birman KP, Xiao Z (1999) Efficient buffering in reliable multicast protocols. In: *Proceedings of the First International Workshop on Networked Group Communication*, pp 188–203, Pisa, Italy, November. Berlin Heidelberg New York: Springer
13. Petersen K, Spreitzer MJ, Terry DB, Theimer MM, Demers D (1997) Flexible update propagation for weakly consistent replication. In: *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pp 288–301, Saint Malo, France, October. ACM
14. Pittel B (1987) On spreading a rumor. SIAM J Appl Math 47(1):213–224
15. Renesse R van (2000) Scalable and secure resource location. In: *Proceedings of the Hawaii International Conference on System Sciences, January 4–7, 2000*, Maui, Hawaii, IEEE
16. Renesse R van, Minsky Y, Hayden M (1998) A gossip-style failure detection service. In: Davies N, Raymond K, Seitz J (eds) *Middleware '98: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp 55–70, The Lake District, England, September. 1998. Berlin Heidelberg New York: Springer

Appendix

A Proofs from Section 4

A.1 Proof of Theorem 4.1

In the following we will show that Direct Diffusion, Path Diffusion and Hybrid Diffusion with Bundle Sampling are each instances of Path Verification (3.1).

Bundle Accumulation (4.2) requires that if host H selects host J as a gossip partner, then J must transmit its bundle bundle_J to H. To satisfy condition (1) of Path Verification (3.1), any proposal in bundle_J must be contained in $\mathcal{P}_J$. This is proved below. The following holds independent of whether Bundle Sampling is being used with Youngest Diffusion, Direct Diffusion or Hybrid Diffusion.

Lemma A.1. *If J is uncorrupted and $(\pi, \mathit{sAge}) \in \mathit{bundle}_J$, then $\pi \in \mathcal{P}_J$.*

Proof. We proceed by induction in the round number. The base case is trivial, since initially bundle_J is empty. We now consider the induction case. Assume that the theorem holds for all rounds before round r. Assume that J is uncorrupted and $(\pi, \mathit{sAge}) \in \mathit{bundle}_J$ in round r. We must show that $\pi \in \mathcal{P}_J$.

By inspection of Bundle Accumulation (4.2), there are three cases:

Case 1. $(\pi, \mathit{sAge}) = (\pi_J, 0)$

Since $\pi = \pi_J$, we must show that π_J is in $\mathcal{P}_J$. Assume π_J is chosen by Youngest Diffusion. Since Youngest Diffusion satisfies Path Verification, it follows that $\pi_J \in \mathcal{P}_J$. If Bundle sampling is used to samples values of d_J, then the corresponding case is $(\pi, \mathit{sAge}) = (d_J, 0)$, and the rest of the argument remains the same.

Case 2. (π, sAge) was in bundle_J at the previous round.

Since J is uncorrupted, from the induction hypothesis we can conclude that $\pi \in \mathcal{P}_J$.

Case 3. There exists some π' such that π' is in bundle_K and $\pi' :: K = \pi$.

In this case, π' was obtained from K. Therefore, by part (i) of the definition of $\mathcal{P}_J$ in Path Verification (3.1), $\pi' :: K \in \mathcal{P}_J$.

The proposals collected in $\mathcal{B}_H$ by Bundle Sampling are used to determine whether host H can find a satisfying set in order to accept an update. To satisfy condition (2) of Path Verification, the proposals contained in $\mathcal{B}_H$ must be in $\mathcal{P}_H$ as well. This is shown in the following lemma.

Lemma A.2. *If H is uncorrupted and $(\pi, sAge) \in$ bundle and bundle $\in \mathcal{B}_H$, then $\pi \in \mathcal{P}_H$.*

Proof. If *bundle* $\in \mathcal{B}_H$, then by inspection of Bundle Sampling (4.3), *bundle* $=$ *bundle*$_J$ :: J, and *bundle*$_J$ was obtained by H from J in some previous round. By part (i) of the definition of $\mathcal{P}_H$ in Path Verification (3.1), we conclude that $\pi \in \mathcal{P}_H$.

Lemma A.1 and A.2 together imply that Bundle Sampling does not falsify conditions (1) or (2) of Path Verification. Thus, Direct Diffusion, Youngest Diffusion and Hybrid Diffusion with Bundle Sampling satisfy Path Verification, so Theorem 4.1 holds.

A.2 Proof of Theorem 4.2

Theorem A.3. *Let f be the number of corrupted hosts, and let H be an uncorrupted host. If $0 \leq a \leq \mathsf{SA}$ and $a \leq r$, then the expected number of uncorrupted samples with sample age a in bundle$_H$ at round r is $(2 - f/n)^a$, and the actual number of uncorrupted samples of sample age a never exceeds 2^a.*

Proof. Let J be H's gossip partner in round r. If H is uncorrupted, define $S(a, H, r)$ to be the number of uncorrupted samples *bundle*$_H$ at the beginning of round r with exactly sample age a. Define J_r to be H's gossip partner in round r. Then, from Bundle Accumulation (4.2), we can conclude that

$$S(0, H, r) = 1 \tag{A-1}$$

$$S(a, H, r) = \begin{cases} S(a-1, H, r-1) + \\ \quad S(a-1, J_r, r-1) & \text{if } J_r \text{ is uncorrupted} \\ S(a-1, H, r-1) & \text{otherwise} \end{cases} \tag{A-2}$$

Equation (A-2) only holds if $0 < a \leq \mathsf{SA}$ and $0 < r$. From this recurrence, induction can be used to show that $S(a, H, r) \leq 2^a$, proving that the absolute number of uncorrupted samples of age a held by a host never exceeds 2^a.

The probability that J_r is corrupted is f/n. By the properties of expectation, we can conclude that

$$\mathcal{E}(S(0, H, r)) = 1 \tag{A-3}$$

$$\mathcal{E}(S(a, H, r)) = \mathcal{E}(S(a-1, H, r-1)) + (1 - f/n)\mathcal{E}(S(a-1, J_r, r-1)) \tag{A-4}$$

If $0 \leq a \leq \mathsf{SA}$ and $a \leq r$, it follows that $\mathcal{E}(S(a, H, r)) = (2 - f/n)^a$. Note that this does not depend on H or r.

B Proofs from Section 5

Proofs in this section require a formalization system execution. Define a *system state* to be a mapping that associates a value with each state variable (*e.g.*, π_H, *bundle*$_H$, $\mathcal{B}_H$). Define a run to be a mapping from a round to a system state – the state that exists at the start of that round. The value of a variable x of a host H at a round r of a run $\mathcal{R}$ is denoted in terms of these abstractions as $\mathcal{R}[r].x_H$.[19]

In the protocols described so far, the choice of gossip partners is the only nondeterminism in the execution of uncorrupted hosts. We represent these gossip partner choices by a *partner choosing* $\mathcal{PC}$, which is a mapping such that $\mathcal{PC}(H, r)$ is H's gossip partner in round r.

B.1 Bounding age_H

In the following, age_H is proved to be bounded by the diffusion time of simple diffusion (*i.e.*, Direct Diffusion with $t = 0$). This result is used in Sect. 5.1 to bound the length of π_H.

Define a *gossip sequence* to be a series of gossips occurring in a given run of a gossip protocol, listed in increasing round order. The host receiving information in the ith gossip is the host sending information in the $i + 1$st gossip. Thus, a gossip sequence corresponds to a path through which information could be transmitted by the gossip protocol. Let the *initial round* of a gossip sequence be the round at which the first gossip in the sequence occurred, and the *final round* be the round at which the last gossip in the gossip sequence occurs. The *age* of a gossip sequence is the number of rounds since the gossip sequence's initial,l round. A gossip sequence is *uncorrupted* if all hosts along the gossip sequence are uncorrupted.

The following theorem shows that the age of any uncorrupted gossip sequence leading from a source host to H serves as an upper bound on age_H.

Theorem B.1. *For a given round r, if there is an uncorrupted gossip sequence σ leading from some source host to host H, such that σ's final round is less than r, then age_H is no greater than the age of σ at round r.*

Proof. We proceed by induction over r. The theorem holds trivially for $r = 0$, since there can be no gossip sequence whose final round is less than 0.

Assume that the theorem holds for round $r - 1$, And let a be the age of σ. Note that a must be at least 1, since round r is presumed to follow the final round of σ, so the age of σ must be at least 1. For each of the following cases, we will show that $age_H \leq a$.

$a = 1$: Since σ has age 1 and it's final round is $r - 1$, σ must consist of a single gossip in round $r - 1$ between H and some source host J. Since σ is uncorrupted, both H and J must be uncorrupted. Therefore, age_J is 0, and, by line (3.2) of Youngest Selection (3.3), the value of age_H must be 1 in round r. Thus, $age_H = a$, so trivially $age_H \leq a$.

$a > 1$: The last gossip in σ must have occurred in some round $r' < r$. We consider two cases:

$r' < r - 1$: Since $r' < r - 1$, round $r - 1$ occurs after the final round of σ. In round $r - 1$, the age of σ is

[19] As a shorthand, if $\mathcal{E}(v_1, \ldots, v_j)$ is an expression in the variables $\{v_1, \ldots, v_j\}$, we write $\mathcal{R}[r].\mathcal{E}(v_1, \ldots, v_j)$ to signify $\mathcal{E}(\mathcal{R}[r].v_1, \ldots, \mathcal{R}[r].v_j)$.

$a-1$. From the induction hypothesis it follows that $age_H \leq a-1$ holds at round $r-1$. By inspection of Youngest Selection, age_H cannot increase by more than 1 per round, so $age_H \leq a$ holds at round r.

$r' = r-1$: Define σ' to be the gossip sequence σ with its final gossip removed. Let J be the host receiving information in the final gossip in σ'. Since σ is uncorrupted, J must be uncorrupted as well. The final gossip in σ must consist of H receiving information from J. Since $r' = r-1$, that gossip occurs in round $r-1$. Since the last gossip in σ occurs in round $r-1$, the last gossip in σ' must occur in some round before $r-1$. Since the age of σ is a in round r, the age of σ' in round $r-1$ must be $a-1$, since σ and σ' share the same initial round. By the induction hypothesis, age_J must be no greater than $a-1$ in round $r-1$. By line (3.2) of Youngest Selection, age_H in round r is no more than 1 greater than the age_J in round $r-1$. Thus, $age_H \leq a$ holds in round r.

In Direct Diffusion with $t=0$, host H accepts update u in round r iff there is an uncorrupted gossip sequence leading from some source to H. Thus, if all hosts accept an update within a rounds, then for every host there is an uncorrupted gossip sequence leading from some source host to H with age no more than a. By Theorem B.1, this implies that for a run of Youngest Selection with the same partner choosing $\mathcal{PC}$, age_H will be bounded by a for every host H. Thus, the diffusion time of Direct Diffusion with $t=0$ serves as a bound on age_H.

B.2 *Proof of worst case behavior*

We prove that worst-case behavior described in Sect. 5.2 produces the largest possible diffusion time. Recall that this worst case behavior has corrupted hosts acting as source hosts that have accepted the wrong update $\hat{u}$ (instead of the initial update u). In addition, if the sampling protocols of Sect. 4 are in use, then corrupted hosts in the worst case do not store or transmit samples from other hosts.

Informally, this behavior is worst because, by acting as source hosts with update $\hat{u}$, a corrupted host displaces proposals originating from uncorrupted hosts, and because the most a corrupted host can do to slow down Bundle Sampling is to not take part.

Let $\mathcal{R}$ be some run of a Path Verification protocol with partner choosing $\mathcal{PC}$ and set $\mathcal{F}$ of corrupted hosts. In run $\mathcal{R}$, no assumptions are made about the behavior of corrupted hosts. Let $\hat{\mathcal{R}}$ be the worst case run with partner choosing $\mathcal{PC}$ and set $\mathcal{F}$ of corrupted hosts, *i.e.*, a run where the corrupted hosts follow the worst-case behavior described above. The theorem we seek is that if a host H accepts an update in round r in the worst-case run $\hat{\mathcal{R}}$, then H accepts the update by round r in $\mathcal{R}$. Instead of proving this theorem, we prove strengthening WC(r), defined as follows.

(B.1) WC(r') holds iff, for all $r \leq r'$: $\mathrm{WC}_1(r) \wedge \mathrm{WC}_2(r) \wedge \mathrm{WC}_3(r) \wedge \mathrm{WC}_4(r)$,
where:

$\mathrm{WC}_1(r)$: If H is uncorrupted, then $\hat{\mathcal{R}}[r].\pi_H = \bot \Rightarrow \mathcal{R}[r].\pi_H = \bot$

$\mathrm{WC}_2(r)$: $\hat{\mathcal{R}}[r].age_H \leq \mathcal{R}[r].age_H$.

$\mathrm{WC}_3(r)$: If $\hat{\mathcal{R}}[r].\pi_H$ has update u, then $\hat{\mathcal{R}}[r].\pi_H = \mathcal{R}[r].\pi_H$ and $\hat{\mathcal{R}}[r].age_H = \mathcal{R}[r].age_H$.

$\mathrm{WC}_4(r)$: If H is uncorrupted and H accepts u at round r of run $\hat{\mathcal{R}}$ then H has accepted u no later than round r of run $\mathcal{R}$.

We now prove that WC(r) holds for all rounds r, for both Youngest Diffusion (3.5) and Hybrid Diffusion (3.6), both as originally described and with Bundle Sampling (4.3). Statements about variables that don't appear in a given protocol in question are considered to hold trivially. This is harmless, since variables that do not appear can have no affect on whether an update is accepted.

Lemma B.2. $\mathrm{WC}_1(r-1) \Rightarrow \mathrm{WC}_1(r)$.

Proof. Assume that H is uncorrupted, and that WC$(r-1)$ and $\hat{\mathcal{R}}[r].\pi_H = \bot$ hold. We must prove that $\mathcal{R}[r].\pi_H = \bot$.

From the initial conditions of Youngest Selection (3.3) and the fact that $\hat{\mathcal{R}}[r].\pi_H = \bot$, it follows that H is not a source host. Let $J = \mathcal{PC}(H, r)$. By line (3.2) of Youngest Selection, if either $\hat{\mathcal{R}}[r-1].\pi_H$ or $\hat{\mathcal{R}}[r-1].\pi_J$ is non-$\bot$, then $\hat{\mathcal{R}}[r].\pi_H$ will be non-$\bot$ as well.[20] Thus, we conclude that:

$$
\begin{aligned}
& \hat{\mathcal{R}}[r-1].\pi_H = \bot \;\wedge\; \hat{\mathcal{R}}[r-1].\pi_J = \bot \\
\Rightarrow\; & \langle\!\langle \text{In } \hat{\mathcal{R}}, \text{ if } J \text{ is corrupted then } \pi_J \text{ is } (\hat{u}, \emptyset). \text{ Since } \hat{\mathcal{R}}[r-1].\pi_J = \bot, \text{ host } J \text{ is uncorrupted. } H \text{ is uncorrupted by assumption. We conclude by } \mathrm{WC}_1(r-1) \rangle\!\rangle \\
& \mathcal{R}[r-1].\pi_H = \bot \;\wedge\; \mathcal{R}[r-1].\pi_J = \bot \\
\Rightarrow\; & \langle\!\langle \text{by line (3.2) of Youngest Selection and because } H \text{ is not a source host} \rangle\!\rangle \\
& \mathcal{R}[r].\pi_H = \bot
\end{aligned}
$$

Lemma B.3. $\mathrm{WC}_2(r-1) \Rightarrow \mathrm{WC}_2(r)$.

Proof. Assume that $\mathrm{WC}_2(r-1)$ holds. We need to show that for any H,

$$\hat{\mathcal{R}}[r].age_H \leq \mathcal{R}[r].age_H.$$

Proof. We consider the following cases:

Case 1. H is corrupted.

By definition of $\hat{\mathcal{R}}$, corrupted hosts act as source hosts, so $\hat{\mathcal{R}}[r].age_H$ is zero. Since zero is the minimal possible age, the lemma holds trivially.

Case 2. H is uncorrupted and H is not a source.

Let J be H's gossip partner in round r. Then,

$$
\begin{aligned}
& \hat{\mathcal{R}}[r].age_H \\
=\; & \langle\!\langle \text{by line (3.2) of Youngest Selection} \rangle\!\rangle \\
& \min(\hat{\mathcal{R}}[r-1].age_H, \hat{\mathcal{R}}[r-1].age_J) + 1 \\
\leq\; & \langle\!\langle \text{By } \mathrm{WC}_2(r-1) \text{ and monotonicity of min} \rangle\!\rangle \\
& \min(\mathcal{R}[r-1].age_H, \mathcal{R}[r-1].age_J) + 1 \\
=\; & \langle\!\langle \text{by line (3.2) of Youngest Selection} \rangle\!\rangle \\
& \mathcal{R}[r].age_H
\end{aligned}
$$

[20] This follows from the definition of age_K given in Sect. 3.2, which states that, for any host K, if $\pi_K = \bot$, then age_K is defined to be ∞.

Case 3. H is an uncorrupted source.

According to Youngest Selection (3.3), age_H is equal to 0 in any run, so the lemma holds.

Lemma B.4. $\mathrm{WC}(r-1) \Rightarrow \mathrm{WC}_3(r)$.

Proof. Assume $\mathrm{WC}(r-1)$, and that $\hat{\mathcal{R}}[r].\pi_H$ has update u. We must show that $\hat{\mathcal{R}}[r].\pi_H = \mathcal{R}[r].\pi_H$ and $\hat{\mathcal{R}}[r].age_H = \mathcal{R}[r].age_H$.

We consider the following cases.

Case 1. H is a source host.

Then, in both $\hat{\mathcal{R}}$ and $\mathcal{R}$, $\pi_H = (u, \emptyset)$ and $age_H = 0$. The lemma therefore holds.

Case 2. H is not a source host and H is corrupted.

By the definition of $\hat{\mathcal{R}}$, proposal $\hat{\mathcal{R}}[r].\pi_H$ must contain update $\hat{u}$. This contradicts our assumption that $\hat{\mathcal{R}}[r].\pi_H$ has update u, so this case does not occur.

Case 3. H is not a source host and H is uncorrupted.

Let J be H's gossip partner in round r. By lines (3.2) and (3.2) of Youngest Selection, the values of π_H and age_H in round r are derived from the values in the previous round of either π_H and age_H or of π_J and age_J. More formally,

$$\begin{aligned} &(\hat{\mathcal{R}}[r].\pi_H = \hat{\mathcal{R}}[r-1].\pi_H \wedge \\ &\quad \hat{\mathcal{R}}[r].age_H = \hat{\mathcal{R}}[r-1].age_H + 1) \\ \vee\ &(\hat{\mathcal{R}}[r].\pi_H = \hat{\mathcal{R}}[r-1].\pi_J :: J \wedge \\ &\quad \hat{\mathcal{R}}[r].age_J = \hat{\mathcal{R}}[r-1].age_J + 1) \end{aligned}$$

Let K be the host among H and J from which proposal $\hat{\mathcal{R}}[r].\pi_H$ is derived, and let L be the other host. Since $\hat{\mathcal{R}}[r].\pi_H$ has update u, $\hat{\mathcal{R}}[r-1].\pi_K$ must have update u as well. By $\mathrm{WC}_3(r-1)$,

$$\begin{aligned} &\hat{\mathcal{R}}[r-1].\pi_K = \mathcal{R}[r-1].\pi_K \\ \wedge\ &\hat{\mathcal{R}}[r-1].age_K = \mathcal{R}[r-1].age_K \end{aligned}$$

By $\mathrm{WC}_2(r-1)$,

$$\hat{\mathcal{R}}[r-1].age_L \leq \mathcal{R}[r-1].age_L.$$

Thus, at round $r-1$, π_K and age_K are the same in runs $\hat{\mathcal{R}}$ and $\mathcal{R}$, and age_L is no larger in $\hat{\mathcal{R}}$ than it is in $\mathcal{R}$. Since Youngest Selection chooses in favor of smaller ages, it follows that, if π_H is derived from π_K at round r of run $\hat{\mathcal{R}}$, then π_H is derived from π_K at round r of run $\mathcal{R}$. Since π_K and age_K are the same in $\hat{\mathcal{R}}$ and $\mathcal{R}$, we can conclude that $\hat{\mathcal{R}}[r].\pi_H = \mathcal{R}[r].\pi_H$ and $\hat{\mathcal{R}}[r].age_H = \mathcal{R}[r].age_H$.

The following three lemmas show that, for the various different protocols, the proposals with update u collected via sampling in $\hat{\mathcal{R}}$ are a subset of the set of the proposals collected via sampling in $\mathcal{R}$.

Lemma B.5. *For a set Π of proposals, define $\Pi|_u$ to be the set of proposals in Π with update u. In Youngest Diffusion and Hybrid Diffusion, if H is uncorrupted and* $\mathrm{WC}(r-1)$ *holds, then*

$$\hat{\mathcal{R}}[r].\mathcal{Y}_H|_u \subseteq \mathcal{R}[r].\mathcal{Y}_H.$$

Proof. Assume that $\mathrm{WC}(r-1)$ holds, H is uncorrupted, and $\pi \in \hat{\mathcal{R}}[r].\mathcal{Y}_H|_u$. We must show that $\pi \in \mathcal{R}[r].\mathcal{Y}_H$.

Let J_r be H's gossip partner in round r. For a given run $\mathcal{R}$, define *samples*$(\mathcal{R})$ to be the sequence

$$\begin{aligned} \{&\mathcal{R}[r-1].\pi_{J_{r-1}} :: J_{r-1}, \\ &\mathcal{R}[r-2].\pi_{J_{r-2}} :: J_{r-2}, \\ &\vdots \\ &\mathcal{R}[0].\pi_{J_0} :: J_0\}. \end{aligned}$$

Thus, *samples*$(\mathcal{R})$ is the sequence of youngest proposals received by host H before round r in run $\mathcal{R}$. Not all of these samples are kept in $\mathcal{Y}_H$, however. In particular, by inspection of Simple Sampling (3.4), $\mathcal{R}[r].\mathcal{Y}_H$ contains only the most recent S non-$\perp$ elements of *samples*$(\mathcal{R})$.

By assumption, $\pi \in \hat{\mathcal{R}}[r].\mathcal{Y}_H|_u$. Therefore, there is some integer i such that the ith element of *samples*$(\hat{\mathcal{R}})$ is π and is also among the most recent S non-$\perp$ elements of *samples*$(\hat{\mathcal{R}})$. Thus,

$$\begin{aligned} &\pi \\ = \quad &(\!(\text{Since } \pi \text{ is the } i\text{th element of } \textit{samples}(\mathcal{R}).)\!) \\ &\hat{\mathcal{R}}[r-i].\pi_{J_{r-i}} :: J_{r-i} \\ = \quad &(\!(\text{From assmpt. that } \pi \text{ has update } u \text{ and } \mathrm{WC}_3(r- \\ &\quad 1).)\!) \\ &\mathcal{R}[r-i].\pi_{J_{r-i}} :: J_{r-i} \end{aligned}$$

which is the ith element of *samples*$(\mathcal{R})$. The only remaining question is whether π is among the most recent S non-$\perp$ elements of *samples*$(\mathcal{R})$. By $\mathrm{WC}_1(r-1)$, if the ith element of *samples*$(\hat{\mathcal{R}})$ is $\perp$, then the ith element of *samples*$(\mathcal{R})$ is $\perp$ as well. Thus, π must be among the most recent S elements of *samples*$(\mathcal{R})$, so $\pi \in \mathcal{R}[r].\mathcal{Y}_H$.

Lemma B.6. *If H is uncorrupted and* $\mathrm{WC}(r-1)$ *holds, then*

$$\hat{\mathcal{R}}[r].\mathcal{D}_H|_u \subseteq \mathcal{R}[r].\mathcal{D}_H$$

Proof. Assume that $\mathrm{WC}(r-1)$ holds, H is uncorrupted, and $\pi \in \hat{\mathcal{R}}[r].\mathcal{D}_H|_u$. We must show that $\pi \in \mathcal{R}[r].\mathcal{D}_H$.

By inspection of of Direct Diffusion (3.2), the fact that π is in $\mathcal{R}[r].\mathcal{D}_H$ implies that π is of the form $(u, [J])$, where $J = \mathcal{PC}(H, r')$, for some round $r' < r$. Additionally the value of $\hat{\mathcal{R}}[r'].d_J$ is $(u, \emptyset)$.

Recall that in run $\hat{\mathcal{R}}$, corrupted hosts act as if they had accepted update $\hat{u}$. Thus, if J were corrupted, $\hat{\mathcal{R}}[r'].d_J$ would be $(\hat{u}, \emptyset)$. Therefore, J must be uncorrupted. Further, J must have accepted update u by round r'. By $\mathrm{WC}_4(r-1)$, host J must have accepted update u by round r' of run $\mathcal{R}$. Therefore, $\mathcal{R}[r'].d_J$ is $(u, \emptyset)$. Since J is H's gossip partner in round r', it follows that $(u, [J])$ is in $\mathcal{R}[r].\mathcal{D}_H$.

Lemma B.7. *If H is uncorrupted and* $\mathrm{WC}(r-1)$ *holds, then, if proposal π has update u, then*

$$(\pi, \textit{sAge}) \in \hat{\mathcal{R}}[r].\mathcal{B}_H \Rightarrow (\pi, \textit{sAge}) \in \mathcal{R}[r].\mathcal{B}_H$$

Proof. We prove a strengthened version of the above theorem. In particular, we show that if π has update u, then

$$(\pi, \mathit{sAge}) \in \hat{\mathcal{R}}[r].\mathit{bundle}_H \;\Rightarrow\; (\pi, \mathit{sAge}) \in \mathcal{R}[r].\mathit{bundle}_H \tag{A-1}$$

$$(\pi, \mathit{sAge}) \in \hat{\mathcal{R}}[r].\mathcal{B}_H \;\Rightarrow\; (\pi, \mathit{sAge}) \in \mathcal{R}[r].\mathcal{B}_H \tag{A-2}$$

We proceed by induction over r. The base case is trivial, since bundle_H and $\mathcal{B}_H$ are initially empty. We therefore assume the theorem holds for round $r-1$, and that H is uncorrupted and $\mathrm{WC}(r-1)$ holds. We must then show that if π has update u, then equations (A-1) and (A-2) hold.

We will first show that equation (A-1) holds. Assume that $(\pi, \mathit{sAge}) \in \hat{\mathcal{R}}[r].\mathit{bundle}_H$, and we will show that $(\pi, \mathit{sAge}) \in \mathcal{R}[r].\mathit{bundle}_H$.

By inspection of Bundle Accumulation, since $(\pi, \mathit{sAge}) \in \hat{\mathcal{R}}[r].\mathit{bundle}_H$, it follows that $\mathit{sAge} < \mathit{bundle}_H$. Let J be H's gossip partner in round r. By inspection of Bundle Accumulation (4.2), there are three cases to consider:

Case 1. $\pi = \hat{\mathcal{R}}[r-1].\pi_H$ and $\mathit{sAge} = 0$.

By $\mathrm{WC}_3(r-1)$, we can conclude

$$\begin{aligned} &\hat{\mathcal{R}}[r-1].\pi_H \text{ has update } u \\ &\quad\Rightarrow\; \hat{\mathcal{R}}[r-1].\pi_H = \mathcal{R}[r-1].\pi_H. \end{aligned} \tag{A-3}$$

Since π has update u, it follows that $\hat{\mathcal{R}}[r-1].\pi_H = \mathcal{R}[r-1].\pi_H$. By inspection of Bundle Accumulation, $(\mathcal{R}[r-1].\pi_H, 0) \in \mathcal{R}[r-1].\mathit{bundle}_H$, so $(\pi, \mathit{sAge}) = (\pi, 0) \in \mathcal{R}[r-1].\mathit{bundle}_H$.

(If Bundle Sampling is used to collect values of d_H instead of values of π_H, then the case should be $\pi = \hat{\mathcal{R}}[r-1].d_H$ and $\mathit{sAge} = 0$. The analogue of equation (A-3) follows from $\mathrm{WC}_4(r-1)$ and the fact that if H is uncorrupted, then $d_H = (u, \emptyset)$ if H has accepted u, and $\perp$ otherwise.)

Case 2. $(\pi, \mathit{sAge}-1) \in \hat{\mathcal{R}}[r-1].\mathit{bundle}_H$

By the induction hypothesis, $(\pi, \mathit{sAge}-1) \in \mathcal{R}[r-1].\mathit{bundle}_H$. By inspection of Bundle Accumulation and the fact that $\mathit{sAge} < \mathsf{SA}$, it follows that $(\pi, \mathit{sAge}-1)$ is in $\mathcal{R}[r].\mathit{bundle}_H$.

Case 3. $(\pi', \mathit{sAge}-1) \in \hat{\mathcal{R}}[r-1].\mathit{bundle}_J$, where $\pi = \pi' :: J$ and $J = \mathcal{PC}(H, r)$.

J must be uncorrupted, since otherwise bundle_J would be empty in run $\hat{\mathcal{R}}$. By the induction hypothesis, $(\pi', \mathit{sAge}-1) \in \mathcal{R}[r-1].\mathit{bundle}_J$ must hold. By inspection of Bundle Accumulation and the fact that $\mathit{sAge} < \mathsf{SA}$, it follows that $(\pi, \mathit{sAge}-1)$ is in $\mathcal{R}[r].\mathit{bundle}_H$.

Since these cases are exhaustive, eq. (A-1) must hold in round r.

It remains to show that eq. (A-2) holds in round r. Assume that $(\pi, \mathit{sAge}) \in \hat{\mathcal{R}}[r].\mathcal{B}_H$. We must show that $(\pi, \mathit{sAge}) \in \mathcal{R}[r].\mathcal{B}_H$ holds.

By inspection of Bundle Sampling (4.3), we can conclude that for some round $r' < r$, $(\pi, \mathit{sAge}) \in \hat{\mathcal{R}}[r'].\mathit{bundle}_J$, where J is H's gossip partner in round r', where $r - r'$ is less than S. Since $\hat{\mathcal{R}}[r'].\mathit{bundle}_J$ is non-empty, it follows that J is uncorrupted. By the induction hypothesis, $(\pi, \mathit{sAge}) \in \mathcal{R}[r'].\mathit{bundle}_J$. Since $r - r'$ is less than S and J is H's gossip partner in round r', $\mathcal{R}[r'].\mathit{bundle}_J \in \mathcal{R}[r].\mathcal{B}_J$ must hold, so $(\pi, \mathit{sAge}) \in \mathcal{R}[r].\mathcal{B}_J$ holds as well. Thus, eq. (A-2) holds.

Lemma B.8. $\mathrm{WC}(r-1) \Rightarrow \mathrm{WC}_4(r)$.

Proof. Assume that $\mathrm{WC}(r-1)$ holds, that host H is uncorrupted, and that H has accepted update u at round r of $\hat{\mathcal{R}}$. We must show that H accepts update u no later than round r of run $\mathcal{R}$. By Lemmas B.5, B.6 and B.7, every proposal with update u gathered by H in run $\hat{\mathcal{R}}$ is also gathered by H in run $\mathcal{R}$. As such, if H finds a satisfying subset for update u in run $\hat{\mathcal{R}}$, it will also find a satisfying subset in run $\mathcal{R}$.

Theorem B.9. *If any host H accepts update u in round r of run $\hat{\mathcal{R}}$, then H accepts u by round r of run $\mathcal{R}$.*

Proof. $\mathrm{WC}(0)$ holds trivially, since the initial states of $\hat{\mathcal{R}}$ and $\mathcal{R}$ are identical. Lemmas B.2, B.3, B.4 and B.8 together imply that $\mathrm{WC}(r-1) \Rightarrow \mathrm{WC}(r)$, so by induction, $\mathrm{WC}(r)$ holds for all rounds r. The theorem is simply $\mathrm{WC}_4(r)$.

C Promiscuous Youngest Diffusion

As noted in Sect. 3.2, one possible variation on Youngest Diffusion would be to allow hosts that have accepted a given update to begin to act as source hosts for that update. This resulting protocol is called Promiscuous Youngest Selection. Hybrid Diffusion (3.6) is, in some sense, a compromise between Youngest Diffusion and Promiscuous Youngest Diffusion. In Youngest Diffusion, hosts only store proposals that originate at the original source hosts. In Hybrid Diffusion, hosts store proposals originating at any updated host, but only if the proposal was obtained directly from the updated host. In Promiscuous Youngest Diffusion, hosts store proposals originating from updated hosts both directly and indirectly.

Promiscuous Youngest Diffusion does not appear to have an easy to simulate worst case behavior. Without such a worst case behavior, simulations do not necessarily reveal the behavior of the protocol under Byzantine failures. We are thus reluctant to advocate this protocol for use in the presence of malicious hosts.

The problem is that partner choosings exist for which the worst case behavior discussed in Sect. 5.2 is not the worst case for Promiscuous Youngest Diffusion. Figure 13 shows one example. There, edges denote information transmitted from one host to another during a gossip, and the numbers on the edges indicate the round in which the information is transmitted.

Recall that the worst case behavior of Sect. 5.2, requires that any corrupted host M act as a source host with the wrong update. In the following, we compare what happens in Fig. 13 with that seemingly worst-case behavior and with the behavior where M acts as if it were an uncorrupted source host. Assume that $t = 2$.

Case 1. M acts like a source host with the **wrong** update $\hat{u}$.

By round 3, note that E has received only 2 proposals proposing update u, from V and W. As such, E has not accepted update u by round 6. So, proposals received by A from X in round 4 will be younger than the proposals received from E

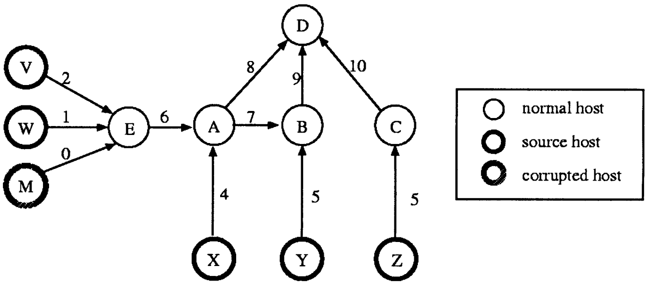

Fig. 13. One host delayed from accepting an update due to another host accepting an update

in round 6. As a result, D will receive proposals with gossip paths $[X, A]$, $[Y, B]$ and $[Z, C]$ respectively. Accordingly, D will accept update u.

Case 2. M acts like a source host with the **right** update u.

Since M acts as a source host with update u, host E will accept update u in round 2. As a result, the proposals originating from E will have age 0. Because of this, A and B will set their youngest proposals to the proposals originating from E in rounds 6 and 7 respectively. Thus, in rounds 8 through 10, host D will receive youngest proposals with gossip paths $[E, A]$, $[E, A, B]$ and $[Z, C]$. The first two intersect, so D does not have a satisfying set and cannot accept u in round r.

It is easy to extend the partner choosing shown in Fig. 13 so that D never accepts an update in Case 2 – for example, by insisting that D only chooses B as its gossip partner after round 10, and all other hosts communicate normally. This shows that at least for some partner choosings, the seeming worst-case behavior of Sect. 5.2 is not the worst-case behavior for Promiscuous Youngest Diffusion. An open problem is whether such a well-defined worst-case behavior exists for Promiscuous Youngest Diffusion.

It is notable that, as shown in the above example, a corrupted host can degrade the behavior of the protocol by acting as if it were a source host with the right update. This situation must be rare, since our simulations show that the addition of source hosts improves the expected performance of Promiscuous Youngest Diffusion. What might be less rare, however, are situations where corrupted hosts can increase diffusion times by distributing the right update specifically where it will delay acceptance of an update. Such an attack would seem, however, to depend on some advance knowledge of how hosts choose gossip partners.

One might wonder why there are no anomalous partner choosings like the one described above for Direct, Youngest, or Hybrid Diffusion. The key distinction between those protocols and Promiscuous Youngest Diffusion is that the former are monotonic in the sense that causing non-faulty hosts to accept the initial update u can only reduce the time until all hosts have accepted u. The partner choosing described above makes it clear that Promiscuous Youngest Diffusion is not monotonic in this sense. The monotonicity of Direct Diffusion follows from the fact that a host accepts an update u once it gossips with $t + 1$ hosts that claim to have accepted u. As such, increasing the number of hosts that have accepted u can only hasten the time at which other hosts will accept u. Youngest Diffusion is monotonic simply because a host that has accepted an update does not change its external behavior in any way – in particular, it does not start behaving as if it were a source host. Hybrid Diffusion is simply the combination of Youngest and Direct Diffusion, and since causing hosts to accept u does not slow down either of the constituent protocols, it does not slow down the combined protocol either.

It may not be necessary to have a single well-defined worst-case behavior for all possible partner choosings. There may be a statistical sense in which the worst-case behavior of Sect. 5.2 is the worst case for Promiscuous Youngest Diffusion, particularly when corrupted hosts are assumed unable to predict the partner selections of other hosts. How to construct such an analysis is a topic for further research.